Report
Nhc29 Nicholas Chen
Pkk46 Pavan Kumar Kokkiligadda

1.4 Report

Please submit a report that answers the following question. Without the report, you will not receive points for this part.

1. What are the contents in the stack? Feel free to describe your understanding.

The stack contained local variables such as the signalno in our signal_handle function. Another thing that was on the stack was the program counter(eip), the stack pointer(esp), and the frame pointer(ebp), which are obtained with the command info registers eip esp ebp. It also had saved registers, arglist and locals. The contents, like the return addresses, are obtained with the command x/32x $esp, which is then used to find the offset.

2. Where is the program counter, and how did you use GDB to locate the PC?

The program_counter was in $eip, and we utilized the command, layout registers, and info registers after each instruction to find the location and what was inside it. We also utilized the x command, allowing us to examine the stack in more depth to find the offset of the signalno. We also utilized info frame to cross-verify the program counter as well.

3. What were the changes to get the desired result?

Here, we needed to find an offset (described in problem 2) and add the offset by the address of the signal number to find the program counter. Now, we add the size of the instruction to the Program counter, causing it to skip the math that causes a floating point exception, reaching the end goal, the print statement.

To find the size of the bad instruction, we just subtracted the end memory of the bad instruction with the beginning to get the full size in the disassembled code here we did (layout asm) in gdb and also disassemble /m main command is utilized.

2.2 Report

In your report, describe how you implemented the bit operations.

From the Brain Kernighan algorithm (n & (n-1)) flips the rightmost set bit to 0. XOR'ing n^(n&(n-1)) leaves just the rightmost set bit to 1 and the rest to 0. Then, log base 2 of it, giving the position of the first set order bit.

Report

Nhc29 Nicholas Chen
Pkk46 Pavan Kumar Kokkiligadda

To set a bit at a given index, we need to find the bitmap index and offset in that byte, the former obtained by floor division of index with and the latter obtained by modulus with 8. Then bitmap[byte_index] = bitmap[byte_index] | (1<< offset)

To get a bit at index, do the same, obtain the byte_index = index//8 and offset = index %8. Then right shift the bitmap[byte_index] by offset value, and also &1 gives whether the bit is 1 or 0. I.e., return (bitmap[byte_index] >> offset)&1

3.3 Report

You do not have to describe this part in the project report.