CS6570 Secure Systems Engineering

Lab 2

Akshith Sriram CS19B005, Bhanu Shashank CS19B043

Look into `Lab2_1.c` for the example code file.

Q1. List all the bugs/vulnerabilities present in the program.

1. In the `first_chars` function, the declared size of the buffer `l.words` is 12 bytes, while a max limit of 16 bytes (hexadecimal `0x10`) is allowed to be read through the `fgets` function. This can be used to overflow the buffer.
2. Unnecessary declaration of the buffer `eg_file` makes it easy for the attacker to access a pointer to the path of the password file.
3. `buf` is a character array of size of 100 bytes, of which at least one character should be null for `printf("The file has %s", buf)` to work correctly. If the size of the file `f` exceeds 100 characters, then the first 100 characters of `f` are directly copied into `buf`, and there is no guarantee for the existence of a null character among them. In such a case, `fread(&buf, sizeof(char), 100, f)` also becomes a vulnerability.

Q2. How is the stack protected from overflow in the program ?

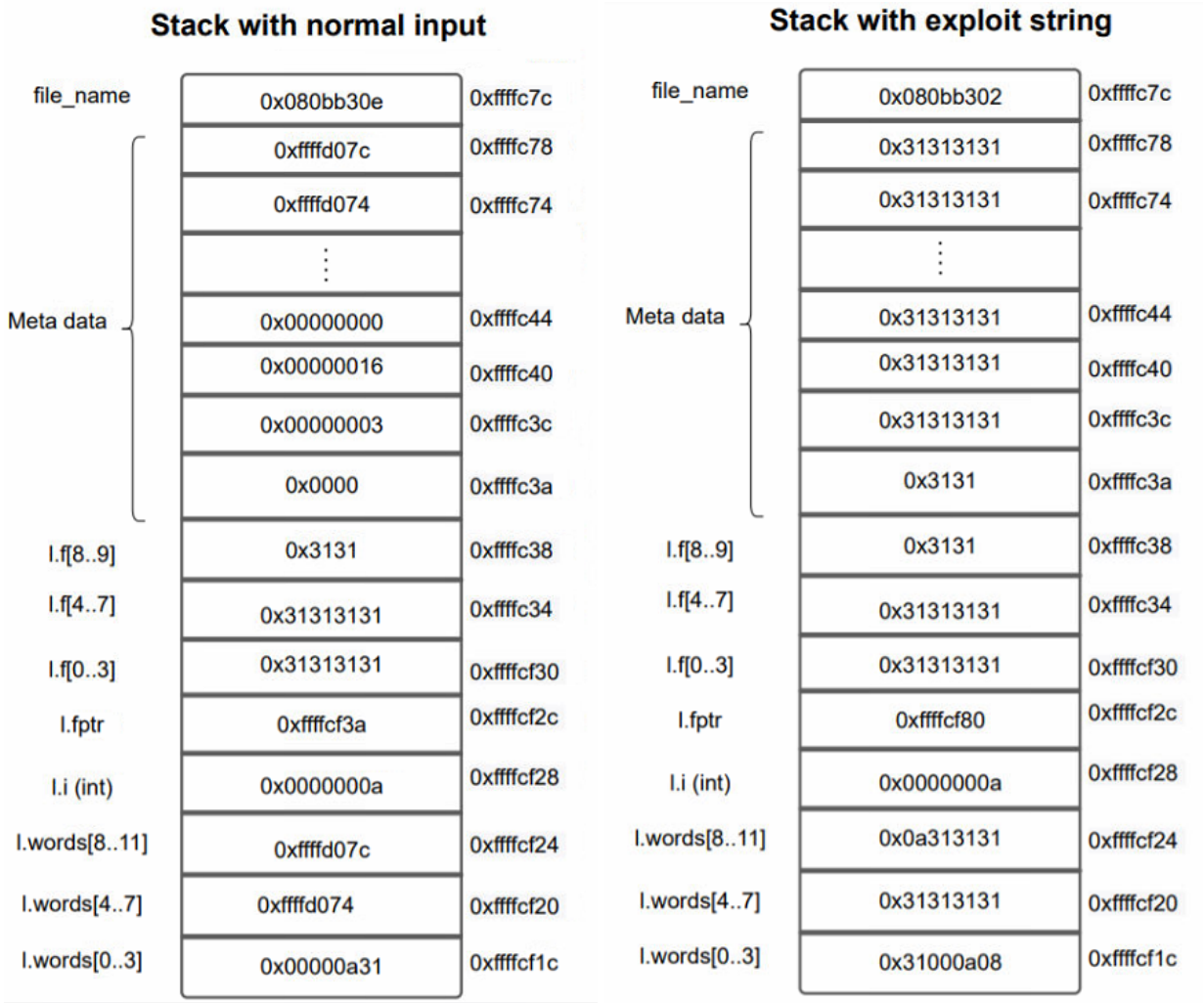
After reading 10 words of input from the user, the program sets the last byte of `l.f` to a null character. This ensures that the instruction `printf("First characters of all words:\n %s\n", l.f)` in the next line works correctly irrespective of what the user gives as input words.

