University of Colorado - Boulder

CSPB 2400
Computer Systems | Summer 2024

Lab 1: Bomblab

Sam Walker

Hoang Truong

Monday, June 24, 2024

Part 1

Execution Check (40 points)

GitHub Repo Link: Repo Link [https://github.com/cu-cspb-2400-summer-2024/lab2-bomblab-sawa9885]

Bomb number and a screenshot of leaderboard:

I solved both bomb 3 and bomb 72. I'm not sure why it says invalid phase 2 for bomb 72 but I figured its ok because I also have completely solved bomb 3.

#	Bomb number	Submission date	Phases defused	Explosions	Score	Status
1	bomb6	Tue Jun 25 13:06	7	0	70	valid
2	bomb39	Wed Jun 26 04:44	7	0	70	valid
3	bomb95	Tue Jun 25 00:02	7	1	70	valid
4	bomb10	Wed Jun 26 04:49	7	1	70	valid
5	bomb63	Tue Jun 25 15:51	7	12	64	valid
6	bomb20	Tue Jun 25 10:50	7	25	58	valid
7	bomb4	Mon Jun 24 04:44	6	0	70	valid
8	bomb29	Mon Jun 24 14:31	1	0	10	invalid phase 2
9	bomb34	Mon Jun 24 16:56	6	1	70	valid
10	bomb32	Mon Jun 24 18:17	6	1	70	valid
11	bomb77	Tue Jun 25 21:54	6	1	70	valid
12	bomb5	Wed Jun 19 16:00	6	3	69	valid
13	bomb18	Sun Jun 23 18:44	6	3	69	valid
14	bomb72	Wed Jun 26 18:13	1	3	9	invalid phase 2
15	bomb3	Sat Jun 8 15:14	6	5	68	valid
16	bomb11	Mon Jun 24 20:49	6	5	68	valid
17	bomb22	Mon Jun 24 22:27	6	6	67	valid
18	bomb31	Wed Jun 19 20:15	6	7	67	valid
19	bomb1	Fri Jun 21 14:19	6	7	67	valid
20	bomb54	Mon Jun 24 19:09	6	17	62	valid
21	bomb67	Mon Jun 24 17:57	6	18	61	valid
22	bomb2	Wed Jun 26 18:17	6	64	50	valid
23	bomb24	Wed Jun 19 14:37	5	1	55	valid
24	bomb55	Sun Jun 23 19:23	5	3	54	valid
25	bomb97	Mon Jun 24 21:48	4	0	40	valid
26	bomb13	Sun Jun 23 22:14	4	3	39	valid
27	bomb21	Wed Jun 26 12:16	4	3	39	valid
28	bomb73	Mon Jun 24 20:43	4	10	35	valid
29	bomb17	Mon Jun 24 23:17	4	24	28	valid
30	bomb47	Sat Jun 22 14:19	3	2	29	valid
31	bomb53	Sat Jun 22 18:38	3	2	29	valid
32	bomb61	Sat Jun 22 22:33	3	3	29	valid
33	bomb33	Mon Jun 24 17:12	3	4	28	valid
34	bomb48	Sat Jun 22 15:18	3	5	28	valid
35	bomb62	Tue Jun 25 15:53	3	6	27	valid
36	bomb37	Sat Jun 22 10:16	1	1	10	invalid phase 2
37	bomb45	Sat Jun 22 12:53	2	1	20	valid
38	bomb46	Sat Jun 22 13:03	2	1	20	valid

Fig. 1 Bomb 3 and 72

Part 2-i - Phase 3

General operation of phase 3:

Phase 3 consists of a switch statement depending on the first value and then a simple check following for the second or even possibly third value. In bomb3 I had d c d input and in bomb72, I simply had d d input. The first input indicates where the jump table jumps to and the subsequent numbers are checked via hardcoded values into eax.

Jump table assembly explanation:

A jump table is essentially an array of pointers (addresses) to code blocks. Each entry in the jump table corresponds to a case in the switch statement. In bomb 72 which I have completed more recently the input values range from 0 to 7 which result in different jumps. By running x/10wd 0x5555555571e0 we can begin dissecting the jumps. Performing x/3i previous address + offset we can explore the exact lines each jump goes to. From there the required value for that jump is displayed.

