

# 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 X
final > task_1_fn1 > task_1_function_0_payload.txt
1 1
2 2
3 newName
4 1
5 2
6 newNameaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
7 1
8 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: newNameaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

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

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:

```
Seq benediximuss@UGUR-ASUS-VIVOBOK: ~/Desktop/C_codes/final/task_1_fn1
benediximuss@UGUR-ASUS-VIVOBOK:~/Desktop/C_codes/final/task_1_fn1$ gcc task_1_function_0_vulnerable.c -o task_1_function_0_vulnerable -m32 -fno-stack-protector -g
benediximuss@UGUR-ASUS-VIVOBOK:~/Desktop/C_codes/final/task_1_fn1$ gdb -q task_1_function_0_vulnerable
Reading symbols from task_1_function_0_vulnerable...
(gdb) break 37
Breakpoint 1 at 0x132e: file task_1_function_0_vulnerable.c, line 37.
(gdb) run
Starting program: /home/benediximuss/Desktop/C_codes/final/task_1_fn1/task_1_function_0_vulnerable
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
0xffffcf82

Breakpoint 1, main () at task_1_function_0_vulnerable.c:37
37      printf("Welcome %s!\n", usernames[USER]);
(gdb) info registers
eax            0xb          11
ecx            0x0          0
edx            0x0          0
ebx            0x56558fc8    1448447944
esp            0xffffcf10    0xffffcf10
ebp            0xffffcfa8    0xffffcfa8
esi            0xffffd074    -12172
edi            0xffffcb00    -134231168
eip            0x5655632e    0x5655632e <main+114>
eflags         0x282        [ SF IF ]
cs             0x23         35
ss             0x2b         43
ds             0x2b         43
es             0x2b         43
fs             0x0          0
gs             0x63         99
(gdb) x/100x $esp
0xffffcf10: 0x00000020 0x00000000 0x00000000 0x00000000
0xffffcf20: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffcf30: 0x00000000 0x00000000 0x00000000 0xf7fd0000
0xffffcf40: 0xf7fc4540 0xffffffff 0x56558fc8 0xf7fc66d0
0xffffcf50: 0xf7fdd000 0x0000000b 0xffffcfbc 0xffffd0d8
0xffffcf60: 0x00000000 0x00000000 0x00000000 0x0000000b
0xffffcf70: 0xf7fc4540 0x00000000 0xf7d0fabc 0xf7fad054
0xffffcf80: 015861e1a0 0x3674656d 0x00003939 0x72756775
0xffffcf90: 0x61736170 0x75630034 0x74746172 0x00000069
0xffffcfa0: 0xffffcfc0 0xf7fad000 0xf7fd020 0xf7da8519
0xffffcfb0: 0xffffd1ab 0x00000070 0xf7fd000 0xf7da8519
0xffffcfc0: 0x00000001 0xffffd074 0xffffd07c 0xffffcfe0
0xffffcf00: 0xf7fad000 0x565562bc 0x00000001 0xffffd074
0xffffcf20: 0xf7fad000 0xffffd074 0xf7fcb00 0xf7fd020
0xffffcf40: 0x01eacbf8 0x4b7c1e8 0x00000000 0x00000000
0xffffcf60: 0x00000000 0xf7fcb00 0xf7fd020 0xc1aa9800
0xffffcf80: 0xf7fda40 0xf7da8a6 0xf7fad000 0xf7da85f3
0xffffcf00: 0x00000000 0x56558ecc 0xffffd07c 0xf7fad020
0xffffcf20: 0x00000000 0x00000000 0xf7da856d 0x56558fc8
0xffffcf40: 0x00000001 0x565560b0 0x00000000 0x565566db
0xffffcf60: 0x565562bc 0x00000001 0xffffd074 0x00000000
0xffffcf80: 0x00000000 0xf7fcaaa0 0xffffd06c 0xf7fda40
0xffffcf00: 0x00000001 0xffffd1ab 0x00000000 0xffffd1fc
0xffffcf20: 0xffffd20c 0xffffd229 0xffffd241 0xffffd279
0xffffcf40: 0xffffd28e 0xffffd29d 0xffffd2a6 0xffffd2be
(gdb)
```

