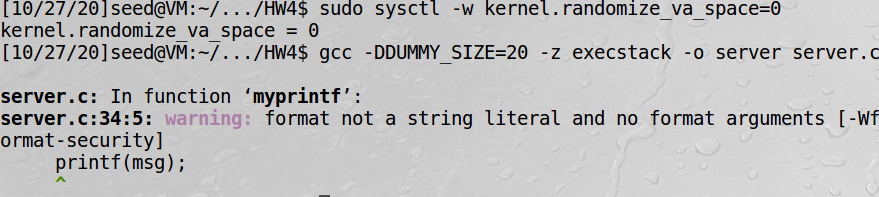
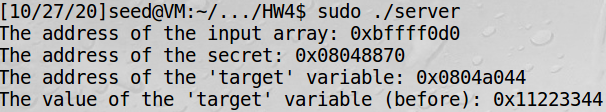
**Task 2.1 –** The Vulnerable Program

1st Terminal

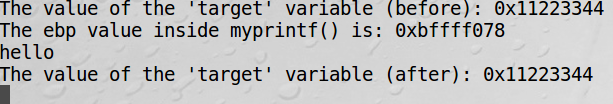




2nd Terminal



1st Terminal



2nd Terminal



1st Terminal



**Task 2.2 –** Understanding the Layout of the Stack

Question 1: What are the memory addresses at the locations marked by ➊, ➋, and ➌?

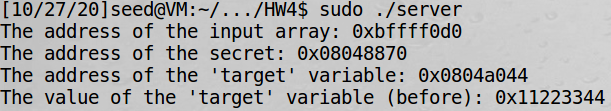
-Answer: ➊ = 0xbffff050, ➋ = 0xbffff078, ➌ = 0xbffff0d0

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

-Answer: 0xbffff0d0 - 0xbffff050 = 0x80 =128bytes, Distance (# of %x) = 128/4 = **32**

**Task 2.3 –** Crash the Program

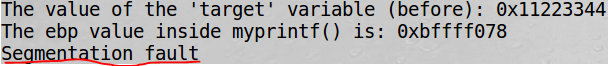
1st Terminal



2nd Terminal



1st Terminal

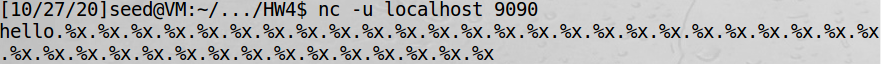


-We are able to crash the program by using the %s format specifier.

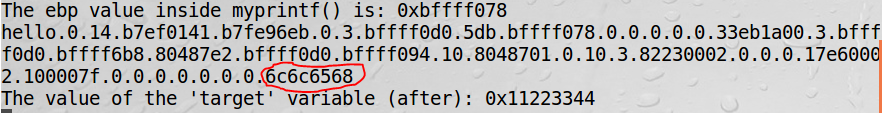
**Task 2.4 –** Print Out the Server Program’s Memory

**4.A** – Stack Data

2nd Terminal



1st Terminal



-As you can see, I was able to print out the first 4 bytes of my input by following it with 40 %x’s.

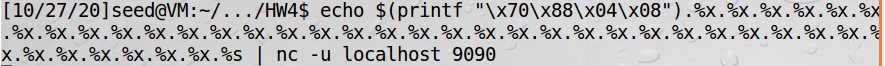
**4.B –** Heap Data

Address of secret message given:

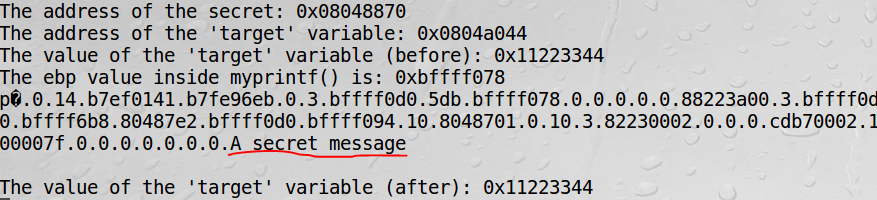


2nd Terminal





1st Terminal

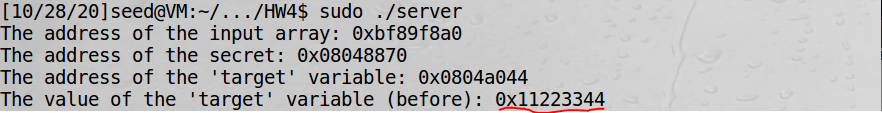


-Here, we are able to print out the secret message by echoing the address of the secret message followed by 39 %x’s and one %s. This prints out “A secret message”, as shown above.

**Task 2.5 –** Change the Server Program’s Memory

**5.A** – Change the value to a different value

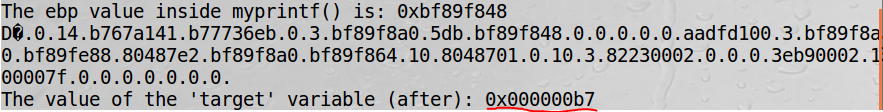
1st Terminal



2nd Terminal



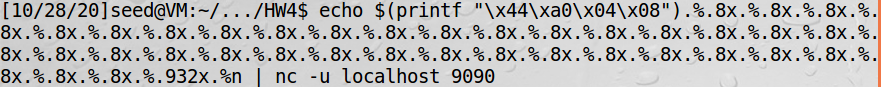
1st Terminal



-Here, you can see that I was able to change the value of the target variable from being 0x11223344 to being 0x000000b7 by printing the *address* of the target variable followed by 39 %x’s and one %n.

