计算机系统基础
实验报告

Lab 2

学生姓名： -

学　　号： -

日　　期： 2022.10.7

一：实验内容

拆二进制炸弹，使用 gdb 工具

二、实验结果

最终密码：

```
each line is important
1 -3 9 -27 81 -243
7 w 512
51539607566
8 2 4080
/(3-4)
1905
```

运行结果：

```
(gdb) r <password.txt
Starting program: /mnt/d/my_project/ICS/lab2/bomb <password.txt
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
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...
You've enter the float point world! It's not hard o(*^ _ ^*)m
Congratulations!
[Inferior 1 (process 322) exited normally]
(gdb)
```

三：实验过程

首先反汇编 bomb，使用 objdump，输入 objdump -d ./bomb > bomb.S 得到反汇编文件结果。在 bomb.S 中查找 main，发现 main 中调用 read_line 函数，猜测是用于读入数据，紧接着调用 phase_1 函数，猜测是第一个谜题的谜面。之后又如此调用 read_line；phase_2、3、4、5、6，上述猜想应该是正确的。

phase_1：

然后使用 gdb 反汇编 bomb，输入 b phase_1，在第一个谜面打断点，然后使用 disassemble 查看该函数反汇编结果。

```
Breakpoint 3, 0x0000555555555b12 in phase_1(char*) ()
(gdb) disas
Dump of assembler code for function _Z7phase_1Pc:
=> 0x0000555555555b12 <+0>:     endbr64
   0x0000555555555b16 <+4>:     push   %rdx
   0x0000555555555b17 <+5>:     movslq 0x2522(%rip),%rsi        # 0x555555558040 <phase_1_offset>
   0x0000555555555b1e <+12>:    lea    0x253b(%rip),%rax        # 0x555555558060 <w1>
   0x0000555555555b25 <+19>:    add    %rax,%rsi
   0x0000555555555b28 <+22>:    call   0x555555555a60 <_Z16string_not_equalPcS_>
   0x0000555555555b2d <+27>:    test   %al,%al
   0x0000555555555b2f <+29>:    jne    0x555555555b36 <_Z7phase_1Pc+36>
   0x0000555555555b31 <+31>:    call   0x555555555a46 <_Z12explode_bombv>
   0x0000555555555b36 <+36>:    pop    %rax
   0x0000555555555b37 <+37>:    ret
End of assembler dump.
(gdb)
```

发现调用 string_not_equal 函数，猜测是判断是否相等，这表明谜底已经有了，在某个位置。接下来查看各个寄存器的值：

```
End of assembler dump.
(gdb) i r
rax            0x7ffff7fac3b0      140737353794480
rbx            0x0                 0
rcx            0x7ffff7c77992      140737350433170
rdx            0x55555556b2c3      93824992326339
rsi            0x55555556b2c0      93824992326336
rdi            0x7fffffffdb08      140737488345864
rbp            0x7fffffffdb08      0x7fffffffdb08
rsp            0x7fffffffdae8      0x7fffffffdae8
r8             0x0                 0
r9             0x55555556b2c0      93824992326336
r10            0x77                119
r11            0x246               582
r12            0x7fffffffdaf0      140737488345840
r13            0x555555555200      93824992236032
r14            0x0                 0
r15            0x7ffff7ffd040      140737354125376
rip            0x555555555b12      0x555555555b12 <phase_1(char*)>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
k0             0x4000000           67108864
k1             0x0                 0
k2             0x7ff07ff           134154239
k3             0x0                 0
k4             0x0                 0
k5             0x0                 0
k6             0x0                 0
k7             0x0                 0
(gdb)
```

结合代码猜测答案可能在寄存器 rax 或者 rsi 中。先让函数运行到调用 string_not_equal 前一步，然后用 exam 查看二者的值：

```
(gdb) si 5
0x0000555555555b28 in phase_1(char*) ()
(gdb) x/s $rsi
0x555555558146 <w1+230>:        "each line is important"
(gdb) x/s $rax
0x555555558060 <w1>:    "This text introduced the main ideas in operating systems by studying one operating system"
```