Q3. List 3 methods to protect the given code.

1. Using `fgets(l.words, 11, stdin)` in line 18 of the program to avoid any sort of buffer overflow in the program.

2. Inserting a canary in the `first_chars` function to ensure that the return address and frame pointer can't be modified.
3. Defining the iterator `l.i` outside the structure (out of overflow range) so that the attacker can't modify it using a buffer overflow.

Q4. Draw the different snapshots of the content of the stackframe along with different values of registers like ebp, esp, etc for each step of the attack.



esp = 0xffffceb0

esp = 0xffffceb0

Normal input: All the ten words given as input are 1s.
Exploit String: Lab2_1_CS19B043_CS19B005.exp

Q5. As an attacker, how can you create an exploit to print the password file ?

The buffer `l.words` is declared to store only 12 characters, but the input reading `fgets(l.words, 0x10, stdin)` allows 16 bytes to be stored in the buffer. This allows an attacker to subvert execution.
In the stack, the buffer `l.words` is immediately above the iterator `l.i` that was used for controlling the number of words taken as input. Through carefully tailored inputs, we reset the variable `l.i` to 0, resulting in the for loop executing more times than expected (10). The number of iterations can be controlled by the attacker as he likes.

Each time the `for` loop is executed, the pointer `l.fptr` (which initially points to the location of `l.f` buffer) increments by 1. So, if the `for` loop executes 15 times, the value in the location f[15] is updated to the value of the first character of the 15th input. Using this, we updated the `file_name` pointer to point to the string `"/etc/passwd"`.

When the file `file_name` is read, the contents of the password file are printed instead of those in `"theflag.txt"`.
The offsets in case of the C program `Lab2_2.c` and the executable (`CS19B043_CS19B005_2`) differ, so the payload was adjusted accordingly to modify the string `file_name`. The exploit string for the C file is `Lab2_1.exp` and for the executable is `Lab2_1_CS19B043_CS19B005.exp`

Look into `Lab2_2.c` for the example code file.

Q1. List all the bugs/vulnerabilities present in the program.

1. The declaration of the function `shell` (which was not used in the program) makes it possible for an attacker to subvert execution and create their own shell. If this declaration is removed, it won't be possible to open a shell easily using buffer overflow.
2. Declaration of the `exec_string` global variable (which is also not used) can be exploited along with buffer overflow to create a shell.
3. The use of `strcpy` in the `get_name` function is a vulnerability, as it does not limit the number of bytes copied from input into `buf`. This bug can be used to overflow the buffer `buf` and access other parts of the stack.

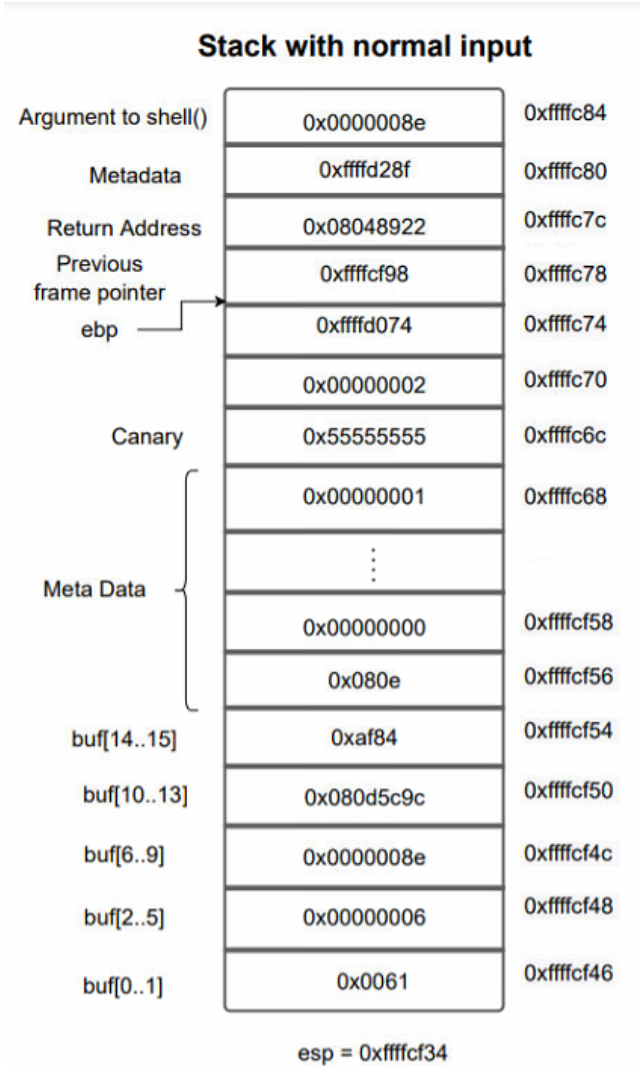
Q2. How is the stack protected from overflow in the program ?

