Lab 03 Stack Debugging

The purpose of this lab is to demonstrate a solid understanding of how the stack is working and show that you are able to accurately pull information out of memory. It will also serve as an introduction/refresher on working with GDB. The lab should be completed on your Kali machine.

Hint: there is a sheet of helpful GDB commands out on the class GitHub page.

Instructions

1. Get the **lab03.out** binary file onto your Kali VM and make sure it runs. You may need to give it executable permissions first.

2. Open up the binary inside of GDB and print out the list of functions. Set a breakpoint at each of the functions.

Function 1

1. Run the program until you hit the **f1()** function and disassemble it. This function has four local integer variables in it which are initialized at the beginning. We'll call the variable with the smallest offset from EBP "Variable 1" and the variable with the largest offset will be "Variable 4".

2. Step through the initialization of the variables and pay attention to what memory address they are being stored at.

   Hint: You can calculate the memory addresses manually by subtracting the offset from EBP. Alternatively, you can display the bytes using the **x** command at **$ebp-0x*Offset*** (replace *Offset* with the hex number). Doing so will show the address too.

3. Once you know the memory locations of the four local variables and what they are initialized to, go to the **Stack Map** spreadsheet and fill out the table for **f1()**
   a. For the ***address*** column, put the memory address of where the variable lives.
   b. For the ***value*** column, put the decimal number stored there.

4. This function does some math with the variables and returns the resulting number. Step down to the **leave** instruction, but do not execute it yet.

5. Use the **x** command to print out the **1 word** (as in 4 bytes) of **decimal** at the address variable 4 is stored in. This should print out the answer to the math equation.

Function 2

1. Start by viewing the disassembly of function f**2()**. This function uses three local variables and passes them to a third function, **f3()**.

2. Step through the initialization of the variables and pay attention to what memory address they are being stored at.

3. Once you know the memory locations of the four local variables and what they are initialized to, go to the **Stack Map** spreadsheet and fill out the table for **f2()**
    a. For the ***address*** column, put the memory address of where the variable lives.
    b. For the ***value*** column, put the decimal number stored there.

4. Step down to the **call** for **f3()** but do not run it yet. Look right above the call, the three local variables are pushed onto the stack in reverse order – that way they can be popped off correctly. It looks like this:
    - push 1 (Variable 3)
    - push 2 (Variable 2)
    - push 3 (Variable 1)
    - call f3()

5. When you push a variable onto the stack, you're effectively making a copy of it. The value of the variable will be stored at a new memory address on the stack where ESP is pointing to. Before the push, ESP is decremented (moved lower from EBP) by 4 bytes and the value is then stored.

6. Print out the 3 variables that were pushed onto the stack with **x /3xw $esp**

7. On the **Stack Map** spreadsheet, record the memory addresses of where the pushed variables' values are being stored at before the call.

Hint: If you know where ESP is at before the call to f3, you can add 4 bytes to it and get the address of the previous push. Alternatively, step through the function and take note of ESP each time a push is made.

8. Let the program finish running and view what is printed at the end.