复制，退出调试，删除所有断点，然后运行。测试得到答案应该是 rsi 中的字符串。

```
Kill the program being debugged? (y or n) y
[Inferior 1 (process 1218) killed]
(gdb) d breakpoint
Delete all breakpoints? (y or n) y
(gdb) r
Starting program: /mnt/d/my_project/ICS/lab2/bomb password.txt
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
You have 6 phases with which to blow yourself up. Have a nice day!
This text introduced the main ideas in operating systems by studying one operating syst

BOOM!!!
The bomb has blown up.
[Inferior 1 (process 3626) exited normally]
(gdb) r
Starting program: /mnt/d/my_project/ICS/lab2/bomb password.txt
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
You have 6 phases with which to blow yourself up. Have a nice day!
each line is important
Phase 1 defused. How about the next one?
```

phase_2：

在 phase_2 打上断点，运行程序，再用 disassemble 查看反汇编代码：

```
(gdb) i b
No breakpoints or watchpoints.
(gdb) b phase_2
Breakpoint 4 at 0x555555555b38
(gdb) r
Starting program: /mnt/d/my_project/ICS/lab2/bomb password.txt
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
You have 6 phases with which to blow yourself up. Have a nice day!
each line is important
Phase 1 defused. How about the next one?
1 2 3 4 5 6

Breakpoint 4, 0x0000555555555b38 in phase_2(char*) ()
(gdb) disas
Dump of assembler code for function _Z7phase_2Pc:
=> 0x0000555555555b38 <+0>:     endbr64
   0x0000555555555b3c <+4>:     push   %rdx
   0x0000555555555b3d <+5>:     lea    0x24dc(%rip),%rsi        # 0x555555558020 <phase_2_nums>
   0x0000555555555b44 <+12>:    call   0x555555555ad0 <_Z16read_six_numbersPcPi>
   0x0000555555555b49 <+17>:    lea    0x24d0(%rip),%rax        # 0x555555558020 <phase_2_nums>
   0x0000555555555b50 <+24>:    mov    0x24e6(%rip),%ecx        # 0x55555555803c <phase_2_nums+28>
   0x0000555555555b56 <+30>:    lea    0x14(%rax),%rdx
   0x0000555555555b5a <+34>:    mov    (%rax),%esi
   0x0000555555555b5c <+36>:    imul   %ecx,%esi
   0x0000555555555b5f <+39>:    cmp    %esi,0x4(%rax)
   0x0000555555555b62 <+42>:    je     0x555555555b69 <_Z7phase_2Pc+49>
   0x0000555555555b64 <+44>:    call   0x555555555a46 <_Z12explode_bombv>
   0x0000555555555b69 <+49>:    add    $0x4,%rax
   0x0000555555555b6d <+53>:    cmp    %rax,%rdx
   0x0000555555555b70 <+56>:    jne    0x555555555b5a <_Z7phase_2Pc+34>
   0x0000555555555b72 <+58>:    pop    %rax
   0x0000555555555b73 <+59>:    ret
End of assembler dump.
(gdb)
```

发现调用了 read_six_number 函数，猜测是读入 6 个数据。在+42 和+49 行发现了跳转的语句，根据内容猜测是一个循环。让程序运行到循环开始的地方，检查寄存器的值：

```
(gdb) b *phase_2 + 34
Breakpoint 5 at 0x555555555b5a
(gdb) c
Continuing.

Breakpoint 5, 0x0000555555555b5a in phase_2(char*) ()
(gdb) disas
Dump of assembler code for function _Z7phase_2Pc:
   0x0000555555555b38 <+0>:     endbr64
   0x0000555555555b3c <+4>:     push   %rdx
   0x0000555555555b3d <+5>:     lea    0x24dc(%rip),%rsi        # 0x555555558020 <phase_2_nums>
   0x0000555555555b44 <+12>:    call   0x555555555ad0 <_Z16read_six_numbersPcPi>
   0x0000555555555b49 <+17>:    lea    0x24d0(%rip),%rax        # 0x555555558020 <phase_2_nums>
   0x0000555555555b50 <+24>:    mov    0x24e6(%rip),%ecx        # 0x55555555803c <phase_2_nums+28>
   0x0000555555555b56 <+30>:    lea    0x14(%rax),%rdx
=> 0x0000555555555b5a <+34>:    mov    (%rax),%esi
   0x0000555555555b5c <+36>:    imul   %ecx,%esi
   0x0000555555555b5f <+39>:    cmp    %esi,0x4(%rax)
   0x0000555555555b62 <+42>:    je     0x555555555b69 <_Z7phase_2Pc+49>
   0x0000555555555b64 <+44>:    call   0x555555555a46 <_Z12explode_bombv>
   0x0000555555555b69 <+49>:    add    $0x4,%rax
   0x0000555555555b6d <+53>:    cmp    %rax,%rdx
   0x0000555555555b70 <+56>:    jne    0x555555555b5a <_Z7phase_2Pc+34>
   0x0000555555555b72 <+58>:    pop    %rax
   0x0000555555555b73 <+59>:    ret
End of assembler dump.
(gdb) i r
rax            0x555555558020      93824992247840
rbx            0x0                 0
```