username 0  
username 1  
username 2

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:

```

benediximuss@UGUR-ASUS-VIVOBOK: ~/Desktop/C_codes/final/task_1_fn1
0xfffffd070: 0x00000001 0xfffffd1ab 0x00000000 0xfffffd1fc
0xfffffd080: 0xfffffd20c 0xfffffd229 0xfffffd241 0xfffffd279
0xfffffd090: 0xfffffd28e 0xfffffd29d 0xfffffd2a6 0xfffffd2be
(gdb) continue
Continuing.
Welcome ugurpasa4!

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

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_vulnerable.c:69
69      return 0;
(gdb) x/100x $esp
0xfffffcf10: 0xf7ffd000 0x00000020 0x4e77656e 0x61656d61
0xfffffcf20: 0x61616161 0x61616161 0x61616161 0x61616161
0xfffffcf30: 0x61616161 0x61616161 0x61616161 0x61616161
0xfffffcf40: 0x61616161 0x61616161 0x61616161 0x61616161
0xfffffcf50: 0x61616161 0x61616161 0x61616161 0x61616161
0xfffffcf60: 0x61616161 0x61616161 0x00006161 0x0000000b
0xfffffcf70: 0xf7fc4540 0x00000000 0xf7d9f4be 0x00000003
0xfffffcf80: 0x6861e4a0 0x3674656d 0x00003939 0x4e77656e
0xfffffcf90: 0x61656d61 0x61616161 0x61616161 0x61616161
0xffffcf9a0: 0x61616161 0x61616161 0x61616161 0x61616161
0xffffcf9b0: 0x61616161 0x61616161 0x61616161 0x61616161
0xffffcf9c0: 0x61616161 0x61616161 0x61616161 0x61616161
0xffffcf9d0: 0x61616161 0x61616161 0x61616161 0xffff0061
0xffffcf9e0: 0xf7fad000 0xf7fad074 0xf7fcb80 0xf7ffd020
0xffffcf9f0: 0x3a53ed8b 0x70c5c79b 0x00000000 0x00000000
0xffffd000: 0x00000000 0xf7fcb80 0xf7ffd020 0xaf4c0700
0xffffd010: 0xf7ffda40 0xf7da84a6 0xf7fad000 0xf7da85f3
0xffffd020: 0x00000000 0x56558ecc 0xffffd07c 0xf7ffd020
0xffffd030: 0x00000000 0x00000000 0xf7da856d 0x56558fc8
0xffffd040: 0x00000000 0x565560b0 0x00000000 0x565560db
0xffffd050: 0x565562bc 0x00000001 0xffffd074 0x00000000
0xffffd060: 0x00000000 0xf7fcaaa0 0xffffd06c 0xf7ffda40
0xffffd070: 0x00000001 0xfffffd1ab 0x00000000 0xfffffd1fc
0xffffd080: 0xfffffd20c 0xfffffd229 0xfffffd241 0xfffffd279
0xffffd090: 0xfffffd28e 0xfffffd29d 0xfffffd2a6 0xfffffd2be
(gdb)

```

username 0

username 1

username 2

Input buffer (new) controlled registers

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.

```

printf("%s\n", new);
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-VIVOB00K: ~/Desktop/C_codes/final/task_1_fn1
benediximuss@UGUR-ASUS-VIVOB00K:~/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-VIVOB00K:~/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: newNameaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

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

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

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

Exiting...
benediximuss@UGUR-ASUS-VIVOB00K:~/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:

```

benedixmuss@UGUR-ASUS-VIVOBOOK: ~/Desktop/C_codes/final/task_1_fn1
benedixmuss@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
benedixmuss@UGUR-ASUS-VIVOBOOK: ~/Desktop/C_codes/final/task_1_fn1 gdb -q task_1_function_0_patched
Loading 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/benedixmuss/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 ugurpasad!

