

计算机系统基础

Lab2 Bomb Lab

姓名：傅文杰

学号:22300240028

2023 年 10 月 20 日

目录

1	实验准备	2
1.1	获取Bomb	2
1.2	实验目的	2
1.3	实验工具	2
2	实验过程	3
2.1	观察反汇编文件bomb.s以及c语言文件bomb.c	3
2.2	phase_1	4
2.3	phase_2	6
2.4	phase_3	10
2.5	phase_4	12
2.6	phase_5	17
2.7	phase_6	19
2.8	secret_phase	22
2.9	答案ans.txt	24

1 实验准备

1.1 获取Bomb

- 通过<http://autolab.sur.moe:5550/22300240028>获取Bomb:22300240028.out，它是二进制格式的代码，无法直接查看
- 将其与bomb.c放入一个文件夹中便于查看
- 在linux终端中执行./22300240028.out会看到以下文字，如果输入的答案不对，则Bomb会爆炸：

```
fwj@DESKTOP-H0129QB:/mnt/d/桌面/ics作业/Lab2/Lab2$ ./22300240028.out
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
hello

BOOM!!!
The bomb has blown up.
fwj@DESKTOP-H0129QB:/mnt/d/桌面/ics作业/Lab2/Lab2$
```

1.2 实验目的

- 这个Bomb一共有6个phase和1个secret phase需要defuse
- 需要将答案字符串写在.txt文件中并在最后一行写上学号
- 因此，我们在终端vim ans.txt创建答案文件并写入，通过./22300240028.out ans.txt并观察文字提示来判断是否成功defuse Bomb

1.3 实验工具

- gdb
具体方法可以参照[gdb调试利器](#)或者在终端中执行man gdb，gdb - help等命令查看

- objdump

在终端中执行：objdump 22300240028.out > bomb.s 通过管道将反汇编文件保存到bomb.s中

在终端中执行：objdump 22300240028.out > bomb.t 可以得到符号表，可能会有些帮助

2 实验过程

2.1 观察反汇编文件bomb.s以及c语言文件bomb.c

- 观察bomb.c文件，我们可以发现其main函数通过read_line函数读取输入，通过phase_i($1 \leq i \leq 6$)函数判断是否会使炸弹爆炸
- 观察bomb.s文件，我们可以发现read_line函数读入的值将会被存放在寄存器%rdi中，再调用相对应的函数。这是因为%rax通常作为存储函数返回值的寄存器，%rdi通常作为存储函数第一个入参的寄存器。phase_i函数使用%rdi寄存器作为输入。

299	400e32: e8 67 06 00 00	callq 40149e <read_line>
300	400e37: 48 89 c7	mov %rax,%rdi
301	400e3a: e8 a1 00 00 00	callq 400ee0 <phase_1>
302	400e3f: e8 80 07 00 00	callq 4015c4 <phase_defused>
303	400e44: bf a8 23 40 00	mov \$0x4023a8,%edi
304	400e49: e8 c2 fc ff ff	callq 400b10 <puts@plt>
305	400e4e: e8 4b 06 00 00	callq 40149e <read_line>
306	400e53: 48 89 c7	mov %rax,%rdi
307	400e56: e8 a1 00 00 00	callq 400efc <phase_2>
308	400e5b: e8 64 07 00 00	callq 4015c4 <phase_defused>
309	400e60: bf ed 22 40 00	mov \$0x4022ed,%edi
310	400e65: e8 a6 fc ff ff	callq 400b10 <puts@plt>
311	400e6a: e8 2f 06 00 00	callq 40149e <read_line>
312	400e6f: 48 89 c7	mov %rax,%rdi
313	400e72: e8 cc 00 00 00	callq 400f43 <phase_3>
314	400e77: e8 48 07 00 00	callq 4015c4 <phase_defused>

- 我们可以在bomb.s汇编代码中通过Ctrl+F搜索phase_i来观察相应的函数

2.2 phase_1

```
346 00000000400ee0 <phase_1>:  
347 400ee0: 48 83 ec 08      sub    $0x8,%rsp  
348 400ee4: be 00 24 40 00    mov    $0x402400,%esi  
349 400ee9: e8 4a 04 00 00    callq 401338 <strings_not_equal>  
350 400eee: 85 c0             test   %eax,%eax  
351 400ef0: 74 05            je     400ef7 <phase_1+0x17>  
352 400ef2: e8 43 05 00 00    callq 40143a <explode_bomb>  
353 400ef7: 48 83 c4 08      add    $0x8,%rsp  
354 400efb: c3              retq  
355
```

- 第347行表示将栈顶指针 $\text{--}8$ ，为函数分配栈帧
- 第348行表示将寄存器 \%rsi 的后32位，即寄存器 \%esi 赋成立即数 $0x402400$
- 第349行调用名为`strings_not_equal`的函数，猜想寄存器 \%esi 应该存储了该函数的入参，该函数可能是判断字符串是否相等的函数
- 第350行用`test`指令检查寄存器 \%eax 是负数，0，还是正数
- 第351行判断如果零标志ZF是0则跳转到400ef7所在位置，发现刚好跳过了`explode_bomb`函数
- 一般来说，寄存器 \%rax 会存储函数的返回值，由此我们可以猜想 \%eax 存储了字符串比较函数的返回值，如果相等则返回0跳过爆炸，不相等则返回非0无法跳转
- 为了验证我们的猜想，我们需要查看`strings_not_equal`的函数
- 第705，706行表明了传入参数有 \%rdi 和 \%rsi ，其中 \%rsi 的后32位存储了立即数 $0x402400$ 。这两行将这两个参数存储到 \%rbx 和 \%rbp 这两个callee-free寄存器中。接着调用了`string_length`函数，猜测可能是求字符串长度的函数。

```

701 0000000000401338 <strings_not_equal>:
702 401338: 41 54          push    %r12
703 40133a: 55            push    %rbp
704 40133b: 53            push    %rbx
705 40133c: 48 89 fb      mov     %rdi,%rbx
706 40133f: 48 89 f5      mov     %rsi,%rbp
707 401342: e8 d4 ff ff ff callq   40131b <string_length>
708 401347: 41 89 c4      mov     %eax,%r12d
709 40134a: 48 89 ef      mov     %rbp,%rdi
710 40134d: e8 c9 ff ff ff callq   40131b <string_length>
711 401352: ba 01 00 00 00 mov     $0x1,%edx
712 401357: 41 39 c4      cmp     %eax,%r12d
713 40135a: 75 3f        jne     40139b <strings_not_equal+0x63>
714 40135c: 0f b6 03      movzbl  (%rbx),%eax
715 40135f: 84 c0        test    %al,%al
716 401361: 74 25        je      401388 <strings_not_equal+0x50>
717 401363: 3a 45 00      cmp     0x0(%rbp),%al
718 401366: 74 0a        je      401372 <strings_not_equal+0x3a>
719 401368: eb 25        jmp     40138f <strings_not_equal+0x57>
720 40136a: 3a 45 00      cmp     0x0(%rbp),%al
721 40136d: 0f 1f 00      nopl    (%rax)
722 401370: 75 24        jne     401396 <strings_not_equal+0x5e>
723 401372: 48 83 c3 01   add     $0x1,%rbx
724 401376: 48 83 c5 01   add     $0x1,%rbp
725 40137a: 0f b6 03      movzbl  (%rbx),%eax
726 40137d: 84 c0        test    %al,%al
727 40137f: 75 e9        jne     40136a <strings_not_equal+0x32>
728 401381: ba 00 00 00 00 mov     $0x0,%edx
729 401386: eb 13        jmp     40139b <strings_not_equal+0x63>
730 401388: ba 00 00 00 00 mov     $0x0,%edx
731 40138d: eb 0c        jmp     40139b <strings_not_equal+0x63>
732 40138f: ba 01 00 00 00 mov     $0x1,%edx
733 401394: eb 05        jmp     40139b <strings_not_equal+0x63>
734 401396: ba 01 00 00 00 mov     $0x1,%edx
735 40139b: 89 d0        mov     %edx,%eax
736 40139d: 5b          pop     %rbx
737 40139e: 5d          pop     %rbp
738 40139f: 41 5c        pop     %r12
739 4013a1: c3          retq

