BOMBLAB28

By: JUNHAN CHEN (19307130180)

phase_1

1.answer: For NASA, space is still a high priority.

2.analyse the asm file:

- `0x00000000000015ab <+4>: sub $0x8,%rsp`:
  allocate memory in the stack to store the string I put in
- `0x00000000000015af <+8>: lea 0x1b9a(%rip),%rsi`:
  load address "$rip+0x1b9a" to %rsi, which contains a string, let's call it s
- `0x00000000000015b6 <+15>: callq 0x1a97 <strings_not_equal>`:
  compare s with input string
- `0x00000000000015bb <+20>: test %eax,%eax`:
- `0x00000000000015bd <+22>: jne 0x15c4 <phase_1+29>`
- `0x00000000000015c4 <+29>: callq 0x1c9e <explode_bomb>`:
  if not equal, bomb will explode

3.By roughly analysing the process, we could assume that s is the answer.Now use GDB debugger to find the content of %rsi:

```
(gdb) b phase_1
Breakpoint 1 at 0x15a7
(gdb) run
Starting program: /home/brayton/bomb28/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
hello
Breakpoint 1, 0x00005555555555a7 in phase_1 ()
(gdb) stepi
0x00005555555555ab in phase_1 ()
(gdb) stepi
0x00005555555555af in phase_1 ()
(gdb) stepi
0x00005555555555b6 in phase_1 ()
(gdb) x/s $rip+0x1b9a
0x555555557150:    "For NASA, space is still a high priority."
```

phase_2

1.ans: 0 1 1 2 3 5

2.analyse the asm file:

- `callq 0x555555555cca <read_six_numbers>`:
  before this step, system allocate memory of six integers, then call the function "read_six_numbers" to store what i put in
- `0x00005555555555ed <+34>: cmpl $0x0,(%rsp)`:
  %rsp points to the first integer, compare it with zero
- `0x00005555555555f1 <+38>: jne 0x5555555555fa <phase_2+47>`:
  if equal, just continue, otherwise, bomb will explode, so the first element should be 0

- `0x00005555555555f3 <+40>: cmpl $0x1,0x4(%rsp)`:
  %rsp+0x4 points to the second integer for every integer takes up 4 bytes. then compare it with 1
- `0x00005555555555f8 <+45>: je 0x5555555555ff <phase_2+52>`
- `0x00005555555555fa <+47>: callq 0x555555555c9e <explode_bomb>`:
  if equal, jump to phase_2+52, otherwise the bomb will explode, so the second element should be 1
- `0x00005555555555ff <+52>: mov %rsp,%rbx`:
  use %rbx to store the pointer in %rsp
- `0x0000555555555602 <+55>: lea 0x10(%rsp),%rbp`:
  use %rbp to store the address "%rsp+16", which stores the fifth element
- `0x0000555555555607 <+60>: jmp 0x555555555617 <phase_2+76>`
- below is a circulation and begin at <+76>:
- `0x000055555555560e <+67>: add $0x4,%rbx`:
  renew the address stored in %rbx with adding 0x4, which aim at pointing to the next element
- `0x0000555555555612 <+71>: cmp %rbp,%rbx`:
  compare the value in %rbp and %rbx
- `0x0000555555555615 <+74>: je 0x555555555623 <phase_2+88>`:
  if equal, quit the circulation, otherwise continue the next step
- `0x0000555555555617 <+76>: mov 0x4(%rbx),%eax`:
  move the value in %rbx+0x4 to %eax
- `0x000055555555561a <+79>: add (%rbx),%eax`:
  add the value in %rbx to %eax
- `0x000055555555561c <+81>: cmp %eax,0x8(%rbx)`:
  compare the value in %eax with what's in %rbx+0x8
- `0x000055555555561f <+84>: je 0x55555555560e <phase_2+67>`
- `0x0000555555555621 <+86>: jmp 0x555555555609 <phase_2+62>`:
  if equal, continue the circulation, otherwise, the bomb will explode

3.Let's take a look at the circulation. %rbx first refers to the first element, %rbx+0x4 refers to the second element, %rbx+0x8 refers to the third element, besides, in every round the %rbx will move to the next element. In line <+81>, it compares the sum of the first and second element with the third element. Similarly, in second round of circulation, it compares the sum of the second and third element with the fourth element, untill %rbx+0x8 refers to the last element. Finally, we find that we should input 6 numbers, the first is 0, the second is 1, and the rest element is addition of two previous elements.