**5.B** – Change the value to 0x500

2nd Terminal



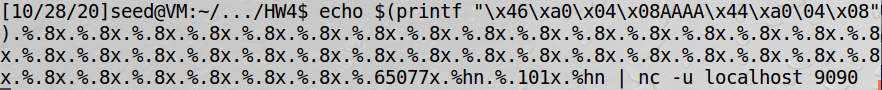
1st Terminal



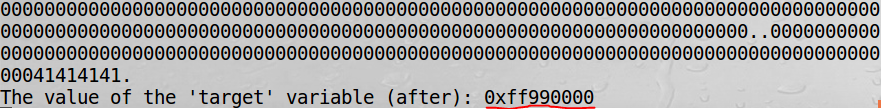
-Here, we do a similar attack as above, except we just add more data being printed (%.8x’s instead of regular %x’s and the final %.932x). This changes the value of the target variable to 0x00000500.

**5.C** – Change the value to 0xFF990000

2nd Terminal



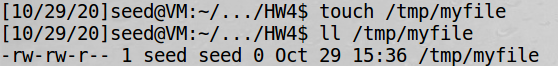
1st Terminal



-Here, we do the same attack as above, but with *even more* data being printed. We also use %hn instead of %n. For this big of a value, we had to break it up into 2 numbers (the ff99 and the 0000). For the 0000, we place data at the address 0x0804a044 (seen above). This took a good bit of trial and error and my answer is not exactly the same as Dr. Cheng’s. Finally, I was successful in writing 0000 to the last 4 numbers of the target variable value. Then I tried a similar number from Dr. Cheng’s slides to write the ff99 (his number was 65433) and adjusted it accordingly until I got the right one. This resulted in the value of the target variable being changed to 0xff990000.

**Task 2.6 –** Inject Malicious Code into the Server Program

1st Terminal (myfile exists)



-We can see here that the file /tmp/myfile exists.

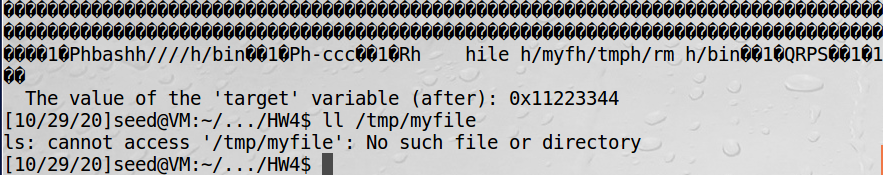
-Then I run exploit.py that writes malicious code to badfile…

2nd Terminal



-Then I send the contents of badfile to localhost…

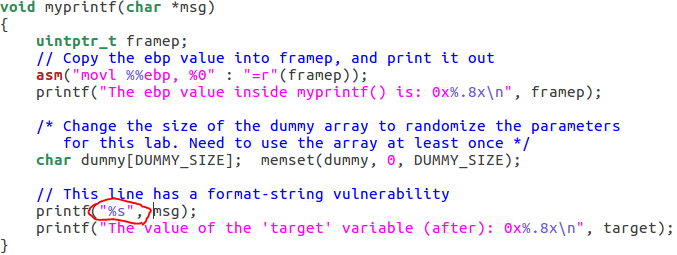
1st Terminal



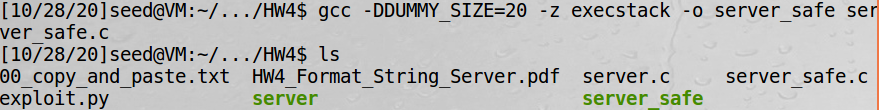
-And finally, here we can see that upon completion of the program, it has deleted the file /tmp/myfile.

**Task 2.8 –** Fixing the Problem

-I fixed the vulnerability in the server.c file by adding a string literal (“%s”) in the myprinf() function, as shown below.



-Once you do this, the compiler warning **does** go away, as shown below. When I recompile the program, there is now no warning.

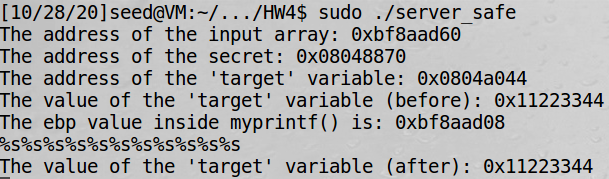


-Also, I tried my “Crash the program” attack again (in task 2.3) and it was unsuccessful in causing a segmentation fault. The server program continues running with no problems after I tried this attack. This behavior can be seen in the screenshots below.

2nd Terminal



1st Terminal (after attack)



-The “Crash the program” attack from task 2.3 had no affect on the safe version of the server program.