Part 2-ii - Phase 4

General operation of phase 4:

Phase 4 takes 2 numbers, the first number is passed into a recursive function call. The second number is checked via hardcoding after the recursive function check is complete.

func4 explanation (What are initial input parameters? How do those parameters change after each recursion call?):

The initial parameters passed in are rsi, rdi, and rdx. In my case, rdx and rsi are hardcoded values with them being 14 and 0 respectively, however, rdi is the register our first input of phase 4 gets stored to. In each recursive call rdx and rdi remain unchanged, with rsi being calculated as such $\frac{rdx-rsi}{2} + rsi + 1$. Further the new $rsi - 1$ which is simply saved to rcx is compared with rdi. For the correct solution for bomb72 the we need rcx to be less than rdi twice and then equal to on the third time. This is exemplified by an input of 13 which yields rsi of 8->12->14 and rcx of 7->11->13. This follows the characterization of the necessary comparisons by analyzing the parameters of func4 and thus defuses the bomb.

Part 2-iii - Phase 5

General operation of phase 5:

For phase 5, there is an array with 15 different. The value of a specific index is used as the next index. The first input indicates where in the array we start. The second input number is a hardcoded checked value after the for loop.

The main for loop explanation (what happens in this loop):

The array consists of values between 0 and 15 inclusive and the for loop exits when the value of the array is 15. As mentioned previously the for loop starts at the index of the array of the first input number. It then loops until it finds 15 by using the value at the index of the previous value. Basically that means that if arr[5]=9 then the next iteration of the loop will be arr[9]=... The hint mentions that chars could be involved but bomb72 used ints so I can't comment on that. To successfully exit the loop as well as the next condition avoiding the bomb explosion it is necessary to run through the entire array. To solve this we can inspect the array values with (gdb) x/60x $rsi. Which displays all 15 array values because 15*4=60. From there we can build our array and start from the value 15 and work backwards to find the starting value.

Part 2-iv - Phase 6

General operation of phase 6:

Phase 6 takes in 6 numbers 1-6 non repeating and traverses a linked list nodes in that order.

What are the condition checks for the 6 inputs in phase 6:

The condition checks check that there is 6 inputs and not less. Another condition check checks to make sure the inputs aren't greater than 6 or less than 1 i.e. the values must be 1-6. The last check makes sure there aren't 2 repeating numbers so we need to input 6 unique numbers 1-6.

Show the linked list values and corresponding address:

Node 1 address is hardcoded in rdx and from there we can view the 4w expressions which show what I will call the id, the value, and the next node address. I don't know what the fourth column is it seems unused.

```
0x555555559630 <node1>: 0x000002e0      0x00000001      0x55559670      0x00005555
(gdb) x/4xw 0x555555559640
0x555555559640 <node2>: 0x00000179      0x00000002      0x55559650      0x00005555
(gdb) x/4xw 0x555555559650
0x555555559650 <node3>: 0x0000017d      0x00000003      0x55559660      0x00005555
(gdb) x/4xw 0x555555559660
0x555555559660 <node4>: 0x00000329      0x00000004      0x55559630      0x00005555
(gdb) x/4xw 0x555555559670
0x555555559670 <node5>: 0x00000345      0x00000005      0x00000000      0x00000000
(gdb) x/4xw 0x555555559120
0x555555559120 <node6>: 0x00000071      0x00000006      0x55559640      0x00005555
```

Fig. 2 Linked list values and corresponding address.

The condition for solving phase 6 was finding out that each id as I call needs to be greater than the previous one. And because we choose the traversal order we simply order them based on smallest to largest id. For bomb 72, this example, the correct solution is 6 2 3 1 4 5 which lines up with the id numbers since 6 is the smallest and 5 has the largest.

Part 3

Pick 1 of the difficulties/challenges you have dealt with in any of the phases and explain shortly (200 words) your process of debugging/finding the solution:

In phase 6 I often got very confused on why I was jumping all throughout the program and I was struggling to understand the importance and reasoning behind most of the jumps. My initial thought was trying to understand everything line by line, however, that strategy wasn't working well for phase 6. Instead I finally realized it was easier to determine general understandings by placing breakpoints directly after the loops. For example I could explore how registers changed after it did its whole loop behaviour for checking the value and could then realize the bigger picture that it used a loop just for checking and it may not be important to understand every little detail of the entire program. This also made it much faster to debug because I wasn't getting caught up in the loops that run 50 times but rather the ones that run just a few important times. In all of the phases I experienced issues with understanding when to uses x/s vs x/4wx vs x/wx and such. I often felt defeated when it would say stuff like that memory location cannot be accessed, however, I realized throughout the process that my confusion early on was corrected just by doing it over and over throughout the whole assignment. I was focused on learning the theory behind certain operations when the actual application of those operations was much more use case specific and I could only learn that by using them thousands of times.