```

```

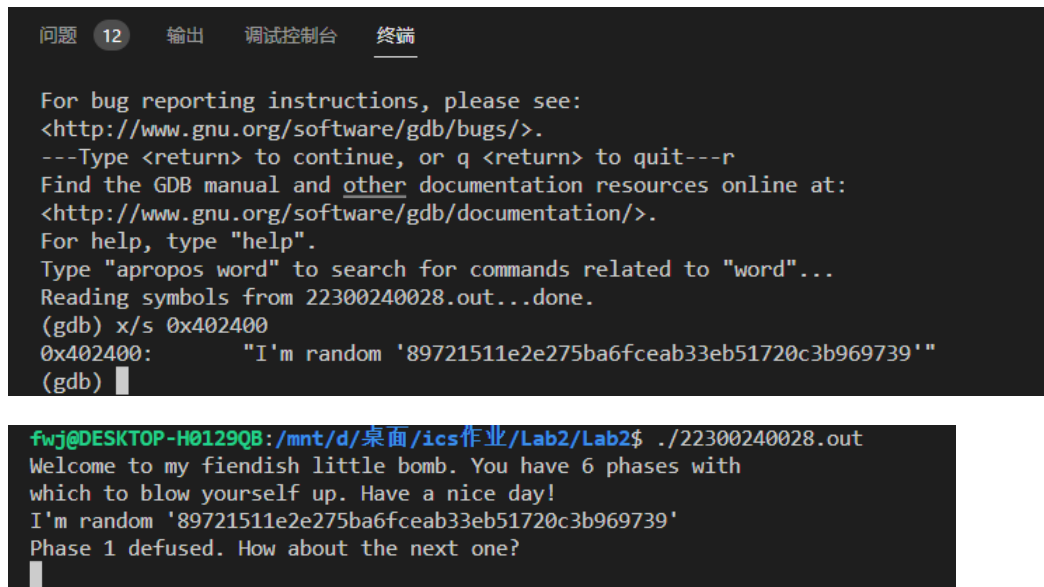
688 00000000040131b <string_length>:
689 40131b: 80 3f 00      cmpb  $0x0, (%rdi)
690 40131e: 74 12         je    401332 <string_length+0x17>
691 401320: 48 89 fa      mov   %rdi,%rdx
692 401323: 48 83 c2 01    add   $0x1,%rdx
693 401327: 89 d0         mov   %edx,%eax
694 401329: 29 f8         sub   %edi,%eax
695 40132b: 80 3a 00      cmpb  $0x0, (%rdx)
696 40132e: 75 f3         jne   401323 <string_length+0x8>
697 401330: f3 c3         repz retq
698 401332: b8 00 00 00 00 mov   $0x0,%eax
699 401337: c3           retq
700

```

- 观察string_length函数，可知传入参数应该是字符串指针，该函数的大致作用是：将字符串指针一个一个往后指，同时更新字符串长度，如果指针指向了0x0，即字符串末尾的'\0'则返回。
- 回过头来继续观察strings_not_equal函数，我们可以理解：到710行为止，函数求出了输入字符串的长度，存放在%r12d中，答案字符串的长度，存放在%eax中。接下来将%edx赋值为1，判断两个字符串长度是否相等，如果不相等则跳转到735行，最后函数会返回1。接下来判断输入字符串中当前是否为空，为空则返回0，不为空则往后一个字符一个字符比较，一旦不相等就返回1。这基本验证了我们的猜想，所以本题的答案应该就是phase_1函数中%esi寄存器所存储的指针所指向的字符串。
- 所以我们只需要gdb 22300240028.out 并x/s 0x402400查看答案字符串即可。
- phase_1 defuse成功!

2.3 phase_2

- 前四行压入寄存器，分配栈帧，保存%rsp的值到%rsi。接下来调用了函数read_six_numbers。
- 在read_six_numbers函数中，我们发现805到813行运用lea和mov为寄存器赋值，而814行将一个看起来像地址一样的值赋给了%esi，不妨用gdb查



```

问题 12  输出  调试控制台  终端

For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
---Type <return> to continue, or q <return> to quit---r
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from 22300240028.out...done.
(gdb) x/s 0x402400
0x402400:      "I'm random '89721511e2e275ba6fceb33eb51720c3b969739'"
(gdb)

fwj@DESKTOP-H0129QB:/mnt/d/桌面/ics作业/Lab2/Lab2$ ./22300240028.out
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I'm random '89721511e2e275ba6fceb33eb51720c3b969739'
Phase 1 defused. How about the next one?

