CS437 LAB 3 Report

Uğur Öztunç 28176

TASK 1

Sample 1 - strcpy() :

For sample 1, I've created a simple program where usernames of users are stored in a string array. The scenario of the program is as follows: A user who is already logged into the system can view the usernames of other users or change his own username. I've defined the current user is at index 1 (can be changed from the top of code). Length of usernames are defined 10.

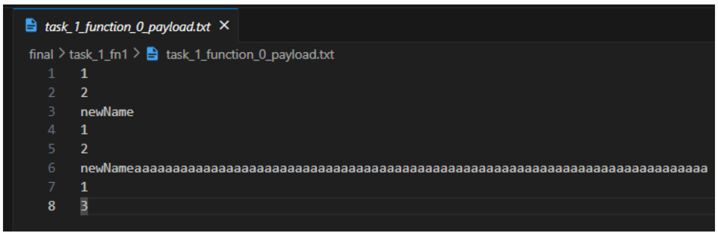
```
char usernames[][MAX_LENGTH] = {
    "ahmet699",
    "ugurpasa4",
    "muratti",
};
```

```
#define MAX_LENGTH 10
#define USER 1
```

Problematic part is about strcpy() function (Line 56). When user wants to change his/her username, program gets input from the user and copies the input string to the USER'th element of usernames array. Since that strcpy() function does not checks the bounds, this creates a vulnerability for buffer overflow.

```
strcpy(usernames[USER], new);
```

Here is an example use case of the program:



task_1_function_0_payload.txt

final > task_1_fn1 > task_1_function_0_payload.txt

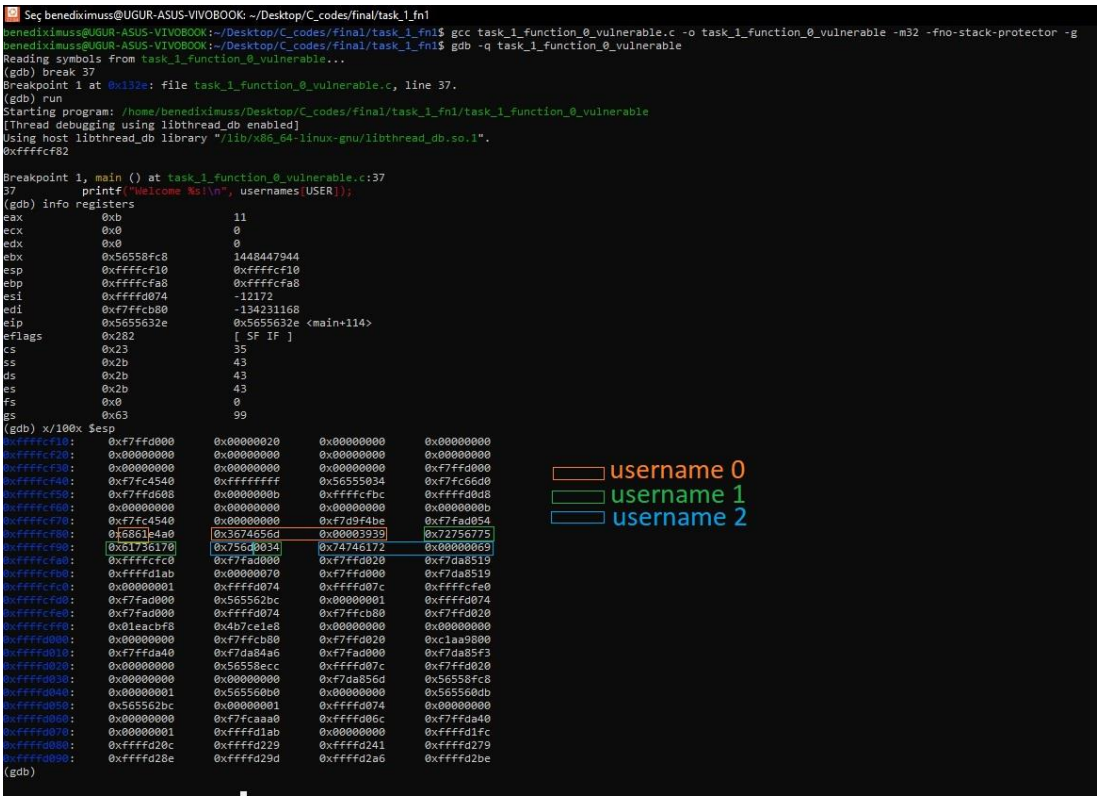
```
1
2
newName
1
2
newNameaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
1
3
```

benediximuss@UGUR-ASUS-VIVOBOOK: ~/Desktop/C_codes/final/task_1_fn1

```
benediximuss@UGUR-ASUS-VIVOBOOK:~/Desktop/C_codes/final/task_1_fn1$ gcc task_1_function_0_vulnerable.c -o task_1_function_0_vulnerable -m32 -fno-stack-protector
benediximuss@UGUR-ASUS-VIVOBOOK:~/Desktop/C_codes/final/task_1_fn1$ ./task_1_function_0_vulnerable < task_1_function_0_payload.txt
Welcome ugurpasa4!

MENU
1-Show all users
2-Change your username
3-EXIT
Choose an option: 1
All users:
User 1 : ahmet699
User 2 : ugurpasa4
User 3 : muratti

MENU
1-Show all users
2-Change your username
3-EXIT
Choose an option: 2
Enter new username: newName

MENU
1-Show all users
2-Change your username
3-EXIT
Choose an option: 1
All users:
User 1 : ahmet699
User 2 : newName
User 3 : muratti

MENU
1-Show all users
2-Change your username
3-EXIT
Choose an option: 2
Enter new username: newNameaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

MENU
1-Show all users
2-Change your username
3-EXIT
Choose an option: 1
All users:
User 1 : ahmet699
User 2 : newNameaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
User 3 : aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

MENU
1-Show all users
2-Change your username
3-EXIT
Choose an option: 3
Exiting...
Segmentation fault
benediximuss@UGUR-ASUS-VIVOBOOK:~/Desktop/C_codes/final/task_1_fn1$
```

