4190.308
Computer Architecture
Spring 2019

Bomb Lab Report
03/25 ~ 04/08

Name: Jinje Han
Student ID: 2018-18786

Introduction

In this lab, a program that gets several inputs is given. The program is similar to a bomb, which means it gets several inputs from the user and explode unless it is the correct input that defuses the explosion. Students have to disassemble the program into assembly file and find the correct input by reading assembly code and testing some inputs.
The main object of this lab is to check out if the student understands the grammar of assembly code. Students can know how their inputs will be treated and affect the program by reading the code. In addition, they can indirectly know how programs they make are translated into assembly language.

Solution

Phase 1

There was a function named <strings_not_equal>, having the value of %rbx as the first argument and 0x402560 as the second argument. I checked what's in the memory with 'x/s 0x402560', and there was string, "Brownie, you are doing a heck of a job." Using 'x/s $rbx' I found that the string I input was there. Due to the name of the function, I input the same string in the memory, and the first bomb was defused.

Phase 2

Due to the name of function <read_seven_numbers>, I input seven integers. I saw the program getting the value in the stack, so I checked several parts of the stack and verified that the integers I wrote was at there. The program got the last integer input, subtracted 0x400 from it and (unsigned) compared with 0x3fc00. The last integer must be bigger than 0x40000. Integer smaller than 0x400 would also pass the comparsion because of negative overflow, I just wrote bigger integer in order to not set the condition code (Actually, it has nothing to do with the result).
Next, the second to last integer must be as big as 5 than the last integer. The gap should have become smaller as 1 it comes forward. The final answer was '262160 262160 262159 262157 262154 262150 262145(=0x40000)'

Phase 3

In order to verify sscanf function's input form, I checked it with 'x/s $rsi'. There came out '%d %c %d'. Therefore, the function needed integer, character, integer in order. If the input is not enough, the bomb exploded. The first integer input must be smaller than 8 in order to not jump to the explosion command. I looked for the reason, and found that the position to jump was determined by that value. In other words, it was a switch statement. Instead of checking all choices, I input integer 3 as the test. In that case, I could avoid the explosion if I input the character that matches with 0x61 in ascii table and integer 0x113 in order. I wrote 'a' and '275' after the number 3, and the bomb was defused.

Phase 4

Like the third bomb, I verify the input form first. The inputs must have been three integers; the bomb explodes when it's less than three. I also checked the stack, and found that the input integers are at %rsp+4, +8, +12 respectively. The function compared the first integer minus four with 0xC(=12) and exploded the bomb if smaller. Therefore, I set the integer moderately big, into 67.

Next, there was a function <func4> that had the value of %rsp and $6 as arguments. Having %rsp as argument, I expected that the function will access to the input integers. Since <func4> was recursive function, it was hard to analyze what the functions do, but it was verified that the function returns 20 times third input integer. This calculation is done one more time and added, so at last 40 times the input is put in %rax.

Finally, this value is compared with the second input integer. The bomb is defused if the value is same. The final input I used is '67 280 7'.

* The answer of this phase had too large range. I guessed that some input value will lead to the secret phase, but it require me so much work to assure it. I would be happy if my guess is right. :)

Phase 5

There was a function named <string_length> at the very first of the code. Right after that, the value of %eax was being compared with $6. At this point, I realized that the function is getting a string as input and having its length as the return value. Then, each characters in the string was in AND operation with 0xf, which is same as dividing the corresponding ascii value

by 16 and getting the remainder of it. Using that values as indexes, characters were gotten from an array in the memory. If the string made is same as a given string ‘flyers’, the bomb was defused. All I did was checking all characters in the array and writing appropriate character inputs. The input was ‘ionefg’.

Phase 6

In this code, I could fully anaylze what this program does by reading the disassembly code. First, the <read_seven_numbers> gets seven inputs. The numbers are saved in the stack. All numbers has to be positive and smaller than 8, and same number must not exist. Second, the numbers are subtracted from 8. Third, using the numbers as indexes, the program gets integer values from array in the memory (specifically, from 0x006402e0 to 0x00640340) and place them at the stack, replacing the original values. Last, the program checks if the values are in descending order in the stack. Therefore, I should have modified the order of integer 1 to 7 for the values of array in the memory come to the stack in order. The order was ‘3 1 7 6 5 2 4’.

Conclusion

From this lab, I could develop my ability to analyze assembly code and knew how register deals with data. Especially, I could fully comprehend how %rsp and %rbp assist each other and are used to find data on stack.
The most struggling part was which input should I write. First, I tried to look into sscanf function, but it was too complicated. I could infer some input functions by name, but sscanf was impossible. I struggled a lot, and at last find the solution by googling the function. From this, I realized that searching what I need is also an important virtue for a programmer.