```

看一下。可以猜想我们需要输入六个整数并以空格隔开。

- 再回过头来看phase_2函数。读入后首先判断栈顶指针指向的地址的值，即 $M[R[\%rsp]]$ 是否为1。接下来不难发现是一个循环：以 $\%rbx$ 为计数器指针，每次加4，以 $\%rbp$ 为终点指针， $0x18 / 0x4$ 刚好等于6，意味着循环5次。在每次循环中 $\%eax$ 存储之前的 $\%rbx$ 寄存器指向的地址的值，并翻倍，与现在的寄存器 $\%rbx$ 寄存器指向的地址的值进行比较，如果不一样就会使得炸弹爆炸。
- 因此可以大致得出答案是1 2 4 8 16 32。
- 不过这么判断的前提是：read_six_numbers函数会将 $M[R[\%rsp]], M[R[\%rsp]+0x4], \dots, M[R[\%rsp]+0x14]$ 设置成输入的六个整数。因此我们还需要回过头来看一下这个函数一开始的寄存器赋值操作。
- $\%rdi$ 和 $\%rsi$ （原来的栈顶）作为前两个参数传入。read_six_numbers函数又将 $M[R[\%rsi]], M[R[\%rsi]+0x4], \dots, M[R[\%rsi]+0x14]$ 作为第3到8个传入参数，由于超出寄存器数量限制，最后两个参数被转移到内存上。如下图所示。同时我们可以看到这个函数的结尾处的跳出判定是大于5就可以，也就是说我们可以在答案结尾添加任意字符（和数字分开即可），例如：1 2 4 8 16 32*。

```

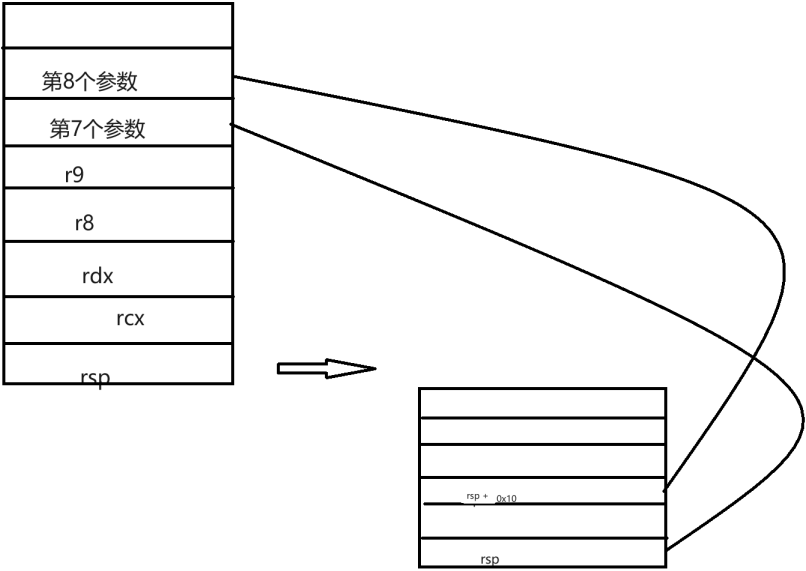
356 000000000400efc <phase_2>:
357 400efc: 55          push    %rbp
358 400efd: 53          push    %rbx
359 400efe: 48 83 ec 28  sub    $0x28,%rsp
360 400f02: 48 89 e6     mov     %rsp,%rsi
361 400f05: e8 52 05 00 00 callq   40145c <read_six_numbers>
362 400f0a: 83 3c 24 01  cmpl    $0x1,(%rsp)
363 400f0e: 74 20       je      400f30 <phase_2+0x34>
364 400f10: e8 25 05 00 00 callq   40143a <explode_bomb>
365 400f15: eb 19       jmp     400f30 <phase_2+0x34>
366 400f17: 8b 43 fc     mov     -0x4(%rbx),%eax
367 400f1a: 01 c0       add     %eax,%eax
368 400f1c: 39 03       cmp     %eax,[%rbx]
369 400f1e: 74 05       je      400f25 <phase_2+0x29>
370 400f20: e8 15 05 00 00 callq   40143a <explode_bomb>
371 400f25: 48 83 c3 04  add     $0x4,%rbx
372 400f29: 48 39 eb     cmp     %rbp,%rbx
373 400f2c: 75 e9       jne     400f17 <phase_2+0x1b>
374 400f2e: eb 0c       jmp     400f3c <phase_2+0x40>
375 400f30: 48 8d 5c 24 04 lea     0x4(%rsp),%rbx
376 400f35: 48 8d 6c 24 18 lea     0x18(%rsp),%rbp
377 400f3a: eb db       jmp     400f17 <phase_2+0x1b>
378 400f3c: 48 83 c4 28  add     $0x28,%rsp
379 400f40: 5b         pop     %rbx
380 400f41: 5d         pop     %rbp
381 400f42: c3         retq
382

```



```
804 00000000040145c <read_six_numbers>:
805 40145c: 48 83 ec 18      sub    $0x18,%rsp
806 401460: 48 89 f2          mov    %rsi,%rdx
807 401463: 48 8d 4e 04      lea    0x4(%rsi),%rcx
808 401467: 48 8d 46 14      lea    0x14(%rsi),%rax
809 40146b: 48 89 44 24 08    mov    %rax,0x8(%rsp)
810 401470: 48 8d 46 10      lea    0x10(%rsi),%rax
811 401474: 48 89 04 24      mov    %rax,(%rsp)
812 401478: 4c 8d 4e 0c      lea    0xc(%rsi),%r9
813 40147c: 4c 8d 46 08      lea    0x8(%rsi),%r8
814 401480: be c3 25 40 00    mov    $0x4025c3,%esi
815 401485: b8 00 00 00 00    mov    $0x0,%eax
816 40148a: e8 61 f7 ff ff    callq 400bf0 <__isoc99_sscanf@plt>
817 40148f: 83 f8 05          cmp    $0x5,%eax
818 401492: 7f 05            jg     401499 <read_six_numbers+0x3d>
819 401494: e8 a1 ff ff ff    callq 40143a <explode_bomb>
820 401499: 48 83 c4 18      add    $0x18,%rsp
821 40149d: c3              retq
```

0x4025c3: "%d %d %d %d %d %d"
(gdb) █



- phase_2 defuse成功!

```
fwj@DESKTOP-H0129QB:/mnt/d/桌面/ics作业/Lab2/Lab2$ ./22300240028.out
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I'm random '89721511e2e275ba6fceb33eb51720c3b969739'
Phase 1 defused. How about the next one?
1 2 4 8 16 32*
That's number 2. Keep going!
```