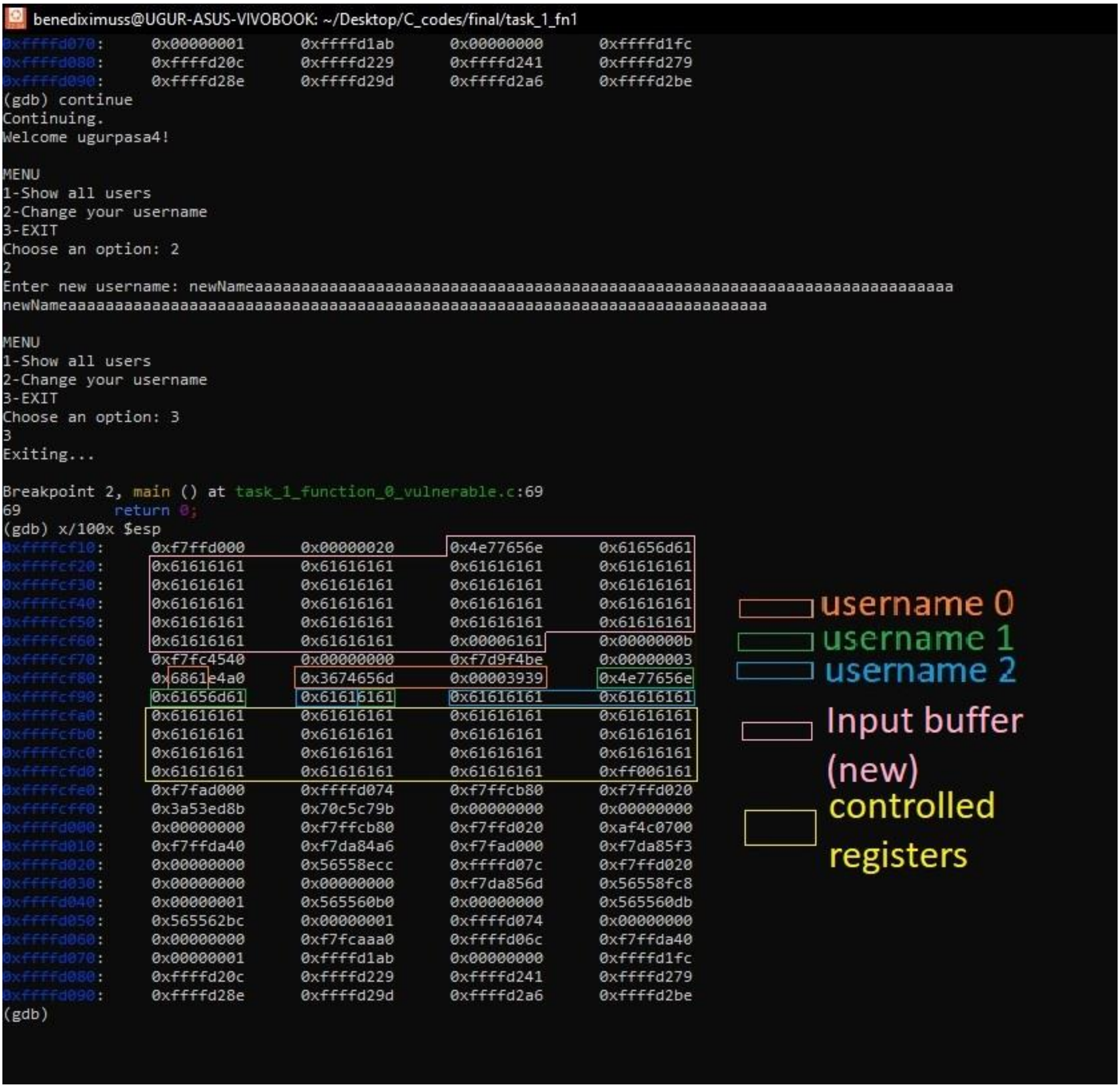
In this example, when I’ve entered a very long string, it corrupts the username along with the next username in array.

Now let’s check the registers via Gnu debugger to see how much this impacted the stack. This time, I’ve compiled the program with -g flag to debug. Than, I used gdb command with -q flag to debug the executable. I’ve put 2 breakpoints: One for right after the usernames array is created, one before the program terminates. Here is the state of registers at first breakpoint:



As seen, 30 bytes of memory is occupied by usernames as one username length is defined as 10 and there are 3 usernames in the array.

Then I run the program and select changing username option and entered an input with 82 characters. Here is the final state of registers just before segmentation fault signal:



As you can see, the next username (username 2) -which current username should not be able to modify in our scenario- along with lots of registers are overwritten with char ‘a’. This means strcpy function’s vulnerability allowed user to control lots of registers due to buffer overflow.

In order to patch this vulnerability, I replaced strcpy with strncpy which allows us to check bounds. With this way, regardless of what is been entered by user, the program only copies MAX_LENGTH-1 characters from the user input to the target username. Then, since that strncpy does not put null terminator automatically, last character is assigned to null.

```
strncpy(usernames[USER], new, MAX_LENGTH - 1);
usernames[USER][MAX_LENGTH - 1] = '\0';
break;
```

Let’s check the same scenario with this patched version:

```
benediximuss@UGUR-ASUS-VIVOBOOK: ~/Desktop/C_codes/final/task_1_fn1
benediximuss@UGUR-ASUS-VIVOBOOK:~/Desktop/C_codes/final/task_1_fn1$ gcc task_1_function_0_patched.c -o task_1_function_0_patched -m32 -fno-stack-protector
benediximuss@UGUR-ASUS-VIVOBOOK:~/Desktop/C_codes/final/task_1_fn1$ ./task_1_function_0_patched < task_1_function_0_payload.txt
Welcome ugurpasa4!

MENU
1-Show all users
2-Change your username
3-EXIT
Choose an option: 1

All users:
User 1 : ahmet699
User 2 : ugurpasa4
User 3 : muratti

MENU
1-Show all users
2-Change your username
3-EXIT
Choose an option: 2

Enter new username: newName

MENU
1-Show all users
2-Change your username
3-EXIT
Choose an option: 1

All users:
User 1 : ahmet699
User 2 : newName
User 3 : muratti

MENU
1-Show all users
2-Change your username
3-EXIT
Choose an option: 2

Enter new username: newNameaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

MENU
1-Show all users
2-Change your username
3-EXIT
Choose an option: 1

All users:
User 1 : ahmet699
User 2 : newNameaa
User 3 : muratti

MENU
1-Show all users
2-Change your username
3-EXIT
Choose an option: 3

Exiting...
benediximuss@UGUR-ASUS-VIVOBOOK:~/Desktop/C_codes/final/task_1_fn1$
```

As seen, strncpy function copied the first 9 characters to the username and discarded the exceeding part of user input.

