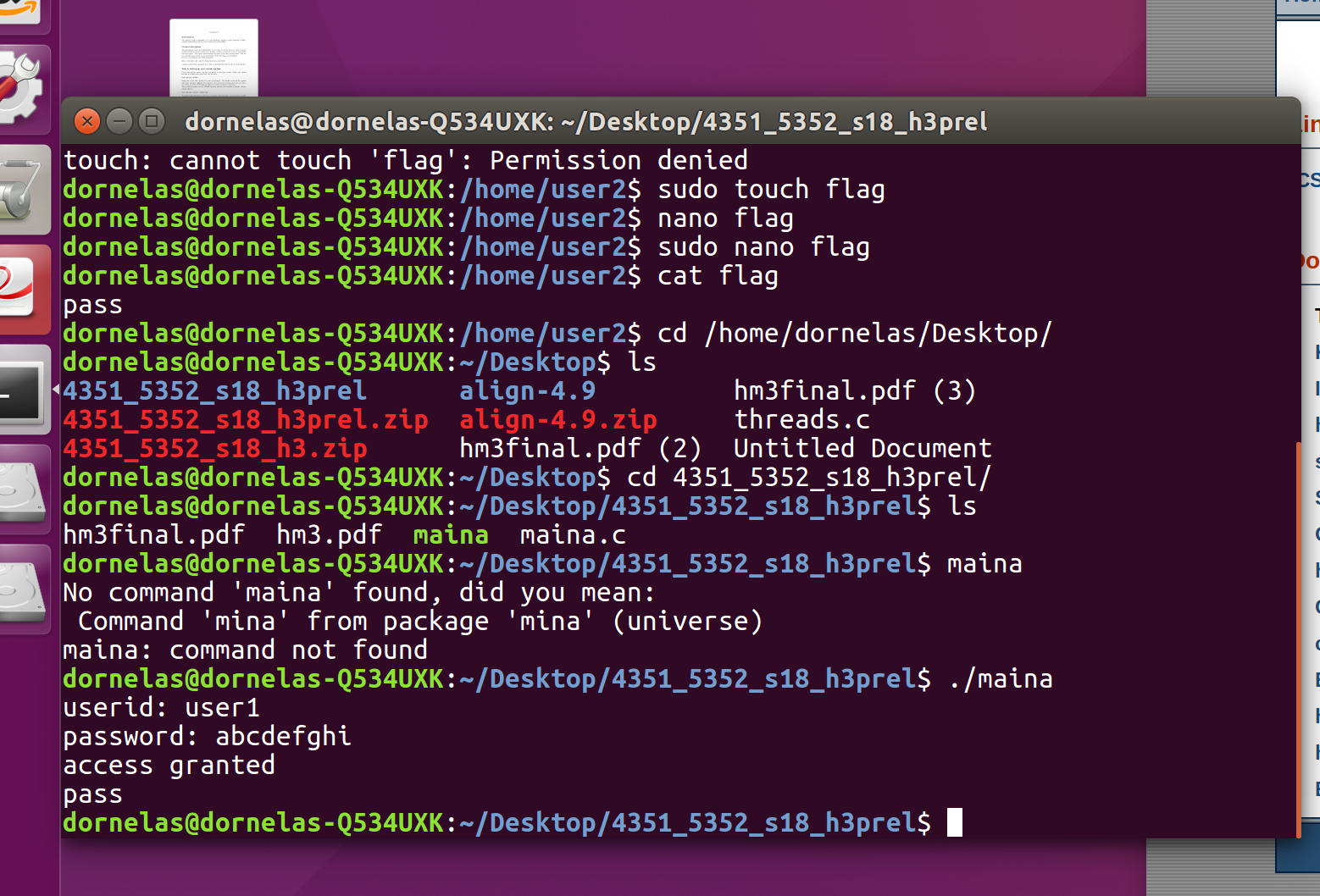