2.4 phase_3

- 在第387行可以发现和phase_2函数中类似的读入，用gdb调试后可知需要输入两个整数，以空格隔开。同样可以在答案结尾添加任意字符（和数字分开即可）。
- 第393和394行表示如果 $M[R[\%rsp]+8] > 7$ 就会爆炸。接下来是一个奇怪的间接跳转，跳转到 $M[0x402470+8*R[\%rax]]$ ，注意这段地址储存在内存中。注意到之前将 $M[R[\%rsp]+8]$ 赋值给了 $\%eax$ ，所以 $\%rax$ 是一个小于等于7的无符号型整数。所以跳转到的地址有： $M[0x402470], M[0x402478], \dots, M[0x4024a8]$ 这8种情况。不妨用gdb查看一下。发现这分别与第397,414,399,401,403,405,407,409行相对应。
- 然后将 $\%eax$ 赋值为一个常数，并跳转到第415行比较第二个参数和这个常数是否相等，不相等则爆炸。因此我们可以得出答案为：

第一个参数	第二个参数
0	207
1	311
2	707
3	256
4	389
5	206
6	682
7	327

```

383 0000000000400f43 <phase_3>:
384 400f43: 48 83 ec 18      sub    $0x18,%rsp
385 400f47: 48 8d 4c 24 0c    lea    0xc(%rsp),%rcx
386 400f4c: 48 8d 54 24 08    lea    0x8(%rsp),%rdx
387 400f51: be cf 25 40 00    mov    $0x4025cf,%esi
388 400f56: b8 00 00 00 00    mov    $0x0,%eax
389 400f5b: e8 90 fc ff ff    callq 400bf0 <__isoc99_sscanf@plt>
390 400f60: 83 f8 01         cmp    $0x1,%eax
391 400f63: 7f 05           jg     400f6a <phase_3+0x27>
392 400f65: e8 d0 04 00 00    callq 40143a <explode_bomb>
393 400f6a: 83 7c 24 08 07    cmpl   $0x7,0x8(%rsp)
394 400f6f: 77 3c           ja     400fad <phase_3+0x6a>
395 400f71: 8b 44 24 08      mov    0x8(%rsp),%eax
396 400f75: ff 24 c5 70 24 40 00 jmpq   *0x402470(,%rax,8)
397 400f7c: b8 cf 00 00 00    mov    $0xcf,%eax
398 400f81: eb 3b           jmp    400fbe <phase_3+0x7b>
399 400f83: b8 c3 02 00 00    mov    $0x2c3,%eax
400 400f88: eb 34           jmp    400fbe <phase_3+0x7b>
401 400f8a: b8 00 01 00 00    mov    $0x100,%eax
402 400f8f: eb 2d           jmp    400fbe <phase_3+0x7b>
403 400f91: b8 85 01 00 00    mov    $0x185,%eax
404 400f96: eb 26           jmp    400fbe <phase_3+0x7b>
405 400f98: b8 ce 00 00 00    mov    $0xce,%eax
406 400f9d: eb 1f           jmp    400fbe <phase_3+0x7b>
407 400f9f: b8 aa 02 00 00    mov    $0x2aa,%eax
408 400fa4: eb 18           jmp    400fbe <phase_3+0x7b>
409 400fa6: b8 47 01 00 00    mov    $0x147,%eax
410 400fab: eb 11           jmp    400fbe <phase_3+0x7b>
411 400fad: e8 88 04 00 00    callq 40143a <explode_bomb>
412 400fb2: b8 00 00 00 00    mov    $0x0,%eax
413 400fb7: eb 05           jmp    400fbe <phase_3+0x7b>
414 400fb9: b8 37 01 00 00    mov    $0x137,%eax
415 400fbe: 3b 44 24 0c      cmp    0xc(%rsp),%eax
416 400fc2: 74 05           je     400fc9 <phase_3+0x86>
417 400fc4: e8 71 04 00 00    callq 40143a <explode_bomb>
418 400fc9: 48 83 c4 18      add    $0x18,%rsp
419 400fcd: c3             retq
420

```

```

(gdb) x/s 0x4025cf
0x4025cf:      "%d %d"
(gdb) 

```

```
(gdb) x/8g 0x402470
0x402470:      0x0000000000400f7c      0x0000000000400fb9
0x402480:      0x0000000000400f83      0x0000000000400f8a
0x402490:      0x0000000000400f91      0x0000000000400f98
0x4024a0:      0x0000000000400f9f      0x0000000000400fa6
(gdb)
```

- 我们取0 207*。phase_3 defuse成功!

```
fwj@DESKTOP-H0129QB:/mnt/d/桌面/ics作业/Lab2/Lab2$ ./22300240028.out
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I'm random '89721511e2e275ba6fceab33eb51720c3b969739'
Phase 1 defused. How about the next one?
1 2 4 8 16 32*
That's number 2. Keep going!
0 207*
Halfway there!
```

2.5 phase_4

- 同phase_3，在第449行经gdb调试（x/s 0x4025cf）可知需要输入两个整数。但是根据判断条件（第452，453行），我们需要输入三个参数。
- 首先判断输入的第一个整数是否小于等于0xe，如果大于则爆炸，小于则把%edx赋成0xe、把%esi赋成0x0、把 $M[R[\%rsp] + 8] \leq 0xe$ 赋给%edi，调用fun_4并传入参数。如果该函数返回值为0，并且第二个参数为0，则可以跳过爆炸。
- 可以看出该函数是一个递归函数，我们写出它对应的C语言代码来进行分析。为了直接求出答案，我们遍历所有可能的输入值。

```
1 #include <stdio.h>
2 int cnt = 0;
3 int flag = 0;
4 //edi=input, edx = 14, esi = 0
```

```

445 000000000040100c <phase_4>:
446 40100c: 48 83 ec 18      sub    $0x18,%rsp
447 401010: 48 8d 4c 24 0c    lea    0xc(%rsp),%rcx
448 401015: 48 8d 54 24 08    lea    0x8(%rsp),%rdx
449 40101a: be cf 25 40 00    mov    $0x4025cf,%esi
450 40101f: b8 00 00 00 00    mov    $0x0,%eax
451 401024: e8 c7 fb ff ff    callq 400bf0 <__isoc99_sscanf@plt>
452 401029: 83 f8 02          cmp    $0x2,%eax
453 40102c: 75 07             jne    401035 <phase_4+0x29>
454 40102e: 83 7c 24 08 0e    cmpl   $0xe,0x8(%rsp)
455 401033: 76 05             jbe    40103a <phase_4+0x2e>
456 401035: e8 00 04 00 00    callq 40143a <explode_bomb>
457 40103a: ba 0e 00 00 00    mov    $0xe,%edx
458 40103f: be 00 00 00 00    mov    $0x0,%esi
459 401044: 8b 7c 24 08       mov    0x8(%rsp),%edi
460 401048: e8 81 ff ff ff    callq 400fce <func4>
461 40104d: 85 c0             test   %eax,%eax
462 40104f: 75 07             jne    401058 <phase_4+0x4c>
463 401051: 83 7c 24 0c 00    cmpl   $0x0,0xc(%rsp)
464 401056: 74 05             je     40105d <phase_4+0x51>
465 401058: e8 dd 03 00 00    callq 40143a <explode_bomb>
466 40105d: 48 83 c4 18       add    $0x18,%rsp
467 401061: c3               retq
468