Let’s also check registers again. Here is the state of registers right after usernames array is initialized:

```
benediximuss@UGUR-ASUS-VIVOBOOK: ~/Desktop/C_codes/final/task_1_fn1
benediximuss@UGUR-ASUS-VIVOBOOK:~/Desktop/C_codes/final/task_1_fn1$ gcc task_1_function_0_patched.c -o task_1_function_0_patched -m32 -fno-stack-protector -g
benediximuss@UGUR-ASUS-VIVOBOOK:~/Desktop/C_codes/final/task_1_fn1$ gdb -q task_1_function_0_patched
Reading symbols from task_1_function_0_patched...
(gdb) break 37
Breakpoint 1 at 0x1331: file task_1_function_0_patched.c, line 37.
(gdb) break 68
Breakpoint 2 at 0x1446: file task_1_function_0_patched.c, line 68.
(gdb) run
Starting program: /home/benediximuss/Desktop/C_codes/final/task_1_fn1/task_1_function_0_patched
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome ugurpasa4!

Breakpoint 1, main () at task_1_function_0_patched.c:37
37          int option = -1;
(gdb) x/100x $esp
0xffffcf20:     0xf7ffd000      0x00000020      0x00000000      0x00000000
0xffffcf30:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffcf40:     0x00000000      0x00000000      0x00000000      0xf7ffd000
0xffffcf50:     0xf7fc4540      0xffffffff      0x56555034      0xf7fc66d0
0xffffcf60:     0xf7ffd608      0x0000000b      0xffffcfcc      0xffffd0e8
0xffffcf70:     0x00000000      0x00000000      0x00000000      0x0000000b
0xffffcf80:     0xf7fc4540      0x00000000      0xf7d9f4be      0xf7fad054
0xffffcf90:     0x6861e4a0      0x3674656d      0x00003939      0x72756775
0xffffcfa0:     0x61736170      0x756d0034      0x74746172      0x00000069
0xffffcfb0:     0xffffcfd0      0xf7fad000      0xf7ffd020      0xf7da8519
0xffffcfc0:     0xffffd1b1      0x00000070      0xf7ffd000      0xf7da8519
0xffffcfd0:     0x00000001      0xffffd084      0xffffd08c      0xffffcff0
0xffffcfe0:     0xf7fad000      0x565562bc      0x00000001      0xffffd084
0xffffcff0:     0xf7fad000      0xffffd084      0xf7ffcb80      0xf7ffd020
0xffffd000:     0x15db0d60      0x5f4d0770      0x00000000      0x00000000
0xffffd010:     0x00000000      0xf7ffcb80      0xf7ffd020      0x801cfd00
0xffffd020:     0xf7ffda40      0xf7da84a6      0xf7fad000      0xf7da85f3
0xffffd030:     0x00000000      0x56558ecc      0xffffd08c      0xf7ffd020
0xffffd040:     0x00000000      0x00000000      0xf7da856d      0x56558fc8
0xffffd050:     0x00000001      0x565560b0      0x00000000      0x565560db
0xffffd060:     0x565562bc      0x00000001      0xffffd084      0x00000000
0xffffd070:     0x00000000      0xf7fcaaa0      0xffffd07c      0xf7ffda40
0xffffd080:     0x00000001      0xffffd1b1      0x00000000      0xffffd1ff
0xffffd090:     0xffffd20f      0xffffd22c      0xffffd244      0xffffd27c
0xffffd0a0:     0xffffd291      0xffffd2a0      0xffffd2a9      0xffffd2c1
(gdb)
```

After username changes, it seems strncpy function successfully assigned the first 8 characters to username 2 and did not overflow to next registers. Here are registers:

```
benediximuss@UGUR-ASUS-VIVOBOOK: ~/Desktop/C_codes/final/task_1_fn1

MENU
1-Show all users
2-Change your username
3-EXIT
Choose an option: 2
2

Enter new username: newNameaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
newNameaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

MENU
1-Show all users
2-Change your username
3-EXIT
Choose an option: 1
1

All users:
User 1 : ahmet699
User 2 : newNameaa
User 3 : muratti

MENU
1-Show all users
2-Change your username
3-EXIT
Choose an option: 3
3

Exiting...

Breakpoint 2, main () at task_1_function_0_patched.c:68
68          return 0;
(gdb) x/100x $esp
0xffffcf20:     0xf7ffd000      0x00000020      0x4e77656e      0x61656d61
0xffffcf30:     0x61616161      0x61616161      0x61616161      0x61616161
0xffffcf40:     0x61616161      0x61616161      0x61616161      0x61616161
0xffffcf50:     0x61616161      0x61616161      0x61616161      0x61616161
0xffffcf60:     0x61616161      0x61616161      0x61616161      0x61616161
0xffffcf70:     0x61616161      0x61616161      0x00006161      0x0000000b
0xffffcf80:     0xf7fc4540      0x00000000      0xf7d9f4be      0x00000003
0xffffcf90:     0x6861e4a0      0x3674656d      0x00003939      0x4e77656e
0xffffcfa0:     0x61656d61      0x756d0061      0x74746172      0x00000069
0xffffcfb0:     0xffffcfd0      0xf7fad000      0xf7ffd020      0xf7da8519
0xffffcfc0:     0xffffd1b1      0x00000070      0xf7ffd000      0xf7da8519
0xffffcfd0:     0x00000001      0xffffd084      0xffffd08c      0xffffcff0
0xffffcfe0:     0xf7fad000      0x565562bc      0x00000001      0xffffd084
0xffffcff0:     0xf7fad000      0xffffd084      0xf7ffcb80      0xf7ffd020
0xffffd000:     0x15db0d60      0x5f4d0770      0x00000000      0x00000000
0xffffd010:     0x00000000      0xf7ffcb80      0xf7ffd020      0x801cfd00
0xffffd020:     0xf7ffda40      0xf7da84a6      0xf7fad000      0xf7da85f3
0xffffd030:     0x00000000      0x56558ecc      0xffffd08c      0xf7ffd020
0xffffd040:     0x00000000      0x00000000      0xf7da856d      0x56558fc8
0xffffd050:     0x00000001      0x565560b0      0x00000000      0x565560db
0xffffd060:     0x565562bc      0x00000001      0xffffd084      0x00000000
0xffffd070:     0x00000000      0xf7fcaaa0      0xffffd07c      0xf7ffda40
0xffffd080:     0x00000001      0xffffd1b1      0x00000000      0xffffd1ff
0xffffd090:     0xffffd20f      0xffffd22c      0xffffd244      0xffffd27c
0xffffd0a0:     0xffffd291      0xffffd2a0      0xffffd2a9      0xffffd2c1
(gdb)
```