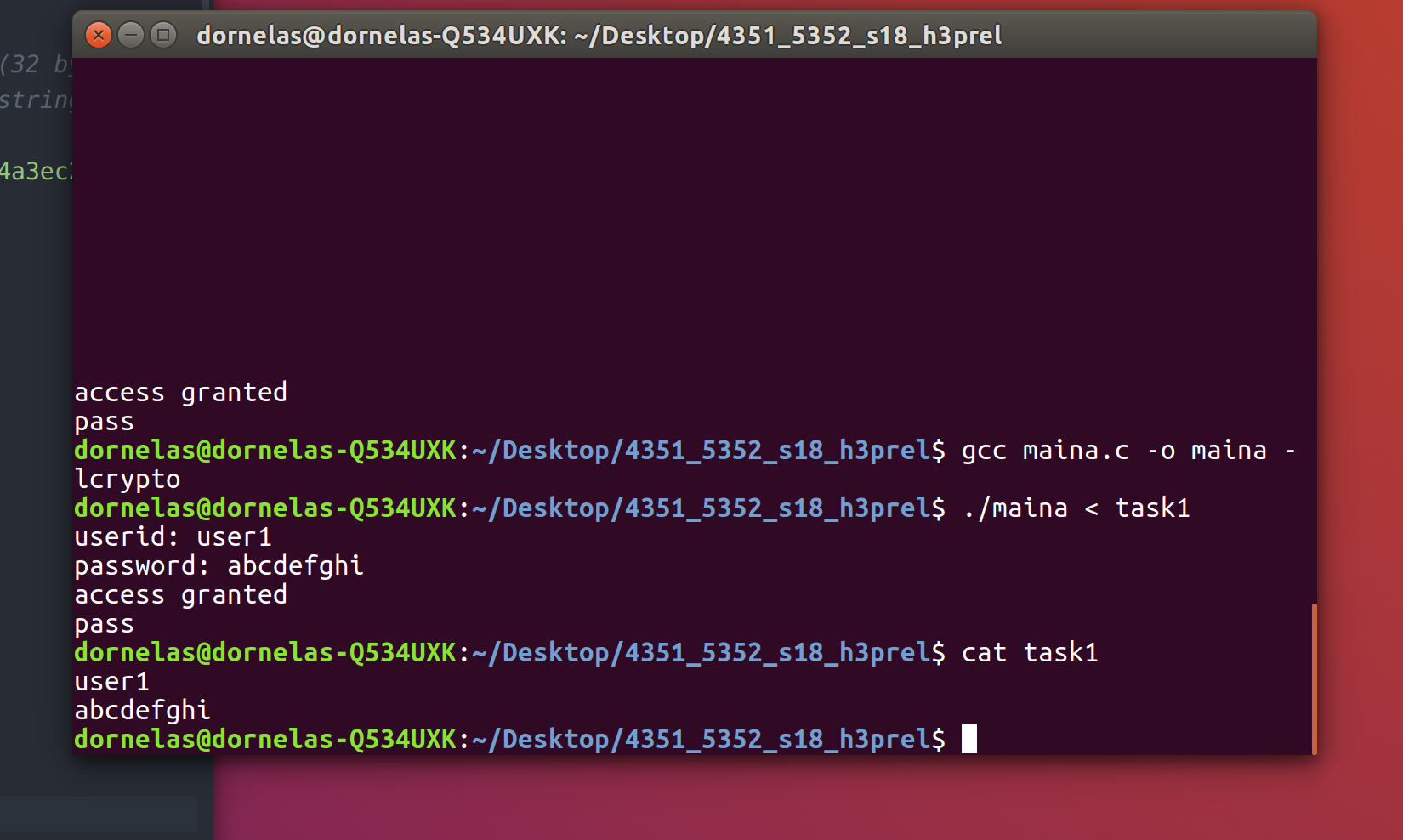
**Buffer Overflow - Daniel Ornelas**