```

```

(gdb) x/s 0x4025cf
0x4025cf:      "%d %d"
(gdb) █

```

```
421 0000000000400fce <func4>:
422 400fce: 48 83 ec 08      sub    $0x8,%rsp
423 400fd2: 89 d0            mov    %edx,%eax
424 400fd4: 29 f0            sub    %esi,%eax
425 400fd6: 89 c1            mov    %eax,%ecx
426 400fd8: c1 e9 1f         shr    $0x1f,%ecx
427 400fdb: 01 c8            add    %ecx,%eax
428 400fdd: d1 f8            sar    %eax
429 400fdf: 8d 0c 30         lea    (%rax,%rsi,1),%ecx
430 400fe2: 39 f9            cmp    %edi,%ecx
431 400fe4: 7e 0c            jle    400ff2 <func4+0x24>
432 400fe6: 8d 51 ff         lea    -0x1(%rcx),%edx
433 400fe9: e8 e0 ff ff ff  callq  400fce <func4>
434 400fee: 01 c0            add    %eax,%eax
435 400ff0: eb 15            jmp     401007 <func4+0x39>
436 400ff2: b8 00 00 00 00  mov    $0x0,%eax
437 400ff7: 39 f9            cmp    %edi,%ecx
438 400ff9: 7d 0c            jge    401007 <func4+0x39>
439 400ffb: 8d 71 01         lea    0x1(%rcx),%esi
440 400ffe: e8 cb ff ff ff  callq  400fce <func4>
441 401003: 8d 44 00 01         lea    0x1(%rax,%rax,1),%eax
442 401007: 48 83 c4 08      add    $0x8,%rsp
443 40100b: c3              retq
444
```

```
5 int fun_4(int esi, int edx, int edi)
6 {
7     cnt ++;
8     if(cnt >= 10000) {
9         flag = 1;
10        return 10086;
11    }
12    int eax = edx;
13    eax -= esi;
14    //eax = edx - esi;
15    int ecx = eax;
16    //ecx >>= 31;
17    for(int i = 1; i <=31; i ++) ecx /= 2;
18    eax += ecx;
19    eax >>= 1;
20    //eax = (eax + eax>>31)>>1;
21    ecx = eax + esi;
22    if(ecx > edi){
23        edx = ecx - 1;
24        return 2 * fun_4(es, edx, edi);
25    }
26    }else{
27        eax = 0;
28        if(ecx >= edi){
29            return eax;
30        }else{
31            esi = ecx - 1;
32            return 2 * fun_4(es, edx, edi) + 1;
33        }
34    }
35 }
```

```
36 int main()
37 {
38     for(int i = 0; i <= 14; i ++){
39         cnt = 0;
40         flag = 0;
41         int res = fun_4(0, 14, i);
42         if(!flag)
43             printf("i=%d %d\n", i , res);
44         else
45             printf("i=%d No result\n", i);
46     }
47     return 0;
48 }
```

```
fwj@DESKTOP-H0129QB:/mnt/d/桌面/ics作业/Lab2/tryc/cs$ gcc main.c -o main
fwj@DESKTOP-H0129QB:/mnt/d/桌面/ics作业/Lab2/tryc/cs$ ./main
i=0 0
i=1 0
i=2 No result
i=3 0
i=4 2
i=5 No result
i=6 No result
i=7 0
i=8 No result
i=9 No result
i=10 1
i=11 3
i=12 7
i=13 No result
i=14 No result
fwj@DESKTOP-H0129QB:/mnt/d/桌面/ics作业/Lab2/tryc/cs$
```

- 这里的C程序假设了`%eax = %rax`，会导致有的输入进入死循环，不过这已经可以得出部分答案，足以跳过爆炸了。

第一个参数	第二个参数
0	0
1	0
3	0
7	0

- 我们取7 0。phase_4 defuse成功!

```
fwj@DESKTOP-H0129QB:/mnt/d/桌面/ics作业/Lab2/Lab2$ ./22300240028.out
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I'm random '89721511e2e275ba6fceab33eb51720c3b969739'
Phase 1 defused. How about the next one?
1 2 4 8 16 32*
That's number 2. Keep going!
0 207*
Halfway there!
7 0
So you got that one. Try this one.
```

2.6 phase_5

- 第477行调用string_length函数求出输入字符串的长度，必须是6，否则爆炸。然后将%rax从1到5循环：每次将 $M[R[\%rbx] + R[\%rax]]$ 的后8位存在%rdx中并只取它后四位（&0xf）。把它作为索引找到 $M[0x4024b0 + R[\%rdx]]$ ，取它的后8位存到 $R[\%rsp] + R[\%rax] + 0x10$ 中。最后在结尾加上'\0'。并和%esi寄存器所存储的指针所指向的字符串相比较，如果相等则跳过爆炸。
- 用gdb调试得到0x4024b0和0x40245e处的字符串。
- 我们需要在第一个字符串里找第二个字符串的字母。也就是说，我们输入的6个字符的后四位十进制值必须是9 15 14 5 6 7。
- 编写C语言程序可以快速得到我们要找的答案。

```
1 #include <stdio.h>
2 int main()
```

```

469 000000000401062 <phase_5>:
470 401062: 53                push   %rbx
471 401063: 48 83 ec 20       sub    $0x20,%rsp
472 401067: 48 89 fb         mov    %rdi,%rbx
473 40106a: 64 48 8b 04 25 28 00 mov    %fs:0x28,%rax
474 401071: 00 00
475 401073: 48 89 44 24 18     mov    %rax,0x18(%rsp)
476 401078: 31 c0            xor    %eax,%eax
477 40107a: e8 9c 02 00 00     callq 40131b <string_length>
478 40107f: 83 f8 06         cmp    $0x6,%eax
479 401082: 74 4e            je     4010d2 <phase_5+0x70>
480 401084: e8 b1 03 00 00     callq 40143a <explode_bomb>
481 401089: eb 47            jmp    4010d2 <phase_5+0x70>
482 40108b: 0f b6 0c 03       movzbl (%rbx,%rax,1),%ecx
483 40108f: 88 0c 24         mov    %cl,(%rsp)
484 401092: 48 8b 14 24       mov    (%rsp),%rdx
485 401096: 83 e2 0f         and    $0xf,%edx
486 401099: 0f b6 92 b0 24 40 00 movzbl 0x4024b0(%rdx),%edx
487 4010a0: 88 54 04 10       mov    %dl,0x10(%rsp,%rax,1)
488 4010a4: 48 83 c0 01       add    $0x1,%rax
489 4010a8: 48 83 f8 06       cmp    $0x6,%rax
490 4010ac: 75 dd            jne    40108b <phase_5+0x29>
491 4010ae: c6 44 24 16 00     movb   $0x0,0x16(%rsp)
492 4010b3: be 5e 24 40 00     mov    $0x40245e,%esi
493 4010b8: 48 8d 7c 24 10     lea    0x10(%rsp),%rdi
494 4010bd: e8 76 02 00 00     callq 401338 <strings_not_equal>
495 4010c2: 85 c0            test   %eax,%eax
496 4010c4: 74 13            je     4010d9 <phase_5+0x77>
497 4010c6: e8 6f 03 00 00     callq 40143a <explode_bomb>
498 4010cb: 0f 1f 44 00 00     nopl   0x0(%rax,%rax,1)
499 4010d0: eb 07            jmp    4010d9 <phase_5+0x77>
500 4010d2: b8 00 00 00 00     mov    $0x0,%eax
501 4010d7: eb b2            jmp    40108b <phase_5+0x29>
502 4010d9: 48 8b 44 24 18     mov    0x18(%rsp),%rax
503 4010de: 64 48 33 04 25 28 00 xor    %fs:0x28,%rax
504 4010e5: 00 00
505 4010e7: 74 05            je     4010ee <phase_5+0x8c>
506 4010e9: e8 42 fa ff ff     callq 400b30 <__stack_chk_fail@plt>
507 4010ee: 48 83 c4 20       add    $0x20,%rsp
508 4010f2: 5b              pop    %rbx
509 4010f3: c3              retq

