

## Bomb Lab

```
karthikeya@DESKTOP-6IENM0S:~/ug/sem3/ca/labs/lab5/bomb91$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I turned the moon into something I call a Death Star.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2. Keep going!
1 u 570
Halfway there!
8 35
So you got that one. Try this one.
jebbig
Good work! On to the next...
4 6 2 5 3 1
Congratulations! You've defused the bomb!
```

### Phase 1:

Test input: Hello

From the logic we can see that input string is passed into function `strings_not_equal`  
As we examine the other argument it is also a string.

In the functions `strings_not_equal` both strings are compared.

```
Reading symbols from bomb...
(gdb) break phase_1
Breakpoint 1 at 0x400e8d
(gdb) run
Starting program: /home/karthikeya/ug/sem3/ca/labs/lab5/bomb91/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Hello

Breakpoint 1, 0x000000000400e8d in phase_1 ()
(gdb) disass phase_1
Dump of assembler code for function phase_1:
=> 0x000000000400e8d <+0>:      sub    $0x8,%rsp
    0x000000000400e91 <+4>:      mov     $0x402450,%esi
    0x000000000400e96 <+9>:      callq  0x4013bc <strings_not_equal>
    0x000000000400e9b <+14>:     test   %eax,%eax
    0x000000000400e9d <+16>:     je     0x400ea4 <phase_1+23>
    0x000000000400e9f <+18>:     callq  0x4014bb <explode_bomb>
    0x000000000400ea4 <+23>:     add     $0x8,%rsp
    0x000000000400ea8 <+27>:     retq

End of assembler dump.
(gdb) break strings_not_equal
Breakpoint 2 at 0x4013bc
(gdb) cont
Continuing.

Breakpoint 2, 0x0000000004013bc in strings_not_equal ()
(gdb) x/s $rdi
0x6047a0 <input_strings>:      "Hello"
(gdb) x/s $rsi
0x402450:      "I turned the moon into something I call a Death Star."
```

**Answer:** I turned the moon into something I call a Death Star.

## Phase 2:

From the logic:

The programs reads 6 integers using function `read_six_numbers`

The first 2 numbers must be 0, 1 we know by compare instructions at <+30> and <+36>

Next logic is a for loop which iterate through the next 4 integers

From the logic we come to know that present number is sum of previous two numbers

0

1

0+1 = 1

1+1 = 2

2+1 = 3

3+2 = 5

```
Breakpoint 1, 0x000000000400ea9 in phase_2 ()
(gdb) disass phase_2
Dump of assembler code for function phase_2:
=> 0x000000000400ea9 <+0>:      push    %rbp
    0x000000000400eaa <+1>:      push    %rbx
    0x000000000400eab <+2>:      sub     $0x28,%rsp
    0x000000000400eaf <+6>:      mov     %fs:0x28,%rax
    0x000000000400eb8 <+15>:     mov     %rax,0x18(%rsp)
    0x000000000400ebd <+20>:     xor     %eax,%eax
    0x000000000400ebf <+22>:     mov     %rsp,%rsi
    0x000000000400ec2 <+25>:     callq   0x4014dd <read_six_numbers>
    0x000000000400ec7 <+30>:     cmpl    $0x0,(%rsp)
    0x000000000400ecb <+34>:     jne     0x400ed4 <phase_2+43>
    0x000000000400ecd <+36>:     cmpl    $0x1,0x4(%rsp)
    0x000000000400ed2 <+41>:     je      0x400ed9 <phase_2+48>
    0x000000000400ed4 <+43>:     callq   0x4014bb <explode_bomb>
    0x000000000400ed9 <+48>:     mov     %rsp,%rbx
    0x000000000400edc <+51>:     lea     0x10(%rsp),%rbp
    0x000000000400ee1 <+56>:     mov     0x4(%rbx),%eax
    0x000000000400ee4 <+59>:     add     (%rbx),%eax
    0x000000000400ee6 <+61>:     cmp     %eax,0x8(%rbx)
    0x000000000400ee9 <+64>:     je      0x400ef0 <phase_2+71>
    0x000000000400eeb <+66>:     callq   0x4014bb <explode_bomb>
    0x000000000400ef0 <+71>:     add     $0x4,%rbx
--Type <RET> for more, q to quit, c to continue without paging--
    0x000000000400ef4 <+75>:     cmp     %rbp,%rbx
    0x000000000400ef7 <+78>:     jne     0x400ee1 <phase_2+56>
    0x000000000400ef9 <+80>:     mov     0x18(%rsp),%rax
    0x000000000400efe <+85>:     xor     %fs:0x28,%rax
    0x000000000400f07 <+94>:     je      0x400f0e <phase_2+101>
    0x000000000400f09 <+96>:     callq   0x400b00 <__stack_chk_fail@plt>
    0x000000000400f0e <+101>:    add     $0x28,%rsp
    0x000000000400f12 <+105>:    pop     %rbx
    0x000000000400f13 <+106>:    pop     %rbp
    0x000000000400f14 <+107>:    retq
End of assembler dump.
```