---

phase_3

1.ans(one of): 4 0

2.analyse the asm file:

- `0x000000000000165f <+32>: lea 0x1ce6(%rip),%rsi`:
  %rip+0x1ce6 stores the string which tells you what you should input
- `0x0000000000001666 <+39>: callq 0x12c0 <__isoc99_sscanf@plt>`:
  store what I input
- `0x0000000000001670 <+49>: cmpl $0x7,(%rsp)`
- `0x0000000000001674 <+53>: ja 0x1714 <phase_3+213>`:
  if the first element I input is bigger than 7, then the bomb will explode; otherwise continue
- `0x000000000000167a <+59>: mov (%rsp),%eax`:
  load the first element to %eax
- `0x000000000000167d <+62>: lea 0x1b3c(%rip),%rdx`:
  load the value in %rip+0x1b3c to %rdx
- `0x0000000000001684 <+69>: movslq (%rdx,%rax,4),%rax`
- `0x0000000000001688 <+73>: add %rdx,%rax`:
  deal with the initial value

- `0x000000000000168b <+76>: notrack jmpq *%rax` :
  according to the first element, the pointer will lead to defferent command row
- `0x000000000000168e <+79>: callq 0x1c9e <explode_bomb>` :
  if no such row, bomb explode
- `0x000000000000169a <+91>: sub $0x2a1,%eax` :
  %eax -= 673
- `0x000000000000169f <+96>: add $0x3d4,%eax` :
  %eax += 980
- `0x00000000000016a4 <+101>: sub $0x223,%eax` :
  %eax -= 547
- `0x00000000000016a9 <+106>: add $0x223,%eax` :
  %eax += 547
- `0x00000000000016ae <+111>: sub $0x223,%eax` :
  %eax -= 547
- `0x00000000000016b3 <+116>: add $0x223,%eax` :
  %eax += 547
- `0x00000000000016b8 <+121>: sub $0x223,%eax` :
  %eax -= 547
- `0x00000000000016bd <+126>: cmpl $0x5,(%rsp)` :
  compare the first element with 5
- `0x00000000000016c1 <+130>: jg 0x16c9 <phase_3+138>`
  if first element is greater than 5, the bomb will explode; otherwise continue
- `0x00000000000016c3 <+132>: cmp %eax,0x4(%rsp)` :
  compare the second element with the value in %eax
- `0x00000000000016c7 <+136>: je 0x16ce <phase_3+143>` :
- `0x00000000000016c9 <+138>: callq 0x1c9e <explode_bomb>` :
  if equal, the answer is right, then quit the process; otherwise the bomb explode
- `0x00000000000016e3 <+164>: mov $0x0,%eax`
- `0x00000000000016e8 <+169>: jmp 0x169a <phase_3+91>` :
  case 1
- `0x00000000000016ea <+171>: mov $0x0,%eax`
- `0x00000000000016ef <+176>: jmp 0x169f <phase_3+96>` :
  case 2
- `0x00000000000016f1 <+178>: mov $0x0,%eax`
- `0x00000000000016f6 <+183>: jmp 0x16a4 <phase_3+101>` :
  case 3
- `0x00000000000016f8 <+185>: mov $0x0,%eax`
- `0x00000000000016fd <+190>: jmp 0x16a9 <phase_3+106>` :
  case 4
- `0x00000000000016ff <+192>: mov $0x0,%eax`
- `0x0000000000001704 <+197>: jmp 0x16ae <phase_3+111>` :
  case 5
- `0x0000000000001706 <+199>: mov $0x0,%eax`
- `0x000000000000170b <+204>: jmp 0x16b3 <phase_3+116>` :
  case 6
- `0x000000000000170d <+206>: mov $0x0,%eax`
- `0x0000000000001712 <+211>: jmp 0x16b8 <phase_3+121>` :
  case 7

3.Through our analyse, we find that this function is a switch, we could choose whatever in {1,2,3,4,5} to be the first element. For convenience, I choose 4 because the process is easy, 0+547-547+547-547=0, so one of the answer is 4 0; also, (3 -547) is right too.

```
4 0
Breakpoint 1, 0x000055555555563f in phase_3 ()
(gdb) x/s $rip+0x1ce6
0x55555555734c:    "%d %d"
```

```
(gdb) x/d $rsp
0x7fffffffdf60:    4
(gdb) x/d $rdx
0x5555555571c0:    -43
(gdb) x/d $rax
0x5555555556f8 <phase_3+185>:    -72
(gdb) x/d $rsp+0x4
0x7fffffffdf64:    0
(gdb) stepi
main (argc=<optimized out>, argv=<optimized out>) at bomb.c:90
90        phase_defused();
```

phase_4

1.ans(one of): 352 4

2.analyse the asm file:

- `0x0000000000001780 <+32>: lea 0x1bc5(%rip),%rsi`:
  %rip+0x1ce6 stores the string which tells you what you should input
- `0x0000000000001787 <+39>: callq 0x12c0 <__isoc99_sscanf@plt>`:
  store what I input
- `0x000000000000178c <+44>: cmp $0x2,%eax`:
- `0x000000000000178f <+47>: jne 0x179c <phase_4+60>`:
  a test for stack overflow attack, just ignore it
- `0x0000000000001791 <+49>: mov (%rsp),%eax`:
  move the second element to %eax
- `0x0000000000001794 <+52>: sub $0x2,%eax`:
  %eax -= 2
- `0x0000000000001797 <+55>: cmp $0x2,%eax`:
  compare 2 with the value in %eax
- `0x000000000000179a <+58>: jbe 0x17a1 <phase_4+65>`
- `0x000000000000179c <+60>: callq 0x1c9e <explode_bomb>`:
  if %eax <= 2, continue; otherwise the bomb explode
- `0x00000000000017a1 <+65>: mov (%rsp),%esi`:
  store the second element in %esi
- `0x00000000000017a4 <+68>: mov $0x9,%edi`
  store 9 in %edi
- `0x00000000000017a9 <+73>: callq 0x1725 <func4>`:
  call func4
- `0x00000000000017ae <+78>: cmp %eax,0x4(%rsp)`
- `0x00000000000017b2 <+82>: jne 0x17c9 <phase_4+105>`:
  if the first element is not equal to the value in %eax, bomb will explode.Otherwise the answer is right, then the process will end

