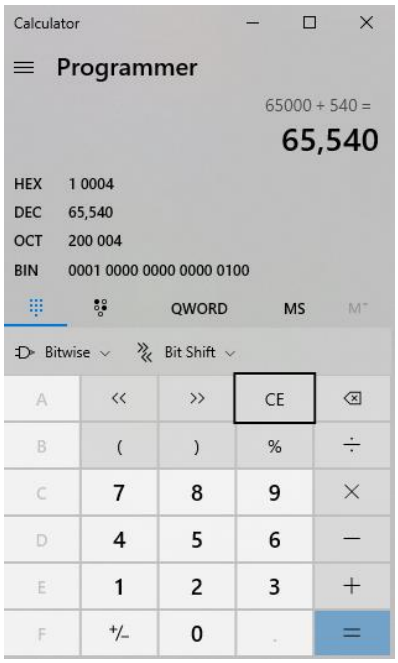
1. Problem Statement:
    a. You have been given 2 programs called Exam3_part1.c and Exam3_part2.c.
    b. The first contains a few blocks of code each with a different problem usually encountered. For this part all you need to do is run each chunk and explain what problems is occurring and why it is occurring the way it is.
    c. The second part contains a full program with a fatal flaw. It takes a command line argument. You must document the code and figure out what flaw is occurring and why it is happening. Once you have documented the code and figured out what it is doing, explain how it can be changed to prevent the bug.
2. Introduction:
    a. In C, each variable has a corresponding data type. Every data type has particular operations that can be carried out on it and distinct memory requirements. The type of data that the variable can store, such as integer, character, floating point, double, etc., is specified. Integer data type stores only whole number not decimal numbers. Signed int allows for positive and negative whole numbers while unsigned int only allows for positive whole numbers. There are different types of integers such as short int and long int. The different between these two is the memory size a short int has two bytes while long int have four bytes. These different memory sizing can cause some vulnerabilities to our program because integer range checking has not been consistently used in the development of the of C and C++.
3. Part 1:
    a. **I thought it was a waste of space to insert images of the print outs of the blocks for part one**
    b. Block 1:
        i. The integer error for this block is overflow error with an unsigned short int. This error occurs because integer is increased beyond its maximum value. For an unsigned short int the maximum value is 65535, holds exactly two bytes.

1. 

    a. Our variable cannot store 20 bits so the result will only be four due to the memory size of the unsigned short int. This overflow is causing an overflow so the solution will remain as four unless we make this short int into a long int. This is a vulnerability because the program continues to execute with the incorrect values without any warnings, developers will not easily recognize these types of errors without prior knowledge.

c. Block 2

The integer error for this block is underflow with unsigned and signed short int. When an underflow occurs, it does a silent wraparound to the maximum value for the data type. In this block we are dealing with unsigned short int and signed short int which their minimum values are 0 and -32768 respectively.

For us -= 1 it will be us = 0-1 which gives us -1 since our minimum is 0 variable us will be wraparound to our maximum value for unsigned short int which is 65535.

For ss -= 1 it will be ss = -32768 - 1 which should give us -32769 but that is out of bound for our range for signed short int which is -32768. This will cause us to have to wraparound to the maximum value of 32767.

d. Block 3

    i. The integer error for this block is integer promotion. Integer promotion happens when a variable is converted from a small-sized integer to a larger one. In this block, we are converting short integers to regular integers. Since the large-size integer can store a superset of values of the smaller type, type promotion is not an issue. The problem is that when printing out

in different formats such as decimal, hexadecimal, and octal, etc. For this block when printing out the hex of the int it differs from the short int hex.

e. Block 4

i. The integer error for this block is sign error due to integer demotion. Integer demotion is when a variable of a higher size integer is converted to a lower-sized integer. In this case with block four, we are going from unsigned integer --> unsigned short --> unsigned char. In our third prinf we see a signed change because going from four bytes to two bytes, we are aware that the last byte identifies the sign of the variable. In the case of unsigned numbers, this could lead to information loss. Demotion can cause an unanticipated change in signs for signed numbers.