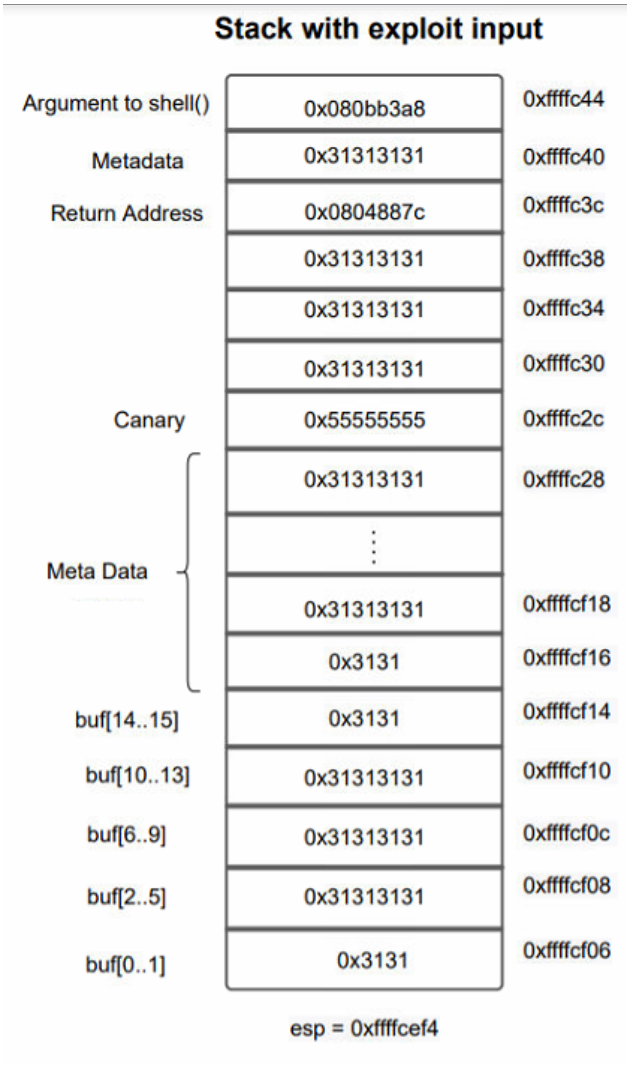
The stack is protected in the program using a canary defined in the `get_name` function. Before the function returns, the value of the canary is checked for modifications due to a buffer overflow. If the canary value is changed, the program exits without further execution.

Q3. How is this protection different from what we studied in class ?

In this case the canary is inserted by the programmer himself. According to what we studied during the class, canaries are inserted in a function by the compilers (automatically) if they detect a potential for buffer flow due to usage of functions like `fgets`, `strcpy`, `gets`, etc.

Q4. Draw the different snapshots of the content of the stackframe along with different values of registers like ebp, esp, etc for each step of the attack.





Normal input = ‘a’

Exploit string = Lab2_1_CS19B043_CS19B005.exp

Q5. As an attacker, how can you create an exploit to print the password file ?

1. The idea in the attack is to modify the return address of the `get_name` function to start the execution of the `shell` function instead of returning to `main`. We have to find the location of the `exec_string` buffer, as it contains the path `"/bin/sh"`. We have to insert this address as an argument to the `shell` function, so that it can pick up that address as the first argument `input` and execute `system(input)`.
2. We can use `gdb` to get the location of `exec_string`. By overflowing the buffer carefully, so that the value of the canary does not change, we can modify the return address and provide the location of `exec_string` as an argument.
3. Then the program starts executing `shell` function, and a shell is created.
4. The offsets are different for `Lab2_2.c` and `CS19B043_CS19B002_2` files, and the exploit strings are adjusted accordingly. The exploit string for the C file is `"Lab2_2.exp"` and for the executable is `"Lab2_2_CS19B043_CS19B005.exp"`.