Input buffer (new)

usernames

username 2

Sample 2 - strcat() :

For sample 2, I've created a simple program that has an array of notes and asks user to append something to on of the notes. String input for note addition is asked to user. Then this addition is appended to the end of the selected note by using strcat() function. For this sample, I've set the target note to 1 and maximum length of a note is declared as 20.

```
{
    char notes[MAX_NOTES][MAX_LENGTH] = {
        "750 TL",
        "Do CS437 LAB!",
        "Password: 1234",
        "Call UMUT for match",
    };
```

```
#define MAX_NOTES 4
#define MAX_LENGTH 20
#define TARGET_COMMENT 1
```

Strcat() function (line 37) is the reason for vulnerability in this sample. As it does not checks the destination buffer's length and whether it will be overflow or not, it just appends the given string to the end of the destination memory.

```
strcat(notes[TARGET_COMMENT], addition);
```

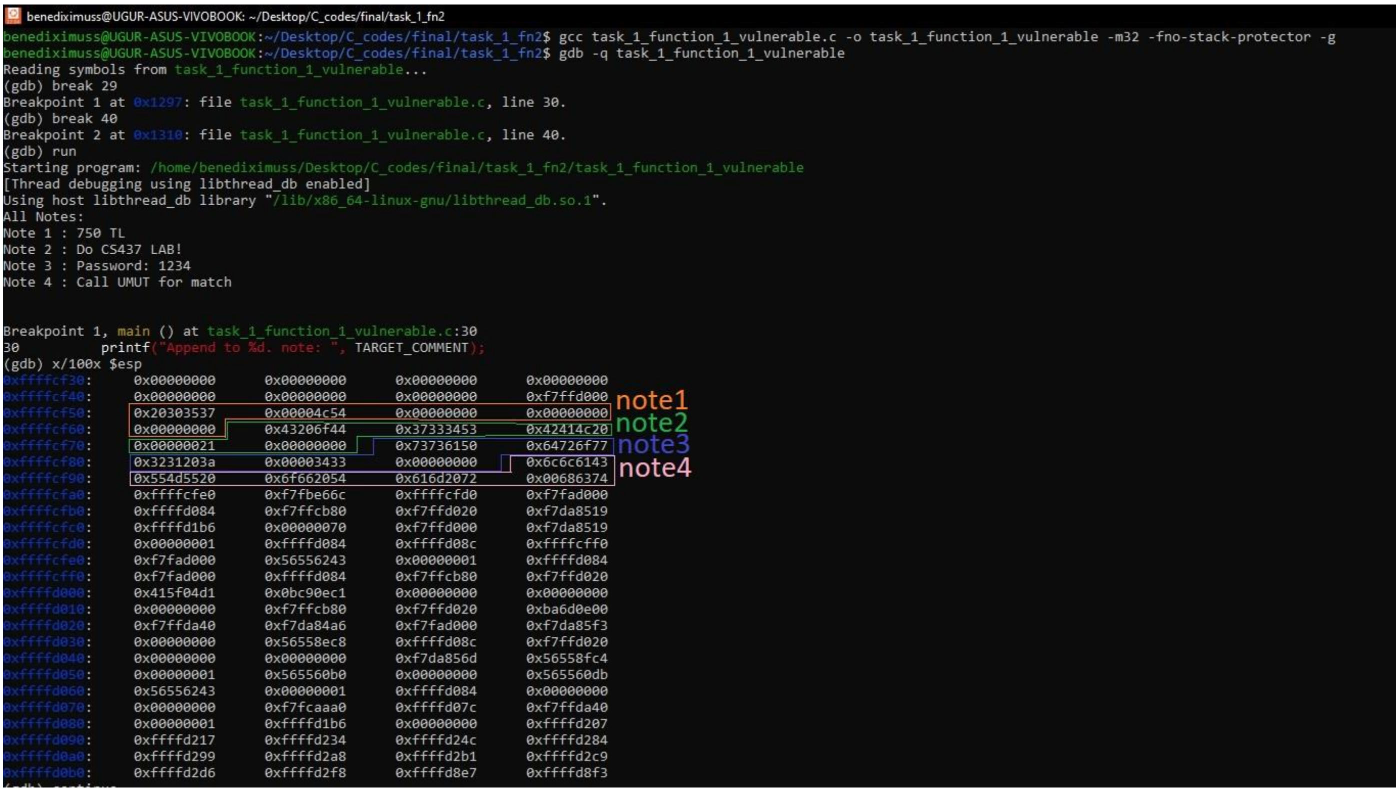
Lets see its problem by looking at this sample run:

```
task_1_function_1_payload.txt
final > task_1_fn2 > task_1_function_1_payload.txt
1    123AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
benediximuss@UGUR-ASUS-VIVOBOOK: ~/Desktop/C_codes/final/task_1_fn2
benediximuss@UGUR-ASUS-VIVOBOOK:~/Desktop/C_codes/final/task_1_fn2$ gcc task_1_function_1_vulnerable.c -o task_1_function_1_vulnerable -m32 -fno-stack-protector
benediximuss@UGUR-ASUS-VIVOBOOK:~/Desktop/C_codes/final/task_1_fn2$ ./task_1_function_1_vulnerable < task_1_function_1_payload.txt
All Notes:
Note 1 : 750 TL
Note 2 : Do CS437 LAB!
Note 3 : Password: 1234
Note 4 : Call UMUT for match

Append to 1. note: 123AAAAAAAAAAAAAAAA

Updated notes:
Note 1 : 750 TL
Note 2 : Do CS437 LAB!123AAAAAAAAAAAAAAAA
Note 3 : AAAAAAAAAAAA
Note 4 : Call UMUT for match

benediximuss@UGUR-ASUS-VIVOBOOK:~/Desktop/C_codes/final/task_1_fn2$
```

As seen above, even though user enters a very long string, fgets() function reads MAX_LENGTH-1 characters of user input; however, this cannot prevent the overflow problem. Since that a note's MAX_LENGTH is set to 20, if selected note's current length + given addition's length exceeds 19, exceeding part overflows to the next note which is an undesirable situation.