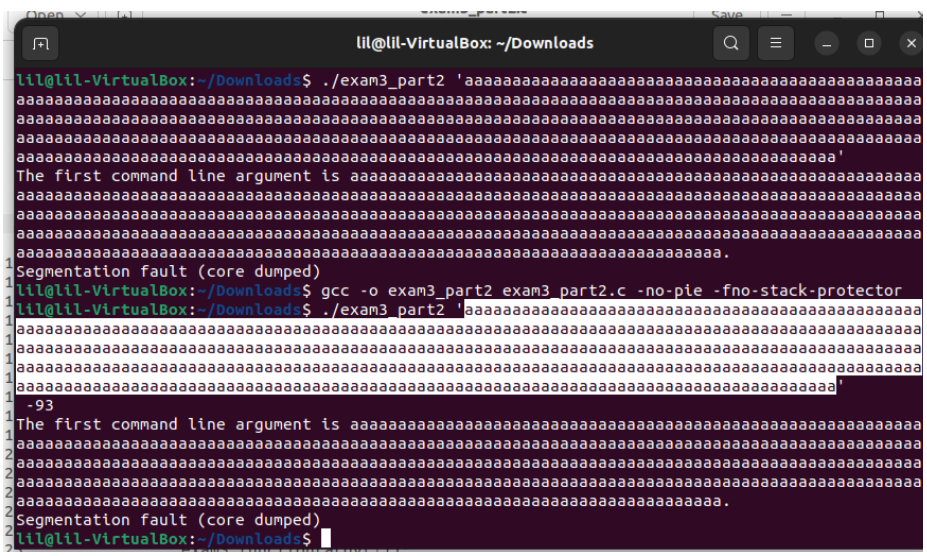
f. Block 5

i. The integer error here is the issue with the logical operation that comes with promoting values. Promotions can occasionally have unexpected results, such as block two. In this block, we have an issue with comparing the two variables since the unsigned char is less than an integer value it is promoted to an integer value for logical operation.

4. Part 2:

a. Figure out what fatal flaw is occurring and why

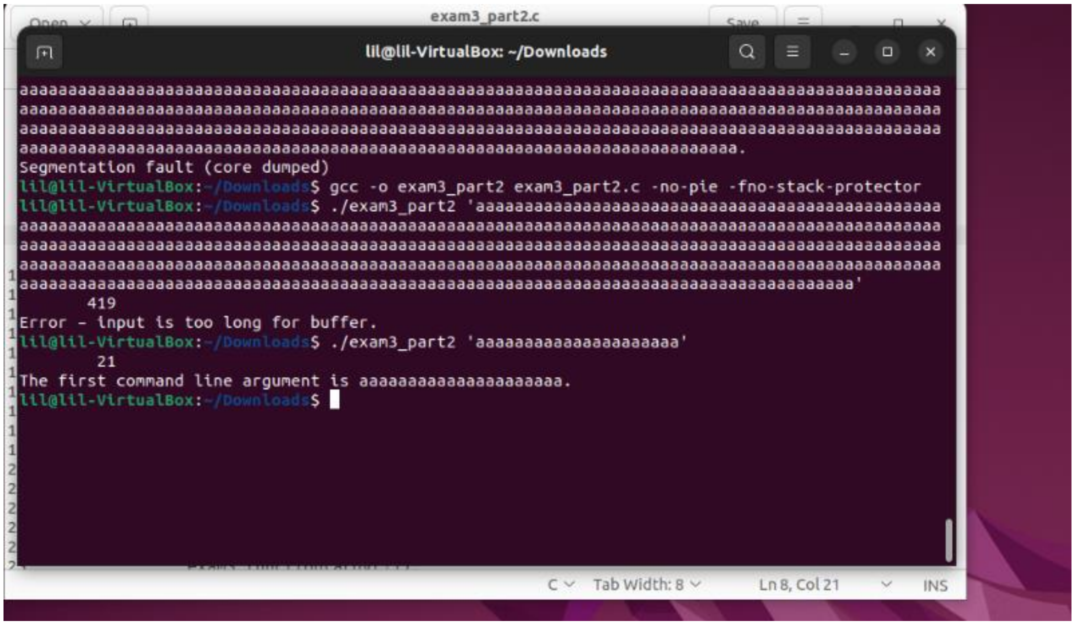
i. For this program, the issue was the input_len variable data type. The original data type was a char the maximum size is 127 so when user input is larger than that size it causes an integer overflow which causes the char value to wraparound to a negative number. So, the if statement “if (input_len < 32)” will always execute true causing our program to always print out the if statement. I found the input_len variable was the main issue by printing out the value of the variable with various inputs.



b. Give solutions to solve the problem and modify the code to reflect this change.

The change I made to the code is to change the datatype of variable input_len to integers instead of a char. I did this because integer has a

maximum value of 2147483647. This makes it harder to have another wrap-around issue so the if statement is correctly executed. Also, I added another condition to the if statement that checks if the length is less than 0 which indicates an integer overflow. t first I tried to just change the “32” to “33” in the if statement that of course didn’t work.

c. 

5. Conclusion and analysis of solutions
    a. The main purpose of this exam was to further understand integer vulnerabilities. For instance, we dealt with overflow, underflow, promotion, demotions, etc. Understanding how the memory sizes of various integer types is important to understanding the vulnerabilities. The simple lack of understanding integer can cause for your program to be easily exploited.
6. Sources:
    a. https://learn.microsoft.com/en-us/cpp/c-language/cpp-integer-limits?view=msvc-170