Mertcan Aşgün

63948

Assignment 4 Report

I started with the command “objdump -t bomb”. Inspecting it a few minutes made me realize the name of the phases. That is also where I realized there is a phase called secret_phase.

The command “objdump -d bomb” is similar to disassemble call in gdb, so I did not use this command very much. It was useful to detect where the secret_phase is called in the program.

Before first phase, I also used the command “strings bomb”. In it, there was a sentence, which I thought would be an answer for a phase. So, I marked it down.

Phase 1: Since I was not familiar with gdb, I exploded the bomb 6 times in total before realizing that adding a break point at explode_bomb function would prevent the bomb’s explosion. For first phase, I disassembled the phase_1 method. In it there was a call to a memory location “0x2a30”. Calling print method as, “print (char*) 0x2a30”, I get:

$1 = 0x2a30 “You can Russia from land here in Alaska.”

And this sentence was exactly the same sentence I marked down previous. So, phase_1 was done.

Phase 2: In phase_2, my first clue was the function call to read_six_numbers. Knowing that the input will be 6 integers, I traced the assembly code and realized that it starts with 0 and it is somehow similar to Fibonacci series. I filled the rest of the integers according to this and I was done.

Phase 3: In phase_3, by printing the call to the memory location, I realized that I needed two integer values. By tracing the code, first input had to smaller than 7. And further below it states that it has to be smaller than 5. Then in a series of addition and subtraction, it gets 15 and compares it with the second argument. Having those clues, I simply started entering inputs like “1 15”, “2 15”, and to my luck, the answer was “1 15”.

Phase 4: In phase_4, again I had two integers as input. Then further below, it calls a helper method called func4. After returned from this function, it compares the returned value with 5. So, it was clear that func4 had to return 5. When I inspected func4 method, I realized that this was a recursive function. Using “info reg” I recorded the values of registers and then find the required pattern that helped me to solve this phase.

Phase 5: I enjoyed the phase_5 a lot. At the start I had no clue about what to do. After tracing the code, I realized that by “and” call, the code simply truncates the bytes of the first input except the least significant 4 bits. So, the first input had to be between 1 and 15. Then by using “info reg” I followed the results. And there was a pattern. 5 lead to 12, 12 lead to 3, 3 lead to 7… At the end of this chain was 15. And the second input was the sum of all those values.

Phase 6: I spent hours on last one. Literally, I had no idea what to do, and the code was also very long for me to trace it line by line. After some time, I discovered a method in gdb and used it on the memory location pointed to node1. When I call: “x/3x 0x555555758230”, I get:

“0x555555758230 <node1>: 0x0000030d 0x00000001 0x55758240”.

When I passed the third memory location to “x/3x”, I realized that this memory was for node2. Following in this fashion, I got 6 nodes’ locations. Then reordering their values, which are located at their first argument, I got the correct answer.

Secret Phase: For the secret_phase, I inspected phase_defused, where the secret_phase method is called. In it, there was a memory location pointed to two digits and one string. Moreover, one other memory location was pointed to the string “DrEvil”. So, I try to add this string to all my answers with 2 digits. Eventually, in one of them I managed to enter into secret_phase. This phase calls a helper function named func7. When I passed some random values and inspecting the values of the register, I realized that there was a pattern. At each step, the input was compared with a certain integer. After a lot of tries I managed to record all those integers and one of them was the answer.