**Task1**.

Program Running with typed inputs

Program Running with redirected input

Name of file for this task is “task1”

**Task2**

To figure out memory layout of maina.c. I compiled the program with the -g flag and I opened the compiled program in gdb to debug the memory. I first tried to open in IDA, but dynamic analysis is needed for the overflow. I then printed the addresses using the code provided, but addresses change every time as the process is allocated to a different segment in memory. In gdb by typing “p $sp” I found that the stack pointer was in 0x7fffffffd928. I also found that in address 0x7fffffffd8a0 you can find the target variable The two variables are not far from each other. Since we are overflowing the pw1 variable I looked for its address which turned out to be 0x7fffffffd7d0. I used python to print the letter A multiple times and give the program different inputs. When I started overflowing variable target, I put as input the expected string that was given in hashhex, since we can get the flag by having hashhex and target be equal. After setting target equal to hashhex. The overflow worked and I saved the malicious input in file mala.

The following is my picture of the memory

0x7fffffffd928 ----------------------------------------- Stack Pointer

---------------------------------------

---------------------------------------

--------------------------------------- Some memory I don’t care about

0x7fffffffd7d0 ----------------------------------------- pw1

---------------------------------------

------------------------------------------

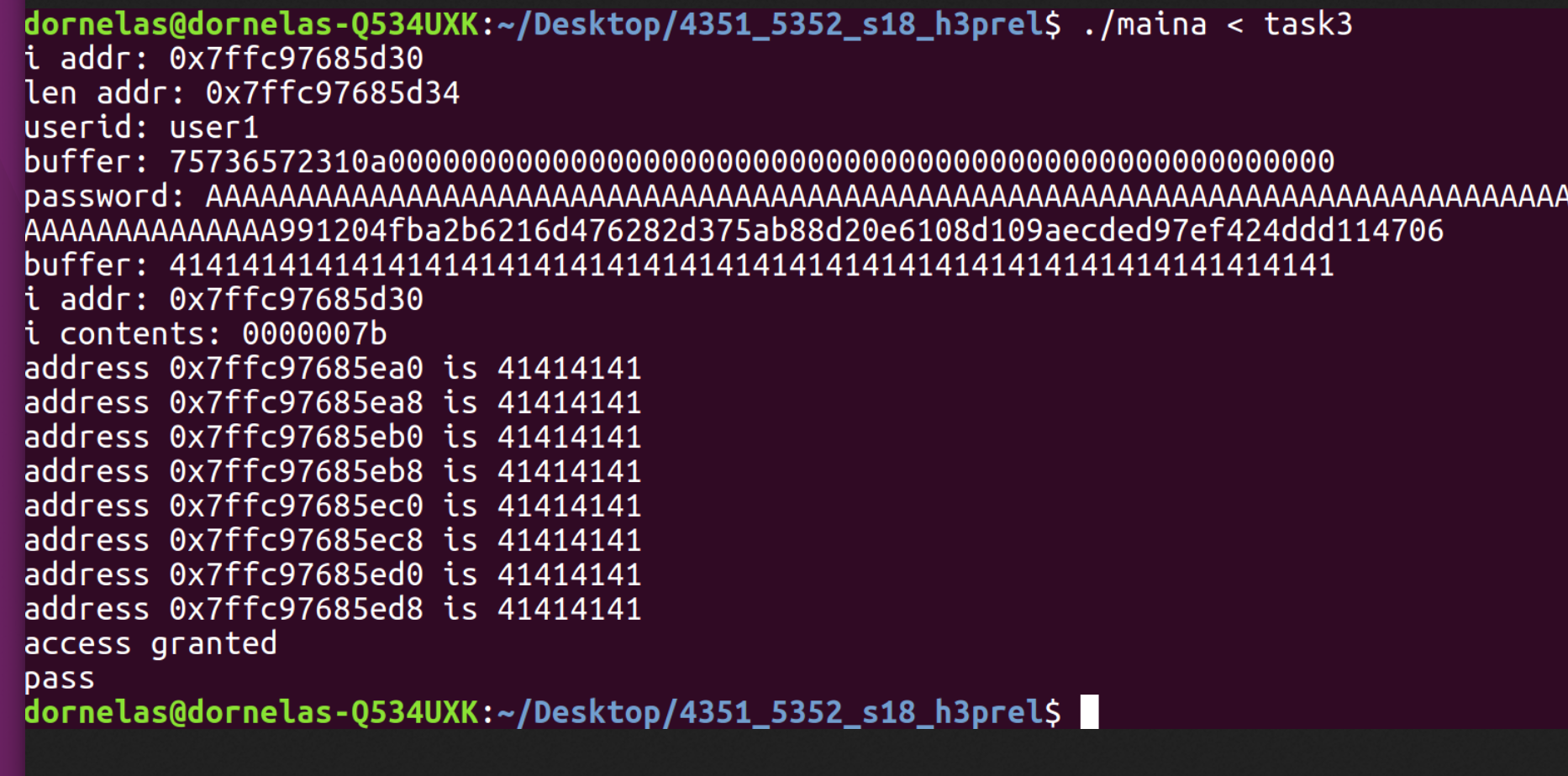
--------------------------------------

0x7ffffffffd8a0 --------------------------------------- target

**Task3**

**A.** In order to create a bad file, I followed the previous steps mentioned in task 2. I used gdb to debug the memory while I tried different bad inputs. I tried overflowing with just a lot of A’s so that memory addresses would change values to x41. Since the program is comparing the hashed target against the hashed user input. I tried to fill the target variable with what would be the hash for AAAAAAAAAAAAAAAA(16 A’s) which is the maximum length of the input. If I know what the hash of my input is, I can overflow the buffer and set my target variable to be equal to that hash, when the comparison happens, it will evaluate true and access will be granted.