Breakpoint 1, main () at task_1_function_0_patched.c:37
37      int option = -1;
(gdb) x/100x $esp
0xffffc020: 0xffffd000 0x00000010 0x00000000 0x00000000
0xffffc040: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffc060: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffc080: 0x7fc45400 0xffffffff 0x56555034 0x7fc66d00
0xffffc0a0: 0x7ffd6008 0x0000000b 0xffffcfc0 0xffffd0e8
0xffffc0c0: 0x00000000 0x00000000 0x00000000 0x0000000b
0xffffc0e0: 0x7fc45400 0x3674a56d 0x00000399 0x7fad0054
0xffffc100: 0x61736170 0x756d0034 0x74746172 0x00000069
0xffffc120: 0xffffcfd0 0xf7fad000 0xf7fd0020 0xf7da8519
0xffffc140: 0xfffffd1b1 0x00000070 0xf7fd0000 0xf7da8519
0xffffc160: 0x00000001 0xfffffd084 0xfffffd08c 0xfffffcff0
0xffffc180: 0xf7fad000 0x5655502b 0x00000001 0xfffffd084
0xffffc1a0: 0xf7fad000 0xfffffd084 0xf7fcdb00 0xf7fd0020
0xffffc1c0: 0x15db0d60 0x5f4d0770 0x00000000 0x00000000
0xffffc1e0: 0x00000000 0xf7ffcdb0 0xf7fd0020 0x801cfd00
0xffffc200: 0xf7fda400 0xf7da8406 0xf7fad000 0xf7da85f3
0xffffc220: 0x00000000 0x565550cc 0xf7fd0020 0xf7fd0020
0xffffc240: 0x00000000 0x00000000 0xf7da856d 0x56555fc0
0xffffc260: 0x00000001 0x565550b0 0x00000000 0x565550db
0xffffc280: 0x565562bc 0x00000001 0xfffffd084 0x00000000
0xffffc2a0: 0x00000000 0xf7fcaaa0 0xfffffd07c 0xf7fda400
0xffffc2c0: 0x00000001 0xfffffd1b1 0x00000000 0xfffffd1ff
0xffffc2e0: 0xfffffd21f 0xfffffd22c 0xfffffd244 0xfffffd244
0xffffc300: 0xfffffd291 0xfffffd2a0 0xfffffd2a9 0xfffffd2c1
(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:

```

benediximus@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: newNameaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
newNameaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
MENU
1-Show all users
2-Change your username
3-EXIT
Choose an option: 1
1
All users:
User 1 : ahmet699
User 2 : newNameaaa
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
0xfffffcf20: 0xf7ffd000    0x00000020    0x4e77656e    0x61656d61
0xfffffcf30: 0x61616161    0x61616161    0x61616161    0x61616161
0xfffffcf40: 0x61616161    0x61616161    0x61616161    0x61616161
0xfffffcf50: 0x61616161    0x61616161    0x61616161    0x61616161
0xfffffcf60: 0x61616161    0x61616161    0x61616161    0x61616161
0xfffffcf70: 0x61616161    0x61616161    0x00000616    0x0000000b
0xfffffcf80: 0xf7fc4540    0x00000000    0xf7d9fab5    0x00000003
0xffffcf90: 0x6861e4a0    0x3674656d    0x00003939    0x4e77656e
0xffffcfa0: 0x61656d61    0x756d0061    0x74746172    0x00000069
0xffffcfb0: 0xffffcfd0    0xf7fad000    0xf7ffd020    0xf7da8519
0xffffcfc0: 0xfffffdb1    0x00000070    0xf7ffd000    0xf7da8519
0xffffcfd0: 0x00000001    0xffffd084    0xfffffd08c    0xffffcff0
0xffffcfe0: 0xf7fad000    0x565562bc    0x00000001    0xffffd084
0xffffcfef0: 0xf7fad000    0xffffd084    0xf7ffc80    0xf7ffd020
0xffffd000: 0x15db0d60    0x5f4d0770    0x00000000    0x00000000
0xffffd010: 0x00000000    0xf7fccb80    0xf7ffd020    0x801cfd00
0xffffd020: 0xf7ffda40    0xf7da84a6    0xf7fad000    0xf7da85f3
0xffffd030: 0x00000000    0x56558ecc    0xfffffd08c    0xf7ffd020
0xffffd040: 0x00000000    0x00000000    0xf7da856d    0x56558fc8
0xffffd050: 0x00000001    0x565560b0    0x00000000    0x565560db
0xffffd060: 0x565562bc    0x00000001    0xffffd084    0x00000000
0xffffd070: 0x00000000    0xf7fc4aa0    0xfffffd07c    0xf7ffda40
0xffffd080: 0x00000001    0xfffffd1b1    0x00000000    0xfffffd1ff
0xffffd090: 0xfffffd20f    0xfffffd22c    0xfffffd244    0xfffffd27c
0xffffd0a0: 0xfffffd291    0xfffffd2a0    0xfffffd2a9    0xfffffd2c1
(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",  
};  
  
printf("All Notes:\n");
```