rax 指向输入的第一个数，rcx 是常数-3

```
0x0000555555555b5c in phase_2(char*) ()
(gdb) x/d $rax
0x555555558020 <phase_2_nums>:  1
(gdb) x/d $rcx
0xffffffd:      Cannot access memory at address 0xffffffd
(gdb) x/d rcx
No symbol "rcx" in current context.
(gdb) i r rcx
rcx            0xfffffffd          4294967293
(gdb)
```

通过阅读反汇编代码，rax 每次自增 4，即指向下一个数，eax 的值是 rax 指向的数，并且每次判断 eax*（-3）是否等于下一个数。因此认为 phase_2 是一个公比为-3 的等比数列，测试后通过。

```
[Inferior 1 (process 6107) exited normally]
(gdb) r
Starting program: /mnt/d/my_project/ICS/lab2/bomb password.txt
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
You have 6 phases with which to blow yourself up. Have a nice day!
each line is important
Phase 1 defused. How about the next one?
1 -3 9 -27 81 -243
That's number 2.  Keep going!
```

phase_3：

打断点运行：

```
Dump of assembler code for function _Z7phase_3Pc:
=> 0x0000555555555b74 <+0>:     endbr64
   0x0000555555555b78 <+4>:     sub    $0x28,%rsp
   0x0000555555555b7c <+8>:     lea    0x4b7(%rip),%rsi        # 0x55555555603a
   0x0000555555555b83 <+15>:    mov    %fs:0x28,%rax
   0x0000555555555b8c <+24>:    mov    %rax,0x18(%rsp)
   0x0000555555555b91 <+29>:    xor    %eax,%eax
   0x0000555555555b93 <+31>:    lea    0xf(%rsp),%rcx
   0x0000555555555b98 <+36>:    lea    0x10(%rsp),%rdx
   0x0000555555555b9d <+41>:    lea    0x14(%rsp),%r8
   0x0000555555555ba2 <+46>:    call   0x555555555170 <__isoc99_sscanf@plt>
   0x0000555555555ba7 <+51>:    cmp    $0x3,%eax
   0x0000555555555baa <+54>:    jne    0x555555555c11 <_Z7phase_3Pc+157>
   0x0000555555555bac <+56>:    cmpl   $0x7,0x10(%rsp)
   0x0000555555555bb1 <+61>:    ja     0x555555555c11 <_Z7phase_3Pc+157>
   0x0000555555555bb3 <+63>:    mov    0x10(%rsp),%eax
   0x0000555555555bb7 <+67>:    lea    0x5f6(%rip),%rdx        # 0x5555555561b4
   0x0000555555555bbe <+74>:    movslq (%rdx,%rax,4),%rax
   0x0000555555555bc2 <+78>:    add    %rdx,%rax
   0x0000555555555bc5 <+81>:    notrack jmp *%rax
   0x0000555555555bc8 <+84>:    cmpl   $0x30,0x14(%rsp)
   0x0000555555555bcd <+89>:    mov    $0x70,%al
   0x0000555555555bcf <+91>:    je     0x555555555c16 <_Z7phase_3Pc+162>
   0x0000555555555bd1 <+93>:    jmp    0x555555555c11 <_Z7phase_3Pc+157>
   0x0000555555555bd3 <+95>:    cmpl   $0xdd,0x14(%rsp)
   0x0000555555555bdb <+103>:   mov    $0x62,%al
   0x0000555555555bdd <+105>:   je     0x555555555c16 <_Z7phase_3Pc+162>
   0x0000555555555bdf <+107>:   jmp    0x555555555c11 <_Z7phase_3Pc+157>
   0x0000555555555be1 <+109>:   cmpl   $0x2f0,0x14(%rsp)
   0x0000555555555be9 <+117>:   mov    $0x63,%al
   0x0000555555555beb <+119>:   je     0x555555555c16 <_Z7phase_3Pc+162>
   0x0000555555555bed <+121>:   jmp    0x555555555c11 <_Z7phase_3Pc+157>
   0x0000555555555bef <+123>:   cmpl   $0x10,0x14(%rsp)
   0x0000555555555bf4 <+128>:   mov    $0x74,%al
   0x0000555555555bf6 <+130>:   je     0x555555555c16 <_Z7phase_3Pc+162>
   0x0000555555555bf8 <+132>:   jmp    0x555555555c11 <_Z7phase_3Pc+157>
   0x0000555555555bfa <+134>:   cmpl   $0x3,0x14(%rsp)
   0x0000555555555bff <+139>:   mov    $0x76,%al
   0x0000555555555c01 <+141>:   je     0x555555555c16 <_Z7phase_3Pc+162>
   0x0000555555555c03 <+143>:   jmp    0x555555555c11 <_Z7phase_3Pc+157>
   0x0000555555555c05 <+145>:   cmpl   $0x200,0x14(%rsp)
   0x0000555555555c0d <+153>:   mov    $0x77,%al
   0x0000555555555c0f <+155>:   je     0x555555555c16 <_Z7phase_3Pc+162>
   0x0000555555555c11 <+157>:   call   0x555555555a46 <_Z12explode_bombv>
```

