Part 3 - 15 points

1. Run with short inputs (e.g., username ≤8 chars, password ≤8 chars) (5 Points)

Q. What do you observe about the flag?

ANS: I could see that the flag remained at 0.

Q.Did the Red LED switch ON?

ANS: The Red LED didn't switch on

Q. What happens to the system password?

ANS: The system password is still protected and has not been revealed yet. It prints to the console that "System Password protected".

2. Break the vault! (10 Points)

Find ways to break the vault i.e. enter username and password combinations that cause:

I. Flag to become non-zero

II. Red LED to switch ON

III. System password to be compromised (printed)

Q. Write a short answer describing how you managed to compromise the vault (describe at least three different ways and their consequences)

Hint: see what happens when you overflow the username vs overflowing the password and observe different ways in which the application and system password is compromised.

ANS:

+ Method 1: Overflow the password: Since the buffer for the password is 8 bytes and has the address next to the flag, when we enter a password that has a length more than 8 characters, then the character would get overflowed to the flag, which makes it non-zero and makes the system print out the system password and switch the RED_LED on.
+ Method 2: Overflow the username: Since the buffer for username is next to the buffer for password and then buffer for flag, if we enter a username that overflows through the password and through the flag, which means that we enter a password that is more than 16 characters long, the gets() function will write the user input into 8 buffers of username and fill 8 buffers of password, and the left is filled in the buffer for flag. So the buffer for the flag would now contain a non-zero value, which would lead to the print out of the system password and make the Red_LED switch on.
+ Method 3: Long overflow: For this method, we input an input for username with a length of more than 20 characters, which means that the byte for the username would fill the password buffer and the flag buffer. After filling those two buffers, it would fill the buffer for the system api key and eventually change the key. Since the flag has been overwritten by a non-zero value from the username, the system would still switch the RED_LED on and print out the system api key. This would lead to a violation of both confidentiality and integrity.
+ Method 4: 8-16 characters of username: Since the buffer for the username is 8 bytes and has the address next to the password, if we enter a username that has 8 to 16 characters, although it will not overflow to the flag and change it due to the password buffer also having 8 bytes, the username will overflow into the password and place its null terminator there. Later, when we input the password, the password buffer will be overwritten, and the null terminator will be placed at the end of the password. Therefore, when the code prints out the current states of the username, the

username will be both our input username and the password concatenated together. Though this does not lead to a compromise in the system password, it still enables a security hole in integrity.

Part 5 - 5 points

5.1: Attach a screenshot of your gdb console showing

i. Raw memory when there is no buffer overflow

```
=== VaultApp Login ===
Enter Username:
tdau
Enter Password:
1234

Breakpoint 2, main () at lab1.c:41
41          printf("\n=== Current States and Addresses ===\n");
(gdb) x/32bx &data.username
0x555555570078 <data>:  0x74    0x64    0x61    0x75    0x00    0x00    0x00    0x00
0x555555570080 <data+8>:        0x31    0x32    0x33    0x34    0x00    0x00    0x00    0x00
0x555555570088 <data+16>:       0x00    0x00    0x00    0x00    0x74    0x64    0x61    0x75
0x555555570090 <data+24>:       0x73    0x65    0x63    0x72    0x65    0x74    0x31    0x32
(gdb)
```

ii. Raw memory when there is an overflow in the username

```
=== VaultApp Login ===
Enter Username:
tdautdautdautdau
Enter Password:
1234

Breakpoint 2, main () at lab1.c:41
41          printf("\n=== Current States and Addresses ===\n");
(gdb) x/32bx &data.username
0x555555570078 <data>:  0x74    0x64    0x61    0x75    0x74    0x64    0x61    0x75
0x555555570080 <data+8>:        0x31    0x32    0x33    0x34    0x00    0x64    0x61    0x75
0x555555570088 <data+16>:       0x00    0x00    0x00    0x00    0x74    0x64    0x61    0x75
0x555555570090 <data+24>:       0x73    0x65    0x63    0x72    0x65    0x74    0x31    0x32
(gdb)
```

iii. Raw memory when there is an overflow in the password

```
=== VaultApp Login ===
Enter Username:
tdau
Enter Password:
123456789

Breakpoint 2, main () at lab1.c:41
41          printf("\n=== Current States and Addresses ===\n");
(gdb) x/32bx &data.username
0x555555570078 <data>:  0x74    0x64    0x61    0x75    0x00    0x00    0x00    0x00
0x555555570080 <data+8>:        0x31    0x32    0x33    0x34    0x35    0x36    0x37    0x38
0x555555570088 <data+16>:       0x39    0x00    0x00    0x00    0x74    0x64    0x61    0x75
0x555555570090 <data+24>:       0x73    0x65    0x63    0x72    0x65    0x74    0x31    0x32
(gdb)
```

iv. Raw memory when the flag is damaged

```
=== VaultApp Login ===
Enter Username:
tdau
Enter Password:
1234567890

Breakpoint 2, main () at lab1.c:41
41          printf("\n=== Current States and Addresses ===\n");
(gdb) x/32bx &data.username
0x555555570078 <data>:  0x74    0x64    0x61    0x75    0x00    0x00    0x00    0x00
0x555555570080 <data+8>:        0x31    0x32    0x33    0x34    0x35    0x36    0x37    0x38
0x555555570088 <data+16>:       0x39    0x30    0x00    0x00    0x74    0x64    0x61    0x75
0x555555570090 <data+24>:       0x73    0x65    0x63    0x72    0x65    0x74    0x31    0x32
(gdb)
```