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 123AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
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: 123AAAAAAAAAAAAAAAAAA  
  
Updated notes:  
Note 1 : 750 TL  
Note 2 : Do CS437 LAB!123AAAAAAAAAAAAAAAAAA  
Note 3 : AAAAAAAAAAAAAA  
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:

```

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 -g
benediximuss@UGUR-ASUS-VIVOBOOK:~/Desktop/C_codes/final/task_1_fn2$ gdb -q task_1_function_1_vulnerable
Reading symbols from task_1_function_1_vulnerable...
(gdb) break 29
Breakpoint 1 at 0xi1297: file task_1_function_1_vulnerable.c, line 30.
(gdb) break 40
Breakpoint 2 at 0xi310: file task_1_function_1_vulnerable.c, line 40.
(gdb) run
Starting program: /home/benediximuss/Desktop/C_codes/final/task_1_fn2/task_1_function_1_vulnerable
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
All Notes:
Note 1 : 750 TL
Note 2 : Do C5437 LAB!
Note 3 : Password: 1234
Note 4 : Call UMMT for match

Breakpoint 1, main () at task_1_function_1_vulnerable.c:30
30      printf("Append to %d. note: ", TARGET_COMMENT);
(gdb) x/100x $esp
0xfffffc30: 0x00000000 0x00000000 0x00000000 0x00000000
0xfffffc40: 0x00000000 0x00000000 0x00000000 0xf77fd000
0xfffffc50: 0x20303537 0x00004c54 0x00000000 0x00000000
0xfffffc60: 0x00000000 0x43206f44 0x37333453 0x42414c20
0xfffffc70: 0x00000021 0x00000000 0x73736150 0x64726f77
0xfffffc80: 0x3231203a 0x00003433 0x00000000 0x6c606c143
0xffffc90: 0x55455520 0xf662054 0x616d2072 0x00686374
0xffffca0: 0xfffffcfe0 0xf7fbc66c 0xffffcfd0 0xf7fad000
0xffffcb0: 0xfffffd084 0xf7ffcb80 0xf7ffd020 0xf7da8519
0xffffcc0: 0xfffffd1b6 0x00000070 0xf7ffd000 0xf7da8519
0xffffcd0: 0x00000001 0xfffffd084 0xfffffd08c 0xffffcfff0
0xffffce0: 0xf7fad000 0x56556243 0x00000001 0xfffffd084
0xffffcf0: 0xf7fad000 0xfffffd084 0xf7ffcb80 0xf7ffd020
0xffffd000: 0x415f04d1 0x0b9c0ec1 0x00000000 0x00000000
0xffffd010: 0x00000000 0xf7ffcb80 0xf7ffd020 0xba6d0e00
0xffffd020: 0xf7ffda40 0xf7da84a6 0xf7fad000 0xf7da85f3
0xffffd030: 0x00000000 0x56558ec8 0xfffffd08c 0xf7ffd020
0xffffd040: 0x00000000 0x00000000 0xf7da856d 0x56558fc4
0xffffd050: 0x00000001 0x565560b0 0x00000000 0x565560db
0xffffd060: 0x56556243 0x00000001 0xfffffd084 0x00000000
0xffffd070: 0x00000000 0xf7fcaaa0 0xfffffd07c 0xf7ffda40
0xffffd080: 0x00000001 0xfffffd1b6 0x00000000 0xfffffd207
0xffffd090: 0xfffffd217 0xfffffd234 0xfffffd24c 0xfffffd284
0xffffd0a0: 0xfffffd299 0xfffffd2a8 0xfffffd2b1 0xfffffd2c9
0xffffd0b0: 0xfffffd2d6 0xfffffd2f8 0xffffd8e7 0xfffffd8f3

```

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:

[illegible]

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.