```

```

(gdb) x/s 0x4024b0
0x4024b0 <array.3449>: "madiuersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?"
0x40245e: "flyers"
(gdb) 

```

```

3 {
4     for(int i = 0; i <= 255; i ++)
5     {
6         if(i % 16 == 5)
7             printf("5: %c\n", i);
8         else if(i % 16 == 6)
9             printf("6: %c\n", i);
10        else if(i % 16 == 7)
11            printf("7: %c\n", i);
12        else if(i % 16 == 9)
13            printf("9: %c\n", i);
14        else if(i % 16 == 14)
15            printf("e: %c\n", i);
16        else if(i % 16 == 15)
17            printf("f: %c\n", i);
18    }
19    return 0;
20 }

```

```

fwj@DESKTOP-H0129Q8:/mnt/d/桌面/ics作业/Lab2/tryc/cs$ gcc ascii.c -o ascii
fwj@DESKTOP-H0129Q8:/mnt/d/桌面/ics作业/Lab2/tryc/cs$ ./ascii
5: 6: 7: 9:      e: f: 5: 6: 7: 9: e: f: 5: % 6: & 7: ' 9: > e: . f: / 5: 5 6: 6 7: 7 9: 9 e: > f: ? 5: E 6: F 7: G 9: I e: N f: O 5: U 6: V 7:
W 9: Y e: ^ f: _ 5: e 6: f 7: g 9: i e: n f: o 5: u 6: v 7: w 9: y e: ~ f: 5: 6: 7: 9: e: 5: 6: 7: 9: e: 5: 6: 7: 9: e: 5: 6: 7: 9: e:
f: 5: 6: 7: 9: e: f: 5: 6: 7: 9: e: f: 5: 6: 7: 9: e: f: 5: 6: 7: 9: e: f: 5: 6: 7: 9: e: f: 5: 6: 7: 9: e: f: 5: 6: 7: 9: e: f: 5: 6: 7: 9: e:
fwj@DESKTOP-H0129Q8:/mnt/d/桌面/ics作业/Lab2/tryc/cs$

```

- 不妨取)/ > %FG。phase_5 defuse成功!

```

So you got that one. Try this one.
)/>%FG
Good work! On to the next...