发现调用了标准库的 scanf 函数。再增加断点，运行到+46 的位置，检查寄存器的值。

```
Breakpoint 12, 0x0000555555555ba2 in phase_3(char*) ()
(gdb) i r
rax            0x0                 0
rbx            0x0                 0
rcx            0x7fffffffdacf      140737488345807
rdx            0x7fffffffdad0      140737488345808
rsi            0x55555555603a      93824992239674
rdi            0x7fffffffdb08      140737488345864
rbp            0x7fffffffdb08      0x7fffffffdb08
rsp            0x7fffffffdac0      0x7fffffffdac0
r8             0x7fffffffdad4      140737488345812
r9             0x0                 0
r10            0x7ffff7d21ac0      140737351129792
r11            0x246               582
r12            0x7fffffffdaf0      140737488345840
r13            0x555555555200      93824992236032
r14            0x0                 0
r15            0x7ffff7ffd040      140737354125376
rip            0x555555555ba2      0x555555555ba2 <phase_3(char*)+46>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
k0             0x20000000          536870912
k1             0x0                 0
k2             0x7ff07ff           134154239
k3             0x0                 0
k4             0x0                 0
k5             0x0                 0
k6             0x0                 0
k7             0x0                 0
(gdb) x/s $rsi
0x55555555603a: "%d %c %d"
(gdb)
```

发现 rsi 的值是%d %c %d，说明要读入一个整数，一个字符，一个整数。

```
(gdb) r
Starting program: /mnt/d/my_project/ICS/lab2/bomb y
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
You have 6 phases with which to blow yourself up. Have a nice day!
each line is important
1 -3 9 -27 81 -243
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
5 c 9

Breakpoint 13, 0x0000555555555baa in phase_3(char*) ()
(gdb) x/d $rsp+0x10
0x7fffffffdad0: 5
(gdb) x/c $rsp+0xf
0x7fffffffdacf: 99 'c'
(gdb) x/d $rsp+0x14
0x7fffffffdad4: 9
(gdb)
```

重新按照要求运行，发现 0x10（%rsp）存放第一个输入，0xf（%rsp）第二个，0x14（%rsp）第三个。

首先+56 要求第一个数不大于 7。然后根据下面的结构推测是 switch-case 结构，并且只有当执行+145 的分支的时候才不会引爆炸弹。经测试，当第一个输入为 7 时会执行该分支。

```
  0x0000555555555c05 <+145>:   cmpl   $0x200,0x14(%rsp)
  0x0000555555555c0d <+153>:   mov    $0x77,%al
  0x0000555555555c0f <+155>:   je     0x555555555c16 <_Z7phase_3Pc+162>
  0x0000555555555c11 <+157>:   call   0x555555555a46 <_Z12explode_bombv>
-Type <RET> for more, q to quit, c to continue without paging--
  0x0000555555555c16 <+162>:   cmp    %al,0xf(%rsp)
  0x0000555555555c1a <+166>:   jne    0x555555555c11 <_Z7phase_3Pc+157>
  0x0000555555555c1c <+168>:   mov    0x18(%rsp),%rax
  0x0000555555555c21 <+173>:   xor    %fs:0x28,%rax
  0x0000555555555c2a <+182>:   je     0x555555555c31 <_Z7phase_3Pc+189>
  0x0000555555555c2c <+184>:   call   0x5555555551a0 <__stack_chk_fail@plt>
  0x0000555555555c31 <+189>:   add    $0x28,%rsp
  0x0000555555555c35 <+193>:   ret
```