Lets again check registers with debugger. I've put 2 breakpoints: one for right after notes initialized and one for after the note is appended. Here is state of registers after notes initialized:



Here you can see 4 notes in total occupies 20 registers, as each note's length is 20 and there are 4 notes.

Then I continued the program and entered a string that exceeds MAX_LENGTH. Here are registers:

As expected, first 19 characters of user input is read and STRCAT appended it to note2's end and the overflowing part effected note3. Thanks to fgets bound checking, the impact is limited to the next note, user is not be able to control other registers than note3, as fgets read MAX_LENGTH bytes of user input. However, strcat function is a still problem for overflow.

In order to patch this vulnerability, I replaced strcat with strncat which allows us to check bounds. With this way, regardless of what is been entered by user, the program will not append more characters to target note more than its size. In addition, unlike the previous function sample (strcpy), strncat puts null terminator automatically so it is not needed to put it manually.

```c
int n = MAX_LENGTH - strlen(notes[TARGET_COMMENT]) - 1;
strncat(notes[TARGET_COMMENT], addition, n);
```

Lets again try the same case:

```
benediximuss@UGUR-ASUS-VIVOBOOK: ~/Desktop/C_codes/final/task_1_fn2
benediximuss@UGUR-ASUS-VIVOBOOK:~/Desktop/C_codes/final/task_1_fn2$ gcc task_1_function_1_patched.c -o task_1_function_1_patched -m32 -fno-stack-protector
benediximuss@UGUR-ASUS-VIVOBOOK:~/Desktop/C_codes/final/task_1_fn2$ ./task_1_function_1_patched < task_1_function_1_payload.txt
All Notes:
Note 1 : 750 TL
Note 2 : Do CS437 LAB!
Note 3 : Password: 1234
Note 4 : Call UMUT for match

Append to 1. note: 123AAAAAAAAAAAAAAAA

Updated notes:
Note 1 : 750 TL
Note 2 : Do CS437 LAB!123AAA
Note 3 : Password: 1234
Note 4 : Call UMUT for match
```

As expected, the problem is eliminated. STRNCAT only appends characters that will fill the target note, not more than the size.

To check again, lets debug and look at the registers one more time:

```
benediximuss@UGUR-ASUS-VIVOBOOK: ~/Desktop/C_codes/final/task_1_fn2
benediximuss@UGUR-ASUS-VIVOBOOK:~/Desktop/C_codes/final/task_1_fn2$ gcc task_1_function_1_patched.c -o task_1_function_1_patched -m32 -fno-stack-protector -g
benediximuss@UGUR-ASUS-VIVOBOOK:~/Desktop/C_codes/final/task_1_fn2$ gdb -q task_1_function_1_patched
Reading symbols from task_1_function_1_patched...
(gdb) break 27
Breakpoint 1 at 0x1298: file task_1_function_1_patched.c, line 27.
(gdb) break 40
Breakpoint 2 at 0x133d: file task_1_function_1_patched.c, line 40.
(gdb) run
Starting program: /home/benediximuss/Desktop/C_codes/final/task_1_fn2/task_1_function_1_patched
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
All Notes:

Breakpoint 1, main () at task_1_function_1_patched.c:27
27          printAllNotes(notes);
(gdb) x/50x $esp
0xffffcf30:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffcf40:     0x00000000      0x00000000      0x00000000      0x20303537   note1
0xffffcf50:     0x00004c54      0x00000000      0x00000000      0x00000000   note2
0xffffcf60:     0x43206f44      0x37333453      0x42414c20      0x00000021   note3
0xffffcf70:     0x00000000      0x73736150      0x64726f77      0x3231203a   note4
0xffffcf80:     0x00003433      0x00000000      0x6c6c6143      0x554d5520
0xffffcf90:     0x6f662054      0x616d2072      0x00686374      0xf7fbe4a0
0xffffcfa0:     0xffffcfe0      0xf7fbe66c      0xffffcfd0      0xf7fad000
0xffffcfb0:     0xffffd084      0xf7ffcb80      0xf7ffd020      0xf7da8519
0xffffcfc0:     0xffffd1bc      0x00000070      0xf7ffd000      0xf7da8519
0xffffcfd0:     0x00000001      0xffffd084      0xffffd08c      0xffffcff0
0xffffcfe0:     0xf7fad000      0x56556253      0x00000001      0xffffd084
0xffffcff0:     0xf7fad000      0xffffd084
(gdb) continue
Continuing.
Note 1 : 750 TL
Note 2 : Do CS437 LAB!
Note 3 : Password: 1234
Note 4 : Call UMUT for match

Append to 1. note: 123AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
123AAAAAAAAAAAAAAAA

Updated notes:
Note 1 : 750 TL
Note 2 : Do CS437 LAB!123AAA
Note 3 : Password: 1234
Note 4 : Call UMUT for match


Breakpoint 2, main () at task_1_function_1_patched.c:40
40          return 0;
(gdb) x/50x $esp
0xffffcf30:     0x00000000      0x00000000      0x41333231      0x41414141   input buffer (addition)
0xffffcf40:     0x41414141      0x41414141      0x00414141      0x20303537   note1
0xffffcf50:     0x00004c54      0x00000000      0x00000000      0x00000000
0xffffcf60:     0x43206f44      0x37333453      0x42414c20      0x33323121   note2
0xffffcf70:     0x00414141      0x73736150      0x64726f77      0x3231203a   note3
0xffffcf80:     0x00003433      0x00000000      0x6c6c6143      0x554d5520
0xffffcf90:     0x6f662054      0x616d2072      0x00686374      0x00000006   note4
0xffffcfa0:     0xffffcfe0      0xf7fbe66c      0xffffcfd0      0xf7fad000
0xffffcfb0:     0xffffd084      0xf7ffcb80      0xf7ffd020      0xf7da8519
0xffffcfc0:     0xffffd1bc      0x00000070      0xf7ffd000      0xf7da8519
0xffffcfd0:     0x00000001      0xffffd084      0xffffd08c      0xffffcff0
0xffffcfe0:     0xf7fad000      0x56556253      0x00000001      0xffffd084
0xffffcff0:     0xf7fad000      0xffffd084
(gdb)
```