```

2.7 phase_6

- 第520行看到read_six_numbers函数可知调用者的栈上按顺序存储了输入的六个数。接下来到542行为止是循环中套了循环，需要保证：每个数字小于等于6且互不相同。其C语言代码如下：

```

511 0000000004010f4 <phase_6>:
512 4010f4: 41 56          push  %r14
513 4010f6: 41 55          push  %r13
514 4010f8: 41 54          push  %r12
515 4010fa: 55            push  %rbp
516 4010fb: 53            push  %rbx
517 4010fc: 48 83 ec 50    sub   $0x50,%rsp
518 401100: 49 89 e5       mov   %rsp,%r13
519 401103: 48 89 e6       mov   %rsp,%rsi
520 401106: e8 51 03 00 00 callq 40145c <read_six_numbers>
521 40110b: 49 89 e6       mov   %rsp,%r14
522 40110e: 41 bc 00 00 00 00 mov   $0x0,%r12d
523 401114: 4c 89 ed       mov   %r13,%rbp
524 401117: 41 8b 45 00     mov   0x0(%r13),%eax
525 40111b: 83 e8 01       sub   $0x1,%eax
526 40111e: 83 f8 05       cmp   $0x5,%eax
527 401121: 76 05         jbe   401128 <phase_6+0x34>
528 401123: e8 12 03 00 00 callq 40143a <explode_bomb>
529 401128: 41 83 c4 01     add   $0x1,%r12d
530 40112c: 41 83 fc 06     cmp   $0x6,%r12d
531 401130: 74 21         je    401153 <phase_6+0x5f>
532 401132: 44 89 e3       mov   %r12d,%ebx
533 401135: 48 63 c3       movslq %ebx,%rax
534 401138: 8b 04 84       mov   (%rsp,%rax,4),%eax
535 40113b: 39 45 00       cmp   %eax,0x0(%rbp)
536 40113e: 75 05         jne   401145 <phase_6+0x51>
537 401140: e8 f5 02 00 00 callq 40143a <explode_bomb>
538 401145: 83 c3 01       add   $0x1,%ebx
539 401148: 83 fb 05       cmp   $0x5,%ebx
540 40114b: 7e e8         jle   401135 <phase_6+0x41>
541 40114d: 49 83 c5 04     add   $0x4,%r13
542 401151: eb c1         jmp   401114 <phase_6+0x20>
543 401153: 48 8d 74 24 18 lea    0x18(%rsp),%rsi
544 401158: 4c 89 f0       mov   %r14,%rax
545 40115b: b9 07 00 00 00 mov   $0x7,%ecx
546 401160: 89 ca         mov   %ecx,%edx
547 401162: 2b 10         sub   (%rax),%edx
548 401164: 89 10         mov   %edx,(%rax)
549 401166: 48 83 c0 04     add   $0x4,%rax
550 40116a: 48 39 f0       cmp   %rsi,%rax
551 40116d: 75 f1         jne   401160 <phase_6+0x6c>
552 40116f: be 00 00 00 00 mov   $0x0,%esi
553 401174: eb 21         jmp   401107 <phase_6+0xa3>
554 401176: 48 8b 52 08     mov   0x8(%rdx),%rdx
555 40117a: 83 c0 01       add   $0x1,%eax
556 40117d: 39 c8         cmp   %ecx,%eax
557 40117f: 75 f5         jne   401176 <phase_6+0x82>

558 401181: eb 05         jmp   401188 <phase_6+0x94>
559 401183: ba d0 32 60 00 mov   $0x6032d0,%edx
560 401188: 48 89 54 74 20 mov   %rdx,0x20(%rsp,%rsi,2)
561 40118d: 48 83 c6 04     add   $0x4,%rsi
562 401191: 48 83 fe 18     cmp   $0x18,%rsi
563 401195: 74 14         je    4011ab <phase_6+0xb7>
564 401197: 8b 0c 34       mov   (%rsp,%rsi,1),%ecx
565 40119a: 83 f9 01       cmp   $0x1,%ecx
566 40119d: 7e e4         jle   401183 <phase_6+0x8f>
567 40119f: b8 01 00 00 00 mov   $0x1,%eax
568 4011a4: ba d0 32 60 00 mov   $0x6032d0,%edx
569 4011a9: eb cb         jmp   401176 <phase_6+0x82>
570 4011ab: 48 8b 5c 24 20 mov   0x20(%rsp),%rbx
571 4011b0: 48 8d 44 24 28 lea    0x28(%rsp),%rax
572 4011b5: 48 8d 74 24 50 lea    0x50(%rsp),%rsi
573 4011ba: 48 89 d9       mov   %rbx,%rcx
574 4011bd: 48 8b 10       mov   (%rax),%rdx
575 4011c0: 48 89 51 08     mov   %rdx,0x8(%rcx)
576 4011c4: 48 83 c0 08     add   $0x8,%rax
577 4011c8: 48 39 f0       cmp   %rsi,%rax
578 4011cb: 74 05         je    4011d2 <phase_6+0xde>
579 4011cd: 48 89 d1       mov   %rdx,%rcx
580 4011d0: eb eb         jmp   4011bd <phase_6+0xc9>
581 4011d2: 48 c7 42 08 00 00 00 movq   $0x0,0x8(%rdx)
582 4011d9: 00
583 4011da: bd 05 00 00 00 mov   $0x5,%ebp
584 4011df: 48 8b 43 08     mov   0x8(%rbx),%rax
585 4011e3: 8b 00         mov   (%rax),%eax
586 4011e5: 39 03         cmp   %eax,(%rbx)
587 4011e7: 7d 05         jge   4011ee <phase_6+0xfa>
588 4011e9: e8 4c 02 00 00 callq 40143a <explode_bomb>
589 4011ee: 48 8b 5b 08     mov   0x8(%rbx),%rbx
590 4011f2: 83 ed 01       sub   $0x1,%ebp
591 4011f5: 75 e8         jne   4011df <phase_6+0xeb>
592 4011f7: 48 83 c4 50     add   $0x50,%rsp
593 4011fb: 5b           pop   %rbx
594 4011fc: 5d           pop   %rbp
595 4011fd: 41 5c         pop   %r12
596 4011ff: 41 5d         pop   %r13
597 401201: 41 5e         pop   %r14
598 401203: c3           retq
599

```

```

1 r14 = 0;
2 r13 = 0;
3 r12d = 0;
4 while(1){
5     rbp = r13;
6     if(num[r13] - 1 > 5)
7         goto bomb;
8     r12d++;
9     if(r12d == 6)
10        break;
11    for(ebx = r12d; ebx <= 5; ebx++){
12        if(num[ebx] == num[rbp])
13            goto bomb;
14    }
15    r13++;
16 }

```

- 接下来到551行是一个循环：将六个整数分别赋值成为用7减去它们自身。
- 接下来到569行是一个链表操作，通过我们输入的6个数字为索引对链表进行重排。即 $M[R[\%rsp]+0x20] = node[M[R[\%rsp]]]$, $M[R[\%rsp]+0x28] = node[M[R[\%rsp]+0x4]]$, \dots , $M[R[\%rsp]+0x48] = node[M[R[\%rsp]+0x14]]$ 。

```

(gdb) x/24wx 0x6032d0
0x6032d0 <node1>: 0x0000014c 0x00000001 0x006032e0 0x00000000
0x6032e0 <node2>: 0x000000a8 0x00000002 0x006032f0 0x00000000
0x6032f0 <node3>: 0x0000039c 0x00000003 0x00603300 0x00000000
0x603300 <node4>: 0x000002b3 0x00000004 0x00603310 0x00000000
0x603310 <node5>: 0x000001dd 0x00000005 0x00603320 0x00000000
0x603320 <node6>: 0x000001bb 0x00000006 0x00000000 0x00000000
(gdb)

```

- 接下来到580行按照栈内链表节点位置顺序重排单链表。即原来链表的 $1 \rightarrow 2 \rightarrow \dots \rightarrow 6$ 顺序被修改为 $M[R[\%rsp]] \rightarrow M[R[\%rsp]+0x4] \rightarrow \dots \rightarrow M[R[\%rsp]+0x14]$

- 最后的循环是用来判断新链表是否是非严格单调递减的。由于： $node3 > node4 > node5 > node6 > node1 > node2$ ，所以栈中的六个数应该是3 4 5 6 1 2。由于输入的数被7减过，所以答案应该是：4 3 2 1 6 5。
- phase_6 defuse成功!

```
fwj@DESKTOP-H0129QB:/mnt/d/桌面/ics作业/Lab2/Lab2$ ./22300240028.out
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I'm random '89721511e2e275ba6fceab33eb51720c3b969739'
Good work! On to the next...
4 3 2 1 6 5
Congratulations! You've defused the bomb!
```