然后该分支要求 0x14（%rsp），即第三个输出为 0x200，即 512 时才不会引爆炸弹。然后根据要求跳到+162，此步骤要求 0xf（%rsp），即第二个输入为 acsii 码对应为%al 存放的值，即 0x77 的字符才不会爆炸。于是确定三个输入的值，运行后通过。

```
Breakpoint 13, 0x0000555555555baa in phase_3(char*) ()
(gdb) c
Continuing.
Halfway there!
```

phase_4：

断点运行，检查寄存器的值后发现 rax 和 rdi 都是输入的值。

```
Dump of assembler code for function _Z7phase_4l:
=> 0x0000555555555c36 <+0>:     endbr64
   0x0000555555555c3a <+4>:     mov    %rdi,%rax
   0x0000555555555c3d <+7>:     sar    $0x20,%rdi
   0x0000555555555c41 <+11>:    push   %rdx
   0x0000555555555c42 <+12>:    lea    -0x1(%rdi),%edx
   0x0000555555555c45 <+15>:    cmp    $0xd,%edx
   0x0000555555555c48 <+18>:    ja     0x555555555c51 <_Z7phase_4l+27>
   0x0000555555555c4a <+20>:    dec    %eax
   0x0000555555555c4c <+22>:    cmp    $0xd,%eax
   0x0000555555555c4f <+25>:    jbe    0x555555555c56 <_Z7phase_4l+32>
   0x0000555555555c51 <+27>:    call   0x555555555a46 <_Z12explode_bomb
v>
   0x0000555555555c56 <+32>:    call   0x555555555548 <_ZL4hopei>
   0x0000555555555c5b <+37>:    cmp    $0x1000000,%eax
   0x0000555555555c60 <+42>:    jne    0x555555555c51 <_Z7phase_4l+27>
   0x0000555555555c62 <+44>:    pop    %rax
   0x0000555555555c63 <+45>:    ret
End of assembler dump.
```

注意到有一个右移的操作，起初以为是将 0x20 右移 rdi 位，始终不对，后面在输入负数后发现，其实是把 rdi 右移 0x20，即 32 位。因此，右移结束后 rdi 应该存放输入的左 32 位。然后 edx 被 rdi 减一赋值，要求 edx 不大于 0xd，故输入的左 32 位不能大于 0xe。

接着注意到出现了 eax，即 rax 的右 32 位，自减 1 后不大于 0xd，即输入的右 32 位不大于 e。

接着发现调用了一个 hope 函数，断点运行：

```
Dump of assembler code for function _ZL4hopei:
=> 0x0000555555555548 <+0>:     mov    $0x1,%r8d
   0x000055555555554e <+6>:     test   %edi,%edi
   0x0000555555555550 <+8>:     je     0x555555555575 <_ZL4hopei+45>
   0x0000555555555552 <+10>:    push   %rbx
   0x0000555555555553 <+11>:    mov    %edi,%ebx
   0x0000555555555555 <+13>:    sar    %edi
   0x0000555555555557 <+15>:    call   0x555555555548 <_ZL4hopei>
   0x000055555555555c <+20>:    mov    %eax,%r8d
   0x000055555555555f <+23>:    imul   %eax,%r8d
   0x0000555555555563 <+27>:    and    $0x1,%bl
   0x0000555555555566 <+30>:    je     0x555555555570 <_ZL4hopei+40>
   0x0000555555555568 <+32>:    lea    0x0(,%r8,4),%r8d
   0x0000555555555570 <+40>:    mov    %r8d,%eax
   0x0000555555555573 <+43>:    pop    %rbx
   0x0000555555555574 <+44>:    ret
   0x0000555555555575 <+45>:    mov    %r8d,%eax
   0x0000555555555578 <+48>:    ret
End of assembler dump.
```