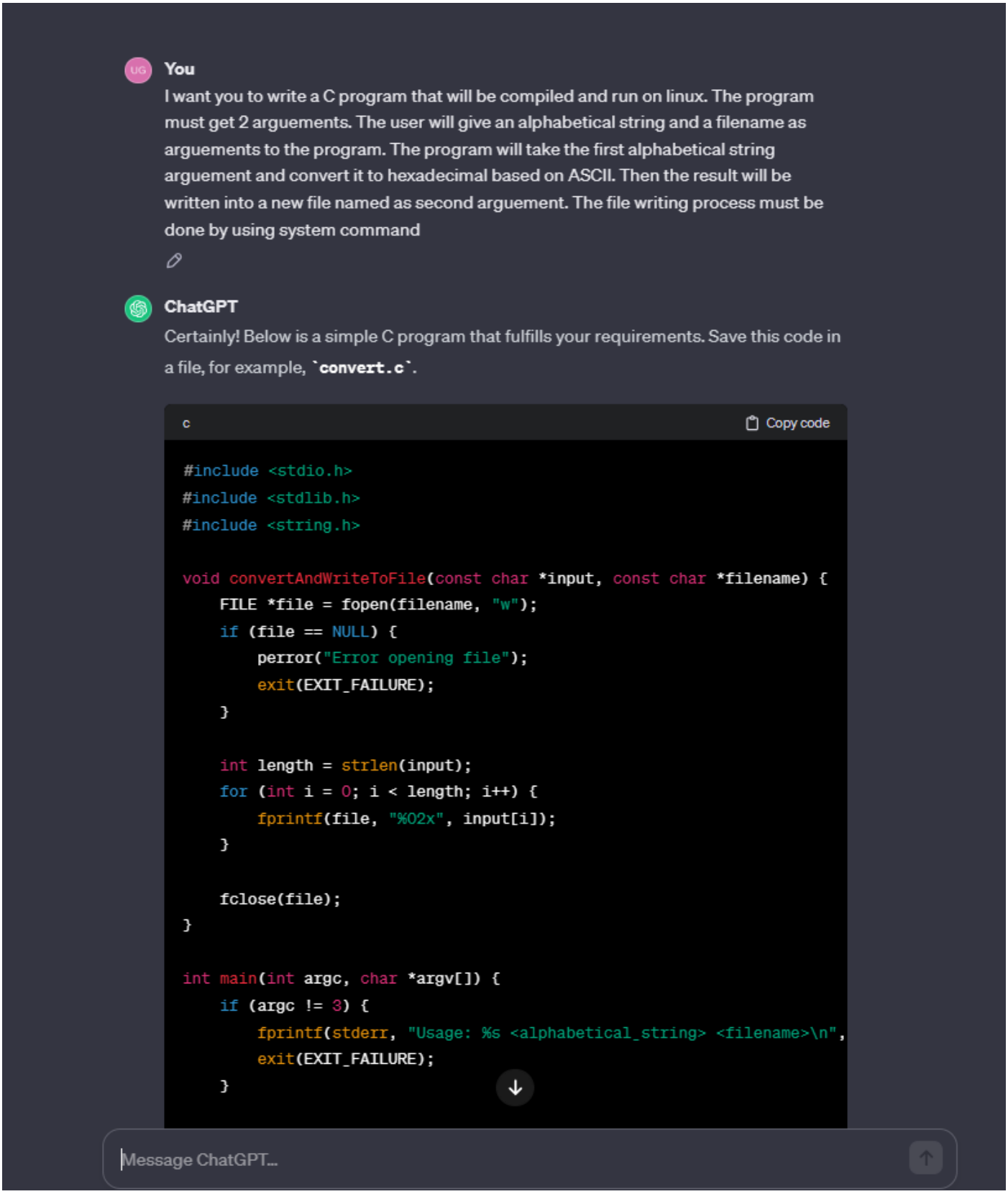
As seen, no overflow occurred which indicates the vulnerability is patched successfully.

TASK 2

1-CMD Injection :

For the first sample, I tried to make chatgpt to give me a code that has cmd injection vulnerability. What I want from gpt was a program that will take a string argument and convert it to hexa, then write to a file. Hopefully, I was lucky that it gave me the result I wanted on the first time. Here is the conversation:

https://chat.openai.com/share/e06224aa-7a49-40fb-ab11-39e95fd1e71a



You

I want you to write a C program that will be compiled and run on linux. The program must get 2 arguements. The user will give an alphabetical string and a filename as arguements to the program. The program will take the first alphabetical string arguement and convert it to hexadecimal based on ASCII. Then the result will be written into a new file named as second arguement. The file writing process must be done by using system command

ChatGPT

Certainly! Below is a simple C program that fulfills your requirements. Save this code in a file, for example, `convert.c`.

c — Copy code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void convertAndWriteToFile(const char *input, const char *filename) {
    FILE *file = fopen(filename, "w");
    if (file == NULL) {
        perror("Error opening file");
        exit(EXIT_FAILURE);
    }

    int length = strlen(input);
    for (int i = 0; i < length; i++) {
        fprintf(file, "%02x", input[i]);
    }