**Answer: 0 1 1 2 3 5**

**Phase 3:**

```

Dump of assembler code for function phase_3:
=> 0x0000000000400f15 <+0>:      sub    $0x28,%rsp
      0x0000000000400f19 <+4>:      mov    %fs:0x28,%rax
      0x0000000000400f22 <+13>:     mov    %rax,0x18(%rsp)
      0x0000000000400f27 <+18>:     xor    %eax,%eax
      0x0000000000400f29 <+20>:     lea    0x14(%rsp),%r8
      0x0000000000400f2e <+25>:     lea    0xf(%rsp),%rcx
      0x0000000000400f33 <+30>:     lea    0x10(%rsp),%rdx
      0x0000000000400f38 <+35>:     mov    $0x4024ae,%esi
      0x0000000000400f3d <+40>:     callq 0x400bb0 <__isoc99_sscanf@plt>

(gdb) x/s 0x4024ae
0x4024ae:      "%d %c %d"

```

From screen shot we can see that it is reading in put in the form of "%d %c %d"

```

0x0000000000400f7f <+106>:    jmpq   0x40105a <phase_3+325>
0x0000000000400f84 <+111>:    mov    $0x75,%eax
0x0000000000400f89 <+116>:    cmpl   $0x23a,0x14(%rsp)
0x0000000000400f91 <+124>:    je     0x40105a <phase_3+325>
0x0000000000400f97 <+130>:    callq  0x4014bb <explode_bomb>
0x0000000000400f9c <+135>:    mov    $0x75,%eax

```

We can see in the above screenshot that it is comparing 2nd %d with 0x23a(570)

```

      0x000000000040105a <+325>:    cmp    0xf(%rsp),%al
      0x000000000040105e <+329>:    je     0x401065 <phase_3+336>
=> 0x0000000000401060 <+331>:    callq  0x4014bb <explode_bomb>
      0x0000000000401065 <+336>:    mov    0x18(%rsp),%rax
      0x000000000040106a <+341>:    xor    %fs:0x28,%rax
      0x0000000000401073 <+350>:    je     0x40107a <phase_3+357>
      0x0000000000401075 <+352>:    callq  0x400b00 <__stack_chk_fail@plt>
      0x000000000040107a <+357>:    add    $0x28,%rsp
      0x000000000040107e <+361>:    retq

End of assembler dump.
(gdb) x/c $rsp+0xf
0x7fffffffdefeff: 97 'a'
(gdb) i r
rax                0x75                117

```

From the above screenshot we can see it is comparing our test input character 'a' with 'u' 117.

**Answer: 1 u 570**

**Phase 4:**

```

0x000000000040110b <+89>:    mov     0x8(%rsp),%rax
0x0000000000401110 <+94>:    xor     %fs:0x28,%rax
0x0000000000401119 <+103>:   je      0x401120 <phase_4+110>
0x000000000040111b <+105>:   callq   0x400b00 <__stack_chk_fail@plt>
0x0000000000401120 <+110>:   add     $0x18,%rsp
0x0000000000401124 <+114>:   retq

End of assembler dump.
(gdb) x/s 0x40264f
0x40264f:      "%d %d"