2.8 secret_phase

- 我们发现bomb.s中还有secret_phase函数，但是之前没有遇到。Ctrl+F查询发现phase_defused函数调用了它。用gdb查看一些地址的字符串时发现了隐藏关卡的突破口：当我们在某个关卡中要输入三个参数，并且最后一个参数是"HiEvil"时就能开启隐藏关卡。检查了一下前面的发现phase_4需要我们输入三个参数，但是只有前两个参数在当时是有用的。

```
(gdb) x/s 0x402619
0x402619: "%d %d %s"
(gdb) x/s 0x603870
0x603870 <input_strings+240>: ""
(gdb) x/s 0x402622
0x402622: "HiEvil"
(gdb) x/s 0x402558
0x402558: "Congratulations! You've defused the bomb!"
(gdb) x/s 0x402520
0x402520: "But finding it and solving it are quite different..."
(gdb) x/s 0x4024f8
0x4024f8: "Curses, you've found the secret phase!"
(gdb) □
```

- 尝试了一下，开启了隐藏关卡。
- 发现在secret_phase中也有read_line，说明我们要在最后再输入一个参数。这个参数和一个地址0x6030f0作为两个参数传入fun_7函数，我们需要使得这个函数的返回值为2。

```

which to blow yourself up. Have a nice day!
I'm random '89721511e2e275ba6fceab33eb51720c3b969739'
Phase 1 defused. How about the next one?
1 2 4 8 16 32*
That's number 2. Keep going!
0 207*
Halfway there!
7 0 HiEvil
So you got that one. Try this one.
)/>%FG
Good work! On to the next...
4 3 2 1 6 5
Curses, you've found the secret phase!
But finding it and solving it are quite different...

```

- 查看这个地址，发现是一个树型结构。

```

0x6030f0 <n1>: 0x00000024 0x00000000 0x00603110 0x00000000
0x603100 <n1+16>: 0x00603130 0x00000000 0x00000000 0x00000000
0x603110 <n21>: 0x00000008 0x00000000 0x00603190 0x00000000
0x603120 <n21+16>: 0x00603150 0x00000000 0x00000000 0x00000000
0x603130 <n22>: 0x00000032 0x00000000 0x00603170 0x00000000
0x603140 <n22+16>: 0x006031b0 0x00000000 0x00000000 0x00000000
0x603150 <n32>: 0x00000016 0x00000000 0x00603270 0x00000000
0x603160 <n32+16>: 0x00603230 0x00000000 0x00000000 0x00000000
0x603170 <n33>: 0x0000002d 0x00000000 0x006031d0 0x00000000
0x603180 <n33+16>: 0x00603290 0x00000000 0x00000000 0x00000000
0x603190 <n31>: 0x00000006 0x00000000 0x006031f0 0x00000000
0x6031a0 <n31+16>: 0x00603250 0x00000000 0x00000000 0x00000000
0x6031b0 <n34>: 0x0000006b 0x00000000 0x00603210 0x00000000
0x6031c0 <n34+16>: 0x006032b0 0x00000000 0x00000000 0x00000000
0x6031d0 <n45>: 0x00000028 0x00000000 0x00000000 0x00000000
0x6031e0 <n45+16>: 0x00000000 0x00000000 0x00000000 0x00000000
0x6031f0 <n41>: 0x00000001 0x00000000 0x00000000 0x00000000
0x603200 <n41+16>: 0x00000000 0x00000000 0x00000000 0x00000000
0x603210 <n47>: 0x00000063 0x00000000 0x00000000 0x00000000
0x603220 <n47+16>: 0x00000000 0x00000000 0x00000000 0x00000000
0x603230 <n44>: 0x00000023 0x00000000 0x00000000 0x00000000
0x603240 <n44+16>: 0x00000000 0x00000000 0x00000000 0x00000000
0x603250 <n42>: 0x00000007 0x00000000 0x00000000 0x00000000
0x603260 <n42+16>: 0x00000000 0x00000000 0x00000000 0x00000000
0x603270 <n43>: 0x00000014 0x00000000 0x00000000 0x00000000
0x603280 <n43+16>: 0x00000000 0x00000000 0x00000000 0x00000000
0x603290 <n46>: 0x0000002f 0x00000000 0x00000000 0x00000000
0x6032a0 <n46+16>: 0x00000000 0x00000000 0x00000000 0x00000000
0x6032b0 <n48>: 0x000000e9 0x00000000 0x00000000 0x00000000
0x6032c0 <n48+16>: 0x00000000 0x00000000 0x00000000 0x00000000

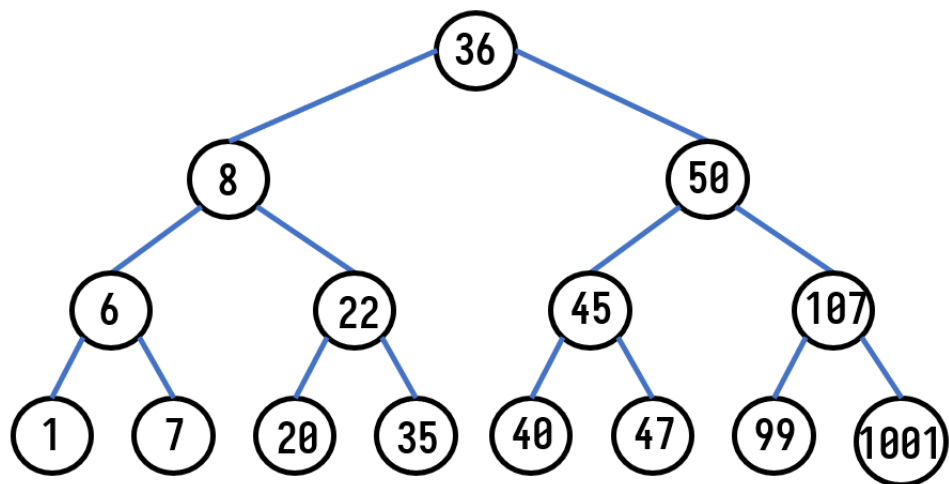
```

- 画出示意图。
- 查看fun_7函数，发现是一个递归函数，写出其C语言代码。

```

1 int fun7(Tree* rdi, int esi) {
2     if (!rdi)
3         return -1;
4     if (rdi->val == esi)
5         return 0;
6     else if (rdi->val < esi)
7         return 2 * fun7(rdi -> right, esi) + 1;

```



```

8     else
9         return 2 * fun7(rdi -> left, esi);
10 }

```

- 要使得其返回2，需要先从左子树返回1。要想返回1，需要先从右子树返回0。所以从36开始，esi一定小于36；进入左子树，esi一定大于8；进入右子树esi一定等于22。故答案为22。
- secret_phase defuse成功!

2.9 答案ans.txt

```

I'm random '89721511e2e275ba6fceab33eb51720c3b969739'
1 2 4 8 16 32*
0 207*
7 0 HiEvil
)/ > %FG
4 3 2 1 6 5
20
22300240028

```