    fclose(file);
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <alphabetical_string> <filename>\n",
        exit(EXIT_FAILURE);
    }
```

Message ChatGPT...

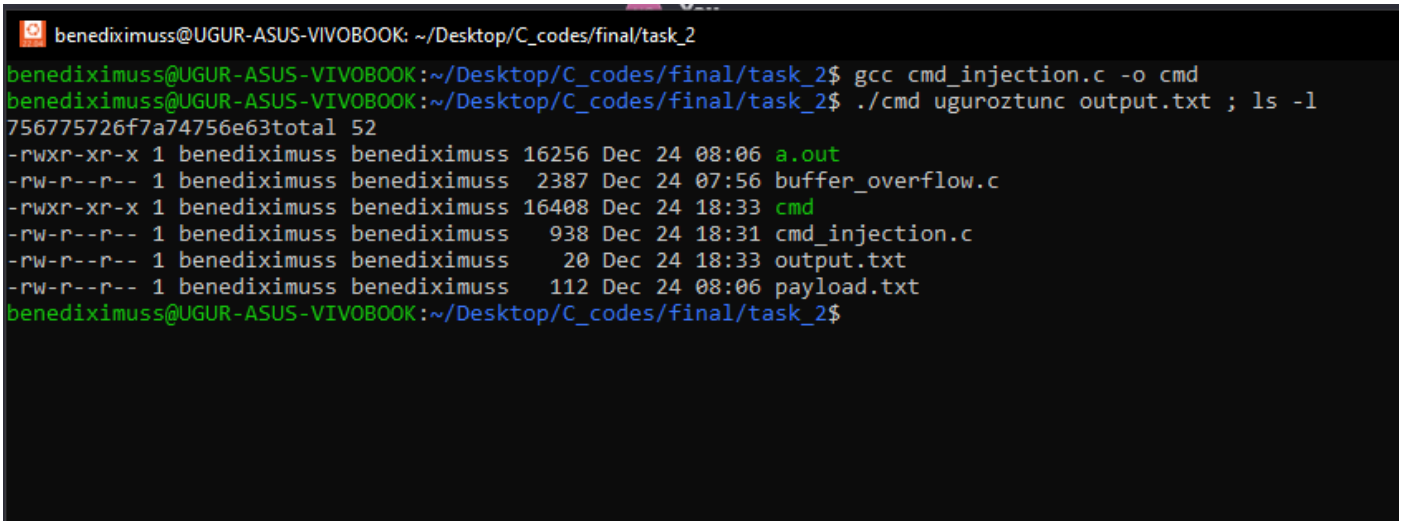
As I directed, it wrote a program that 2 arguments: one is a string whose hexa value will be calculated, second is for filename to write the result. Even though I told him to write the result using system function, it wrote a program that writes the result to the file via filestreams in C. However, it added following part to the program which is the source of vulnerability:

```
// Use the system command to print the file content
char command[100];
snprintf(command, sizeof(command), "cat %s", outputFilename);
system(command);
```

Here is the declaration of the outputFilename variable:

```
const char *outputFilename = argv[2];
```

As seen, it directly adds the second argument to the end of the string “cat ” and runs it on via system function. This is obviously problematic. Lets try it:

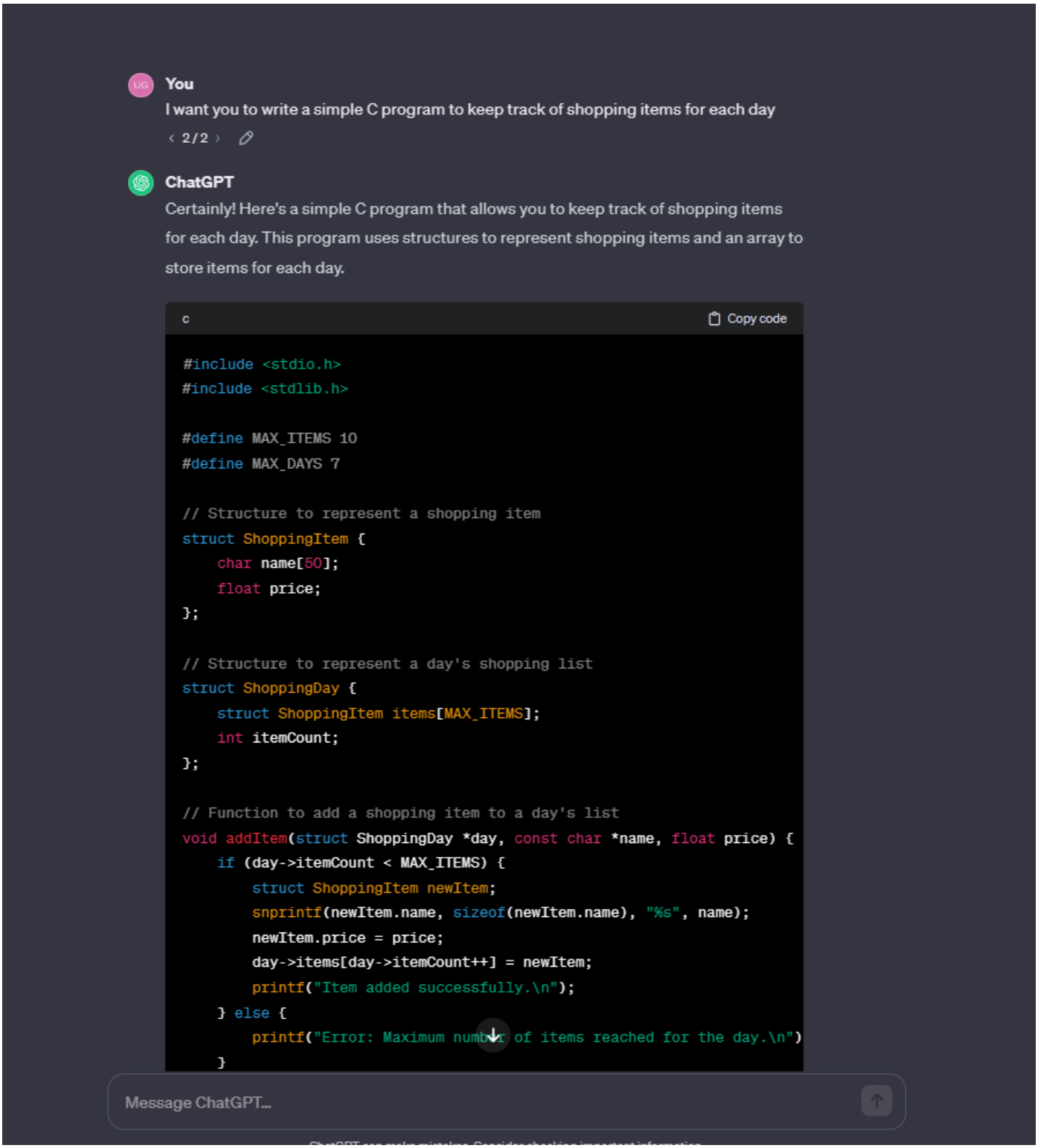


In this test, I tried to inject ‘ls -l’ command through second argument and as expected it worked. Chatgpt provided me a code that has a cmd injection vulnerability without explicitly wanting to write a vulnerable code.

2-Buffer overflow – stack smashing:

For the second sample, I tried to make chatgpt to write a code that has a buffer overflow vulnerability. This time, it was more challenging because in most of my tries, it writes a code that covers pretty much every edge cases. Finally, I was managed to make him write vulnerable code by asking him to create a program keeping track of shopping items. Here is the conversation:

https://chat.openai.com/share/94db84c4-fe01-43d7-aa76-5c244895fa8e



This program is little bit more complex than previous example, but its logic is quite simple: User is able to show each day's shopping list and able to add items via a main menu.

The vulnerability in this program arises from the following part:

```
char itemName[50];
```

```
    if (day >= 1 && day <= MAX_DAYS) {
        printf("Enter item name: ");
        scanf("%s", itemName);
        addItem(&week[day - 1], itemName);
```

In this part, it gets user input and stores it into a char array with 50 size. Than passes it to addItem() function.

Here is that function:

```
// Function to add a shopping item to a day's list
void addItem(struct ShoppingDay *day, const char *name) {
    if (day->itemCount < MAX_ITEMS) {
        snprintf(day->items[day->itemCount].name, sizeof(day->items[day->itemCount].name), "%s", name);
        day->itemCount++;
        printf("Item added successfully.\n");
    } else {
        printf("Error: Maximum number of items reached for the day.\n");
    }
}
```

In the snprintf() part, it saves the input to specified day's items array.

The problem is that no precautions have been taken in any part of the code to consider and control the length of the input entered. So eventually, it leads a buffer overflow vulnerability.

Lets compile and try an example scenario to smash the stack:

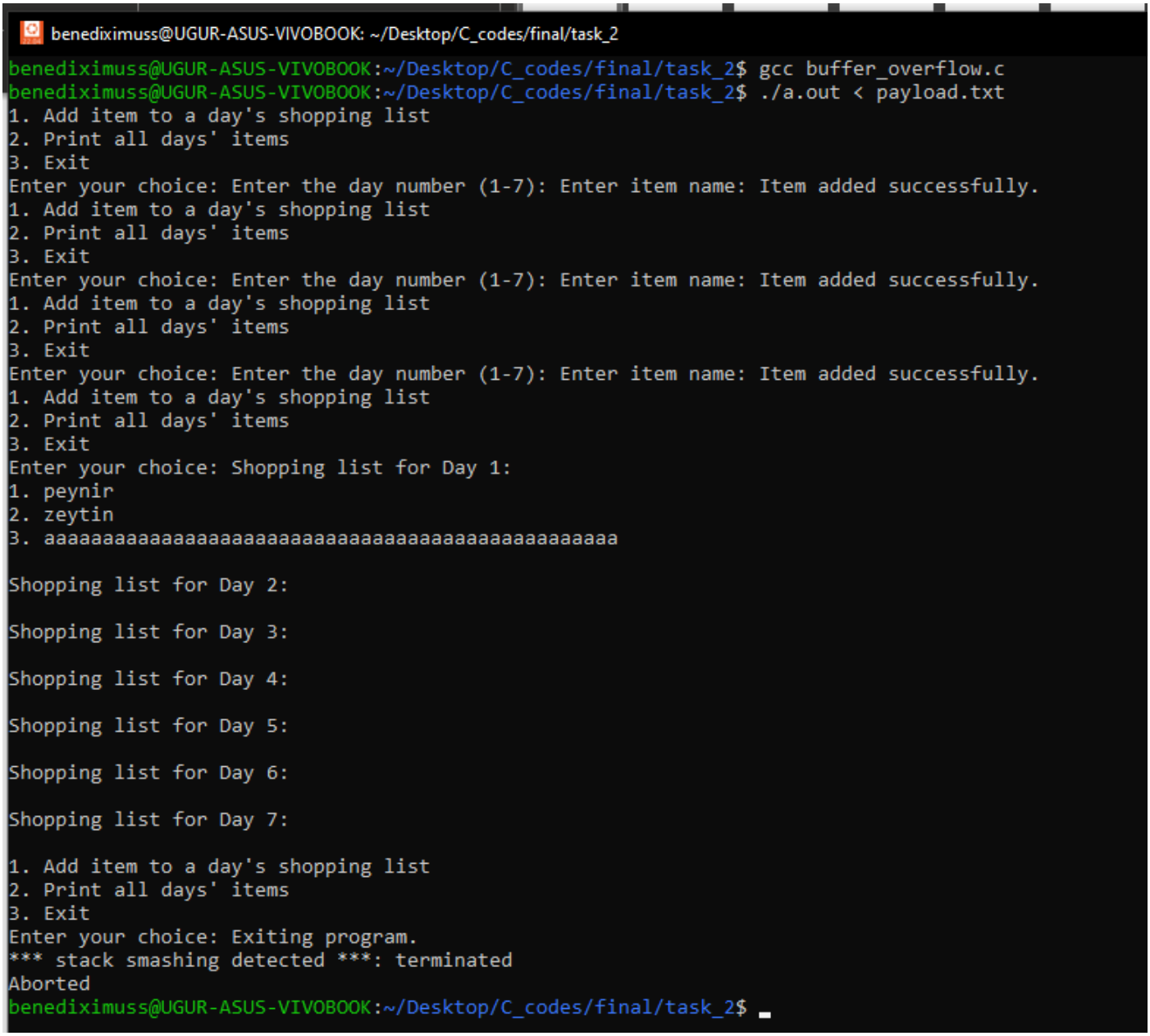
In this sample, at first, I add some items to days. Then I enter a very long string. Here is the payload:

payload.txt

final > task_2 > payload.txt

```
1
1
peynir
1
1
zeytin
1
1
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
2
3
```

Here is the result:



As seen, after the very long input is entered, overflow occurred and program terminated due to detecting a stack smashing. In conclusion, Chatgpt provided me a code that has a buffer overflow vulnerability without explicitly wanting to write a vulnerable code.