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

```

benediximuss@UGUR-ASUS-VIVOBOK: ~/Desktop/C_codes/final/task_1_fn2
benediximuss@UGUR-ASUS-VIVOBOK:~/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-VIVOBOK:~/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

```

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

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 arguments. The user will give an alphabetical string and a filename as arguments to the program. The program will take the first alphabetical string argument and convert it to hexadecimal based on ASCII. Then the result will be written into a new file named as second argument. 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

```
#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",
            argv[0]);
        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:

```
benediximuss@UGUR-ASUS-VIVOBOK: ~/Desktop/C_codes/final/task_2
benediximuss@UGUR-ASUS-VIVOBOK:~/Desktop/C_codes/final/task_2$ gcc cmd_injection.c -o cmd
benediximuss@UGUR-ASUS-VIVOBOK:~/Desktop/C_codes/final/task_2$ ./cmd uguroztunc output.txt ; ls -l
756775726f7a74756e63total 52
-rwxr-xr-x 1 benediximuss benediximuss 16256 Dec 24 08:06 a.out
-rw-r--r-- 1 benediximuss benediximuss 2387 Dec 24 07:56 buffer_overflow.c
-rwxr-xr-x 1 benediximuss benediximuss 16408 Dec 24 18:33 cmd
-rw-r--r-- 1 benediximuss benediximuss 938 Dec 24 18:31 cmd_injection.c
-rw-r--r-- 1 benediximuss benediximuss 20 Dec 24 18:33 output.txt
-rw-r--r-- 1 benediximuss benediximuss 112 Dec 24 08:06 payload.txt
benediximuss@UGUR-ASUS-VIVOBOK:~/Desktop/C_codes/final/task_2$
```

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>

 You

I want you to write a simple C program to keep track of shopping items for each day

< 2/2 > 

 ChatGPT

Certainly! Here's a simple C program that allows you to keep track of shopping items for each day. This program uses structures to represent shopping items and an array to store items for each day.

```
c Copy code

#include <stdio.h>
#include <stdlib.h>

#define MAX_ITEMS 10
#define MAX_DAYS 7

// Structure to represent a shopping item
struct ShoppingItem {
    char name[50];
    float price;
};

// Structure to represent a day's shopping list
struct ShoppingDay {
    struct ShoppingItem items[MAX_ITEMS];
    int itemCount;
};

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

Message ChatGPT...



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];
```

```
61         if (day >= 1 && day <= MAX_DAYS) {
62             printf("Enter item name: ");
63             scanf("%s", itemName);
64             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:

```

8 // Function to add a shopping item to a day's list
9 void addItem(struct ShoppingDay *day, const char *name) {
10     if (day->itemCount < MAX_ITEMS) {
11         snprintf(day->items[day->itemCount].name, sizeof(day->items[day->itemCount].name), "%s", name);
12         day->itemCount++;
13         printf("Item added successfully.\n");
14     } else {
15         printf("Error: Maximum number of items reached for the day.\n");
16     }
17 }
18

```

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:

```
final > task_2 > payload.txt
```

1	1
2	1
3	peynir
4	1
5	1
6	zeytin
7	1
8	1
9	aa
10	2
11	3



Here is the result:

```
benediximuss@UGUR-ASUS-VIVOBOK: ~/Desktop/C_codes/final/task_2
benediximuss@UGUR-ASUS-VIVOBOK:~/Desktop/C_codes/final/task_2$ gcc buffer_overflow.c
benediximuss@UGUR-ASUS-VIVOBOK:~/Desktop/C_codes/final/task_2$ ./a.out < payload.txt
1. Add item to a day's shopping list
2. Print all days' items
3. Exit
Enter your choice: Enter the day number (1-7): Enter item name: Item added successfully.
1. Add item to a day's shopping list
2. Print all days' items
3. Exit
Enter your choice: Enter the day number (1-7): Enter item name: Item added successfully.
1. Add item to a day's shopping list
2. Print all days' items
3. Exit
Enter your choice: Enter the day number (1-7): Enter item name: Item added successfully.
1. Add item to a day's shopping list
2. Print all days' items
3. Exit
Enter your choice: Shopping list for Day 1:
1. peynir
2. zeytin
3. aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Shopping list for Day 2:

Shopping list for Day 3:

Shopping list for Day 4:

Shopping list for Day 5:

Shopping list for Day 6:

Shopping list for Day 7:

1. Add item to a day's shopping list
2. Print all days' items
3. Exit
Enter your choice: Exiting program.
*** stack smashing detected ***: terminated
Aborted
benediximuss@UGUR-ASUS-VIVOBOK:~/Desktop/C_codes/final/task_2$
```

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.