Defusing Report - Bomb 24

Ameer Taweel

Introduction

In this report, I will explain how I defused Bomb 24 for the fourth assignment of COMP201. In the beginning, I will explain how I started the process of understanding and navigating around the bomb. Then I will explain how I solved each of the five phases. And finally, I will explain how I found and solved the secret phase.

Defusing Began

- Read and understand `bomb.c`.
- Read and understand the output of: `objdump -t bomb`
- Read and understand the output of: `objdump -d bomb`
- Read and understand the output of: `strings bomb`
- Watch the tutorial about `gdb`, and read more about debugging assembly with it on the internet.

Phase #1

- Read the code of the phase.
- The code is straightforward. I used the string passed to `<strings_not_equal>` as my solution.

Phase #2

- Understanding `<read_six_numbers>` was not easy because of the call to `0x5555555552d0`. Running `disass` did not work there, so I could not read the instructions.
- I found a way to print the instructions at that place:
  `disass beginning_address end_address`
- It turned out that the call to `0x5555555552d0` is a call to `<scanf>`.
- Found out that:
  $a_0 = 0$
  $a_1 = 1$
  $a_i = a_{i-1} + a_{i-2}$
- Extract the remaining four numbers.

Phase #3

- Find the value that makes the conditional jump go to the correct instruction.
- The idea is simple but the execution took some time because I made some errors in my calculations on hex numbers. So I did not get it right from the first try, and spent some time tracking the issue in my calculation.

Phase #4

- Understand the assembly code.
- There is a recursive function, `<func4>`, that I have to track.
- I did not want to track it by hand, because it takes a lot of time and is prone to errors.
- So I wrote a simulator for the registers and stack state that gives me the correct answer.
- The code for the simulator is provided as `func4.cpp`.
- I wrote it in C++ because it has a stack in the standard library, unlike C. And I did not want to reinvent the wheel for this assignment.
- Note: Information on how to compile and run the simulator are provided as a comment at the top of the file.

Phase #5

- Understand the assembly code.
- There is a recursive part in the code.
- Write a simulator for the recursive part to give me the answer, for the same reasons mentioned in phase 4.
- The code for the simulator is provided as `phase_5.cpp`.
- This is a C++ file, but it is technically a valid C code as well.
- Note: Information on how to compile and run the simulator are provided as a comment at the top of the file.

Secret Phase

- Run `objdump -d bomb | grep "phase"`.
- Found two candidate functions to be the secret phase:
    - `<phase_6>`
    - `<secret_phase>`
- `<phase_6>` is not getting called from any other function, so it is a trap.
- `<secret_phase>` is called from `<phase_defused>`.
- It uses the input for phase 4.
- Find the string that causes `<secret_phase>` to be called.
- `<secret_phase>` has a recursive function, `<fun7>`.
- I could not write a simulator for `<fun7>`, because it uses very separated memory blocks that were not very predictable based on the input.
- I had to trace `<fun7>` by hand.
- I started from the end result and traced back to the start of the function.
- I found the correct input.