Lab: Stack and Stack Frame

Mayur Suresh

CSC 472-02: Software Security 2

Date: October 2nd, 2022

In this lab we will use MobaXterm to go through the lab1 C program which has 2 functions, the main function and the add_plus1() function. We use Badger CTF and use gdb to disassemble the lab1 program to see what is inside a program with assembly code. We will also see how memory is handled and moved according to the program with eight 34-bit registers, the stack, and the stack frame. We can see how each line of the program works but setting but a break line with the br* memory address and then running (r) in order to run the following line the si command is used to see how each line interacts behind the scene.

Assembly code that creates the stack frame followed by their memory addresses are:
Source Image 1
0x08049195: PUSH EBP
0x08049196: MOVE EBP, ESP

The meaning of the lines:
0x080491b1: mov DWORD PTR [ebp-0x10],0x5
0x080491b8: mov DWORD PTR [ebp-0xc],0x6
The first line means moving the 32-bit representation of 0x5 into 4 bytes at address [ebp-0x10]
The second line means moving the 32-bit representation of 0x6 into 4 bytes at address [ebp-0xc]. So, this means the that int x is 5 and int y is 6 as shown in the lab1.c.

The reason why on the stack that the values of 6 and 5 are listed twice on the stack is because the initial 6 and 5 at the addresses (according to Source Image 2) 0xffffd6b8 and 0xffffd6bc are the storing of the 6 and 5 to the int a and int b addresses in the main function. While the other 5 and 6 at the addresses (according to Source Image 2) 0xffffd6a8 and 0xffffd6ac are there when the add_plus1 function is called in the main function and then passing through the a and b variables to this function.

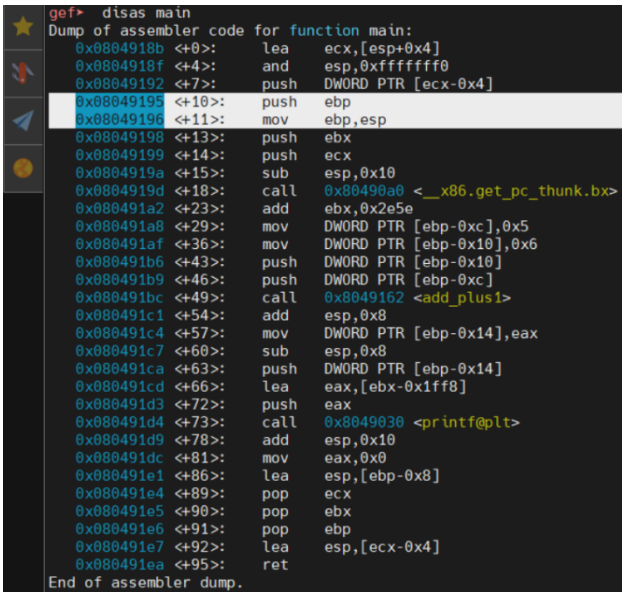
Upon disassembling the add_plus1() function the first 2 assembly code are the code that are implemented in order to start the stack frame for the add_plus1() function.

After going through line by line as shown in Source Image 3 that after the ADD EAX, EDX it is inferred that the result is stored at the Register EAX as also shown on the stack where the EAX register has the summation.

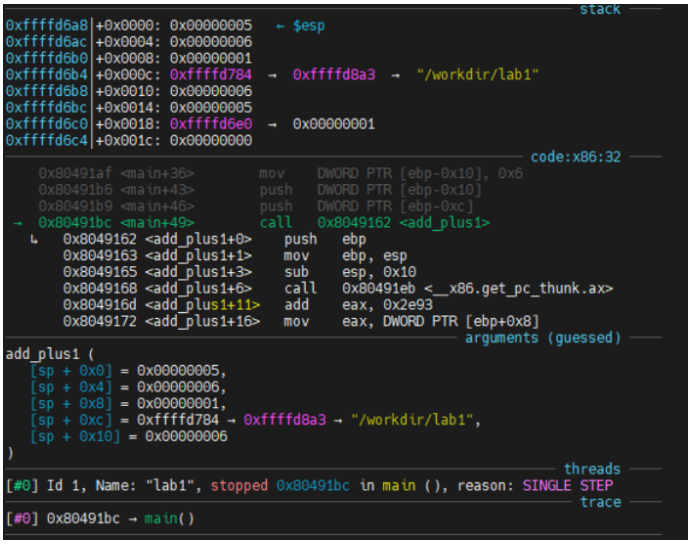
All in all, this lab's purpose was to show what happens behind the scene of a program. How when data stored in a variable is moved to various registers to then push it to the method when called. This lab also showed us how the stack and the stack frame behave when data is moves around. It also showed how when a function is called a new stack frame is implemented.

Source Images Below.

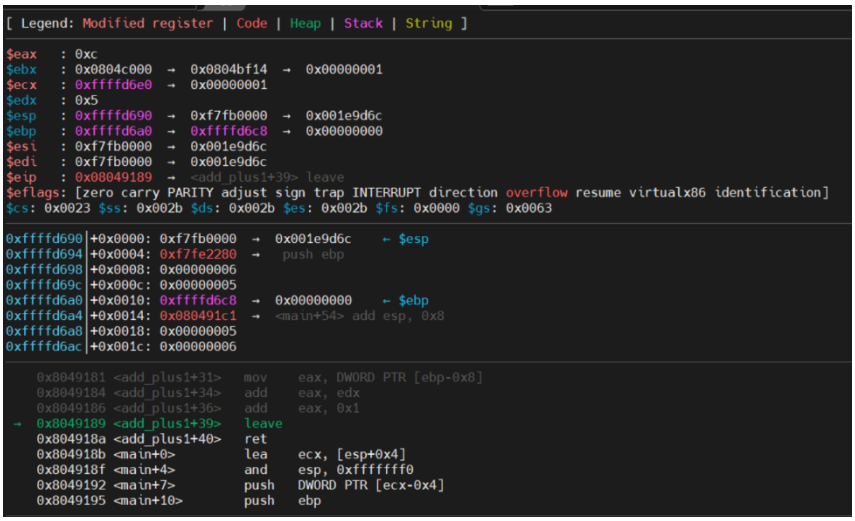
Source Images



Source Image 1



Source Image 2



Source Image 3