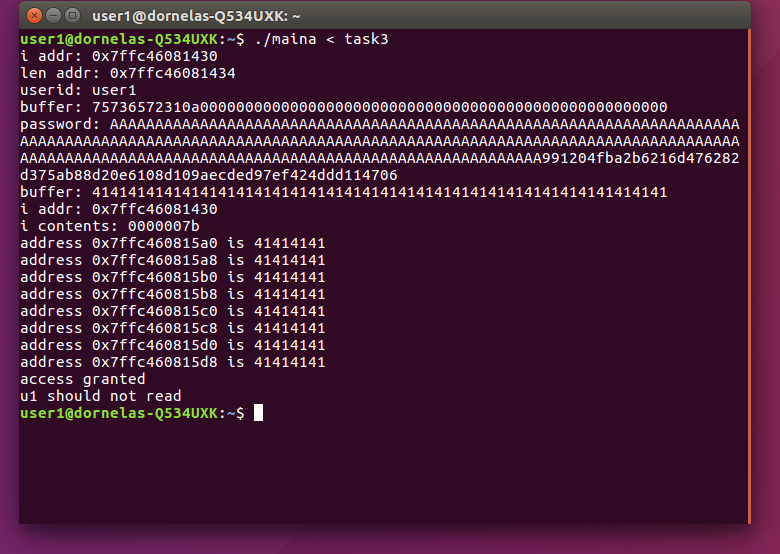
**B**

****

**\*\*Running the program\*\* ./maina < mala\*\* Note: Picture uses task3 as malicious file name was changed later.**

**Task4**

After setting program ownership to user2, setting programs as setuserid and making sure files were not readable by user 1. This was the output from running the file as user1:



**Task 5**

Individual files mainb,mainc, and maind were provided by the teacher. I followed the same steps of using gdb to exploit the vulnerability in each program.

**Running the programs:**

**./mainb < malb**

**./mainc < malc**

**./maind < mald**

**A.** Filenames for bad files for each individual program are the following

Program Bad file Name

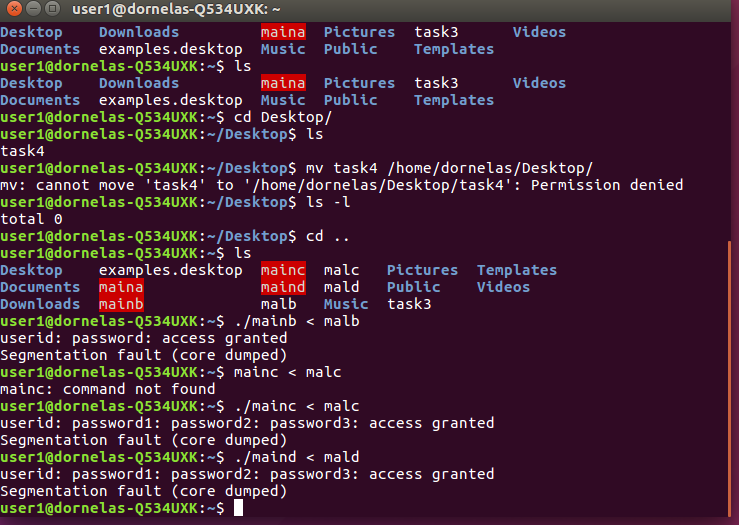
Mainb malb

Mainc malc

Maind mald

\*\*Files are available in folder\*\*

**B**Screenshot of succesful execution as user1 following the steps from task4



**Task 6**

A server username and password was provided so that we could exploit the vulnerabilities in the server using the same malicious files given. The programs on the server would be the same that we received by email.

Flags obtained from server

Flag B for usr 38 is : RgQV$LzVoC

Flag C for usr 38 is : #C%uM6WDfe

Flag D for usr 38 is : \*M1E8aRSU8

**Conclusion**

1. The assignment was easy once you understand the memory layout in each program, but figuring out the layout and the input for each malicious file is challenging. One of the challenges I encountered was figuring out which variable to overwrite, until I realized that it should be the one being compared to the user input by looking at the source code.
2. I worked individually
3. In this assignment I learned how to do a buffer overflow and I think that having different programs was great as I got to prove that I actually learned. I also learned how to debug memory using gdb.
4. Next year, you could have programs with different methods in which the return address needs to be changed so that the user can call the method that prints the flag.