```

From the screenshot, we can see that our input is two integers

```

Dump of assembler code for function phase_4:
0x00000000004010b2 <+0>:    sub     $0x18,%rsp
0x00000000004010b6 <+4>:    mov     %fs:0x28,%rax
0x00000000004010bf <+13>:   mov     %rax,0x8(%rsp)
0x00000000004010c4 <+18>:   xor     %eax,%eax
0x00000000004010c6 <+20>:   lea     0x4(%rsp),%rcx
0x00000000004010cb <+25>:   mov     %rsp,%rdx
0x00000000004010ce <+28>:   mov     $0x40264f,%esi
0x00000000004010d3 <+33>:   callq   0x400bb0 <__isoc99_sscanf@plt>
0x00000000004010d8 <+38>:   cmp     $0x2,%eax
0x00000000004010db <+41>:   jne     0x4010e3 <phase_4+49>
=> 0x00000000004010dd <+43>:   cmpl    $0xe,(%rsp)
0x00000000004010e1 <+47>:   jbe     0x4010e8 <phase_4+54>
0x00000000004010e3 <+49>:   callq   0x4014bb <explode_bomb>
0x00000000004010e8 <+54>:   mov     $0xe,%edx

```

From the screenshot, we can see that our first input must be less than 0xe (14). Based on our first input func4 returns a specific value, which must be equal to 0x23 (35) as you can see in the below screenshot.

```

0x00000000004010f5 <+67>:   callq   0x40107f <func4>
0x00000000004010fa <+72>:   cmp     $0x23,%eax
0x00000000004010fd <+75>:   jne     0x401106 <phase_4+84>
0x00000000004010ff <+77>:   cmpl    $0x23,0x4(%rsp)
0x0000000000401104 <+82>:   je      0x40110b <phase_4+89>
--Type <RET> for more, q to quit, c to continue without paging--
0x0000000000401106 <+84>:   callq   0x4014bb <explode_bomb>

```

From the logic of func4 we can see it is 8.

**Answer: 8 35**

**Phase 5:**

As I examine the logic of phase\_5, it is clear that we need a string of 6 characters

```
0x0000000000401125 <+0>:    push    %rbx
0x0000000000401126 <+1>:    mov     %rdi,%rbx
0x0000000000401129 <+4>:    callq   0x40139e <string_length>
0x000000000040112e <+9>:    cmp     $0x6,%eax
0x0000000000401131 <+12>:   je      0x401138 <phase_5+19>
0x0000000000401133 <+14>:   callq   0x4014bb <explode_bomb>
0x0000000000401138 <+19>:   mov     %rbx,%rax
```

And every character has a specific value and we add the values of each character of our string. Finally the total must be equal to 0x34 (52)

I had found the values of some characters by giving inputs such as a b c d e f and g h i j k l.

```
0x0000000000401151 <+44>:    add     $0x1,%rax
0x0000000000401155 <+48>:    cmp     %rdi,%rax
0x0000000000401158 <+51>:    jne     0x401144 <phase_5+31>
0x000000000040115a <+53>:    cmp     $0x34,%ecx
0x000000000040115d <+56>:    je      0x401164 <phase_5+63>
0x000000000040115f <+58>:    callq   0x4014bb <explode_bomb>
0x0000000000401164 <+63>:    pop     %rbx
0x0000000000401165 <+64>:    retq
```

And finally made a string which defuses phase\_5 “jebbig”, there can be multiple strings that can defuse phase\_5.

**Answer: jebbig**

**Phase 6:**



From the screenshot below we can say that it reads 6 numbers as in phase\_2 otherwise the bomb explodes.

