**Secure Systems Engineering (CS6570)**

**Assignment-2**

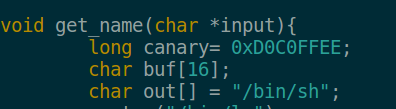
**NAME**: Mohammed Umair Pandit

**ROLL NO:** CS23M037

There are quite a few vulnerabilities in the provided code, there are as follows:

**Constant Canaries:**

These fixed Canaries values are used in stack protection mechanisms to detect buffer overflow attacks. These Canaries are predetermined values placed in memory locations before critical data structures making sure those are not fiddled with or tampered with. During program execution if the canary value is modified, it is indicating that a there is a possibility that a buffer overflow has occurred and the program will terminate or take necessary action to prevent exploitation and send and “stack smashing detected” message to the user.

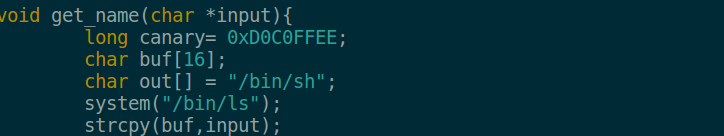
This is the reason canaries are used, and in the program a fixed canary value is used , which could prove to be less reliant than an canary which is generated by a compiler which is usually unique and random. The fixed canary value here gives and opportunity to manipulate and read the stack even more clearly, since we will not have to search or dig through the stack to identify the canary value.

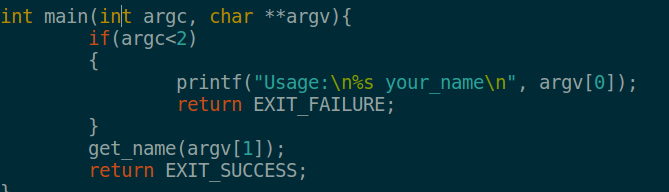
This vulnerability can be fixed by enabling f-stack-protector while compiling the code or using it in the MakeFile, what this flag or command ensures is that it provides canary in the stack and then make sures that the canary value has not changed or any overflow has not occurred, usually it does this by using bit manipulation technique to check before and after compiling canary values.