3.Through the help of GDB and my effort to find the regulation, I finally figure out what does func4 do:
It has two initial values: 0 and the second element I input; then it works like this: add the two former value then add the second element I input, which is the third value. Continue this progress for 8 times, so the values will be: 0 4 8 16 28 48 80 132 216 352. Thus, one of the answer is 352 4.

```
2 9
(gdb) x/s $rsi
0x55555555734c:    "%d %d"
(gdb) x/d $rsp
0x7fffffffdf60:    9
(gdb) x/d $rsp+0x4
0x7fffffffdf64:    2
(In normal situation, %rsp stores the first element I input, but in this function, it reverse the order, I still don't
```

phase_5

1.ans: qrqrqr

2.analyse the asm file:

- `0x00005555555557d9 <+4>: push %rbx` :
  push %rbx into stack
- `0x00005555555557da <+5>: mov %rdi,%rbx` :
  move the string I input to %rbx
- `0x00005555555557dd <+8>: callq 0x555555555a76 <string_length>` :
  calculate the length of the string
- `0x00005555555557e2 <+13>: cmp $0x6,%eax`
- `0x00005555555557e5 <+16>: jne 0x555555555813 <phase_5+62>` :
  if the length is 6, continue; otherwise the bomb will explode
- `0x00005555555557e7 <+18>: mov %rbx,%rax` :
  move the string to %rax
- `0x00005555555557ea <+21>: lea 0x6(%rbx),%rdi` :
  move the value in %rbx+0x6 to %rdi
- `0x00005555555557ee <+25>: mov $0x0,%ecx` :
  %ecx = 0
- `0x00005555555557f3 <+30>: lea 0x19e6(%rip),%rsi` :
  set the value of %rsi with $rip+0x19e6
- `0x00005555555557fa <+37>: movzbl (%rax),%edx` :
  move the lowest 4 bits of the first char into %edx
- `0x00005555555557fd <+40>: and $0xf,%edx` :
  %edx &= 0xf
- `0x0000555555555800 <+43>: add (%rsi,%rdx,4),%ecx` :
  %ecx += %rsi+4*%rdx
- `0x0000555555555803 <+46>: add $0x1,%rax` :
  remove the first char from the string
- `0x0000555555555807 <+50>: cmp %rdi,%rax` :
  compare the string in %rdi with those in %rax
- `0x000055555555580a <+53>: jne 0x5555555557fa <phase_5+37>` :
  if not equal, jump to
- `0x000055555555580c <+55>: cmp $0x30,%ecx` :
  compare the value in %ecx with 0x30
- `0x000055555555580f <+58>: jne 0x55555555581a <phase_5+69>` :
  if not equal, the bomb explodes.Otherwise, the answer is right.

3.Through debugging, I find that this process just transport each char in the string into a certain number then add them up, if the result is 48, that's a right answer.So I first input "rrrrrr", then find that r refers to number 6, then I try to find a char which refers to 10. It turns out to be char 'q', so "qrqrqr" will be one of the right answers.

```
rrrrrr
(gdb) x/s $rax
0x555555559800 <input_strings+320>:    "rrrrrr"
(gdb) x/s $rbx
0x555555559800 <input_strings+320>:    "rrrrrr"
(gdb) x/s $rdi
0x555555559806 <input_strings+326>:    ""
(gdb) x/d $rip+0x19e6
0x5555555571e0 <array.3471>:    2
(gdb) x/d $rax
0x555555559800 <input_strings+320>:    114
```

```
(gdb) x/d $ecx
0x6:    Cannot access memory at address 0x6
```

---

phase_6

1.ans: 3 6 4 5 1 2

2.analyse the asm file: (due to the length of the file, I decide not to copy it here)

3.The function works like this:

- input 6 numbers, the range is [1,6]
- all the numbers should be different, so I should arrange the six numbers(1,2,3,4,5,6) in a certain order.
- an array has been set in the file: : 164, :127, :900, :742, :228, :897
- These nodes should be listed in decreasing order. So the answer is:3 6 4 5 1 2.

```
4 3 2 1 6 5
......
(gdb) x/d $rdx
0x555555559260 <node4>:    742
......
(gdb) x/d $rdx
0x555555559250 <node3>:    900
......
(gdb) x/d $rdx
0x555555559240 <node2>:    127
......
(gdb) x/d $rdx
0x555555559230 <node1>:    164
......
(gdb) x/d $rdx
0x555555559110 <node6>:    897
......
(gdb) x/d $rdx
0x555555559270 <node5>:    228
```