And from the compare instruction it is clear the numbers must be less than 6.

```
Dump of assembler code for function phase_6:
0x000000000401166 <+0>:    push    %r14
0x000000000401168 <+2>:    push    %r13
0x00000000040116a <+4>:    push    %r12
0x00000000040116c <+6>:    push    %rbp
0x00000000040116d <+7>:    push    %rbx
0x00000000040116e <+8>:    sub     $0x60,%rsp
0x000000000401172 <+12>:   mov     %fs:0x28,%rax
0x00000000040117b <+21>:   mov     %rax,0x58(%rsp)
0x000000000401180 <+26>:   xor     %eax,%eax
0x000000000401182 <+28>:   mov     %rsp,%rsi
0x000000000401185 <+31>:   callq   0x4014dd <read_six_numbers>
0x00000000040118a <+36>:   mov     %rsp,%r12
0x00000000040118d <+39>:   mov     %rsp,%r13
0x000000000401190 <+42>:   mov     $0x0,%r14d
0x000000000401196 <+48>:   mov     %r13,%rbp
0x000000000401199 <+51>:   mov     0x0(%r13),%eax
0x00000000040119d <+55>:   sub     $0x1,%eax
0x0000000004011a0 <+58>:   cmp     $0x5,%eax
0x0000000004011a3 <+61>:   jbe     0x4011aa <phase_6+68>
0x0000000004011a5 <+63>:   callq   0x4014bb <explode_bomb>
```

From the iterative part I found that the numbers should not repeat.

At 0x6042f0 there is a linked list of 6 integers

```
(gdb) x/100x 0x6042f0
0x6042f0 <node1>:    0xee  0x02  0x00  0x00  0x01  0x00  0x00  0x00
0x6042f8 <node1+8>:    0x00  0x43  0x60  0x00  0x00  0x00  0x00  0x00
0x604300 <node2>:    0x2f  0x02  0x00  0x00  0x02  0x00  0x00  0x00
0x604308 <node2+8>:    0x10  0x43  0x60  0x00  0x00  0x00  0x00  0x00
0x604310 <node3>:    0x38  0x03  0x00  0x00  0x03  0x00  0x00  0x00
0x604318 <node3+8>:    0x20  0x43  0x60  0x00  0x00  0x00  0x00  0x00
0x604320 <node4>:    0x4d  0x00  0x00  0x00  0x04  0x00  0x00  0x00
0x604328 <node4+8>:    0x30  0x43  0x60  0x00  0x00  0x00  0x00  0x00
0x604330 <node5>:    0x9a  0x02  0x00  0x00  0x05  0x00  0x00  0x00
0x604338 <node5+8>:    0x40  0x43  0x60  0x00  0x00  0x00  0x00  0x00
0x604340 <node6>:    0x46  0x00  0x00  0x00  0x06  0x00  0x00  0x00
0x604348 <node6+8>:    0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x604350 <bomb_id>:    0x5b  0x00  0x00  0x00
```

We can see the integers in the little endian format in the screenshot.

Node 1 - 0x02ee

Node 2 - 0x022f

Node 3 - 0x0338

Node 4 - 0x004d

Node 5 - 0x029a

Node 6 - 0x0046

Descending order of nodes by data in them 3 1 5 2 4 6

```

401254: 00
401255: bd 05 00 00 00      mov     $0x5 %ebp
40125a: 48 8b 43 08          mov     0x8(%rbx) %rax
40125e: 8b 00                mov     (%rax) %eax
401260: 39 03                cmp     %eax (%rbx)
401262: 7d 05                jge     401269 <phase_6+0x103>
401264: e8 52 02 00 00      callq   4014bb <explode_bomb>
401269: 48 8b 5b 08          mov     0x8(%rbx) %rbx

```

From the above screenshot, the cmp instruction indicates that (%rbx) must be greater than the value in rax, otherwise the bomb explodes.

I had put a breakpoint at the above compare instruction and repeatedly observed the values in (%rbx) and \$rax using different test inputs and drew patterns between them. (for example if our first input is 4 the the first value in (%rbx) is the value in index 3)

It is clear that the values in memory (%rbx) and its corresponding values must be in descending order of the values in the nodes.

Our input arranges the indices in it. So that

For

4      6      2      5      3      1

Corresponding nodes in rax and %(rbx) for each iteration  
(They actually store the data present in the above nodes)

\$rax

1      5      2      4      6

%(rbx)

3      1      5      2      4

For every iteration value in %(rbx) greater than \$rax

**Answer: 4 6 2 5 3 1**

```

karthikeya@DESKTOP-6IENM0S:~/ug/sem3/ca/labs/lab5/bomb-lab$ ./bomb < Solution.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2. Keep going!
Halfway there!
So you got that one. Try this one.
Good work! On to the next...
Congratulations! You've defused the bomb!

```