**Shared Library Functions:**

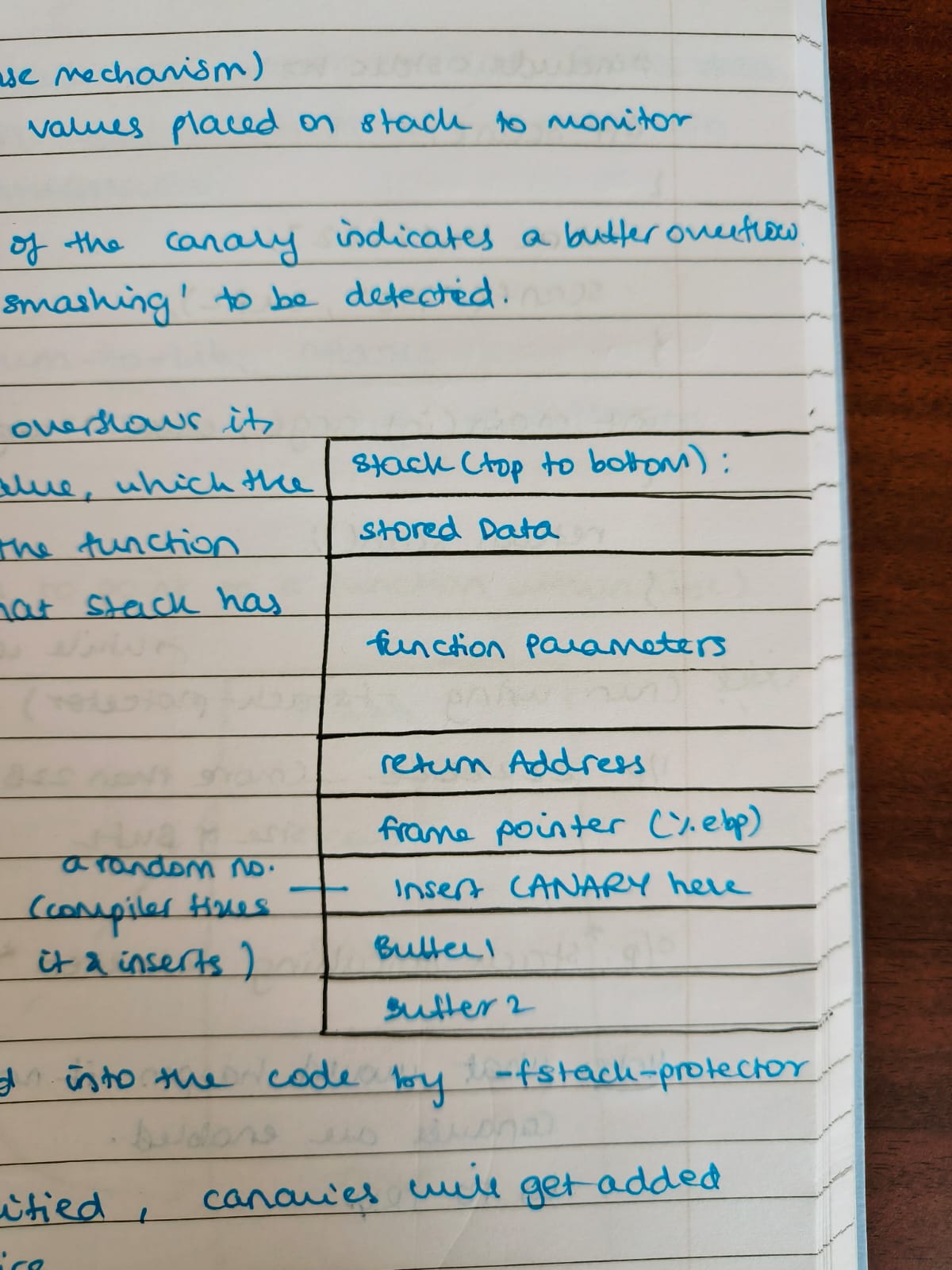
Since the program given is coded in C language, we are bound to use the functionality the language has to offer, and by that reason, the code contains functions which are used from the library of the C language. Now these functions are necessities, some not, but the point is that these functions are needed to use the language effectively, which in turn is the reason these vulnerabilities exist in the first place.

This Vulnerability is famously also known as ‘Return-to-Libc’ attack, where the attacker uses the function of Libc from its library to subvert the execution of the program out of the stack in the code segment , where the attacker can run its exploit/malicious code, and that is because even though we have enabled non-executable stack and made canaries, we are not executing the stack in the first place, just overflowing the return address with the Libc function call to subvert the execution of the program and return to a function in Libc where it can execute its exploit code. And this is exactly what we have carried out in this task as well.

In this program we have used a common Libc function call called ‘System(parameter)’ which takes a pointer to a string as a parameter and executes it.

Above is the snippet of the code given, we have retrieved the address of the system by debugging it using GDB and overflowed the return address to the main which is present as shown below after get\_name function. Which subverts to Libc function call and after passing desired parameters (i.e exploit code) exploitation is carried out.

//insert assembly return address snippet



We can summarize our ‘Ret2Libc’ Attack using above stack representation. Since we Know the size of Buffers in our locals of our program we overflow the buffer with some ‘nops’ (“A” in this case) until canary value is met, then leave / replace the same canary value after overflowing the buffers , then insert again a ‘nops’(“A” again) again to fill the frame pointer ‘ebp’ usually it is of 4 Bytes in size, then we input or add the Little Endian format of address of the Libc Function call ‘System’ in place of return address. After the ‘System’ address is overflowed in return address we insert the address of the ‘Exit’ function to make a clean exit out of the program, then we insert the shell code we want to execute after exit address, which the function ‘System’ will take as parameter and then when return address is hit, it subverts to ‘System’ function call where the parameter of shell code passed above it in the ‘function parameters’ is executed by the ‘System’ function call.

These Ret2Libc Attacks can be prevented by implementing ASLR (Address Space Layout Randomization) which by default is included in current systems. What ASLR does is it randomizes the address space layout of the process , such that each execution would have a different memory map. Thusi in turn making it difficult for the attacker to run exploits.