先给 r8d 赋值为 1，接着判断 exi，即输入的左 32 位是否为 0，不为 0 就 edi 右移 1 位，继续调用 hope 函数。发现这是个递归的函数。递归结束的标志是 edi 为 0。每次递归都将 r8d 左移 2 位。经测试，将 edi 初始为 8 时，rdi 会左移 8 位；将 edi 初始为 4 时，rdi 会左移 4 位。在 phase_4 函数中，结束调用 hope 函数后，发现有一个判断 hope 调用情况的语句，要求 r8d 必须被左移 12 次，于是将 edi 初始化为 0xc，即输入的左 32 位为 0xc，同时右 32 位在 0x1-0xe 中随便取，运行后发现通过样例：

```
(gdb) r
Starting program: /mnt/d/my_project/ICS/lab2/bomb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1"
.
You have 6 phases with which to blow yourself up. Have a nice day!
each line is important
1 -3 9 -27 81 -243
7 w 512
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
51539607566
So you got that one.  Try this one.
```

```
(gdb) r
Starting program: /mnt/d/my_project/ICS/lab2/bomb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1"
.
You have 6 phases with which to blow yourself up. Have a nice day!
each line is important
1 -3 9 -27 81 -243
7 w 512
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
51539607553
So you got that one.  Try this one.
```

phase_5：

断点 phase5，输入运行。发现 phase_5 参数为 3 个 long long int，于是重新运行，输入 3 个整数 666 777 888。发现寄存器 rdi，rsi，rdx 存入输入的三个数：

```
multi-thre Thread 0x7ffff7a577 In: phase_5
rax            0xffffffff00000000  -4294967296
rbx            0x0                 0
rcx            0x555555557ce8      93824992247016
rdx            0x378               888
rsi            0x309               777
rdi            0x29a               666
rbp            0x7fffffffdb08      0x7fffffffdb08
rsp            0x7fffffffdaa0      0x7fffffffdaa0
r8             0xffffffff          4294967295
r9             0x555555558670      93824992249456
r10            0x7ffff7da4588      140737351665032
r11            0x246               582
r12            0x7fffffffdaf0      140737488345840
r13            0x555555555200      93824992236032
--Type <RET> for more, q to quit, c to continue without paging--
```

接着继续运行程序，发现在运行到 baselock:: release 的部分时，会将寄存器 ebp 与 0xff0 比较，如果不相等就会爆炸，而此时 ebx 存入的是第三个输入：

```
0x00005555555555e1a in baselock::is_holding(int) ()
0x00005555555555e1e in baselock::is_holding(int) ()
0x00005555555555e21 in baselock::is_holding(int) ()
0x00005555555555e23 in baselock::is_holding(int) ()
0x00005555555555e25 in baselock::is_holding(int) ()
0x00005555555555e28 in baselock::is_holding(int) ()
0x00005555555555e2b in baselock::is_holding(int) ()
0x00005555555555e2d in baselock::is_holding(int) ()
0x00005555555554bc in baselock::release(int, int) ()
0x00005555555554c2 in baselock::release(int, int) ()
(gdb) i r ebx
ebx            0xffffdaa8          -9560
(gdb) i r ebp
ebp            0x378               888
(gdb)
```

于是猜测第三个数为 0xff0，即 4080。重新运行后发现直接通过了。

开始检查第一和第二个数的条件。发现对于第一个数输入不同的值，对于第三个数有不同的结果影响。

在 phase_5 中发现：

当第一个数小于等于 1 时，第三个数应该是 0xf，即 15：

```
> 0x555555555528 <_ZN5lock27releaseEii+18>      cmp    $0xf,%ebp
  0x55555555552b <_ZN5lock27releaseEii+21>      jne    0x555555555542 <_ZN5lock27releaseEii+44>
  0x55555555552d <_ZN5lock27releaseEii+23>      dec    %al
  0x55555555552f <_ZN5lock27releaseEii+25>      jne    0x555555555542 <_ZN5lock27releaseEii+44>
  0x555555555531 <_ZN5lock27releaseEii+27>      mov    $0xffffffff,%eax
```

当大于 1 小于等于 3 时，第三个数应该是 0xf00，即 3840：

```
  0x5555555554ef <_ZN5lock17releaseEii+15>      call   *0x10(%rax)
> 0x5555555554f2 <_ZN5lock17releaseEii+18>      cmp    $0xf00,%ebp
  0x5555555554f8 <_ZN5lock17releaseEii+24>      jne    0x55555555550f <_ZN5lock17releaseEii+47>
  0x5555555554fa <_ZN5lock17releaseEii+26>      dec    %al
  0x5555555554fc <_ZN5lock17releaseEii+28>      jne    0x55555555550f <_ZN5lock17releaseEii+47>
  0x5555555554fe <_ZN5lock17releaseEii+30>      mov    $0xffffffff,%eax
```

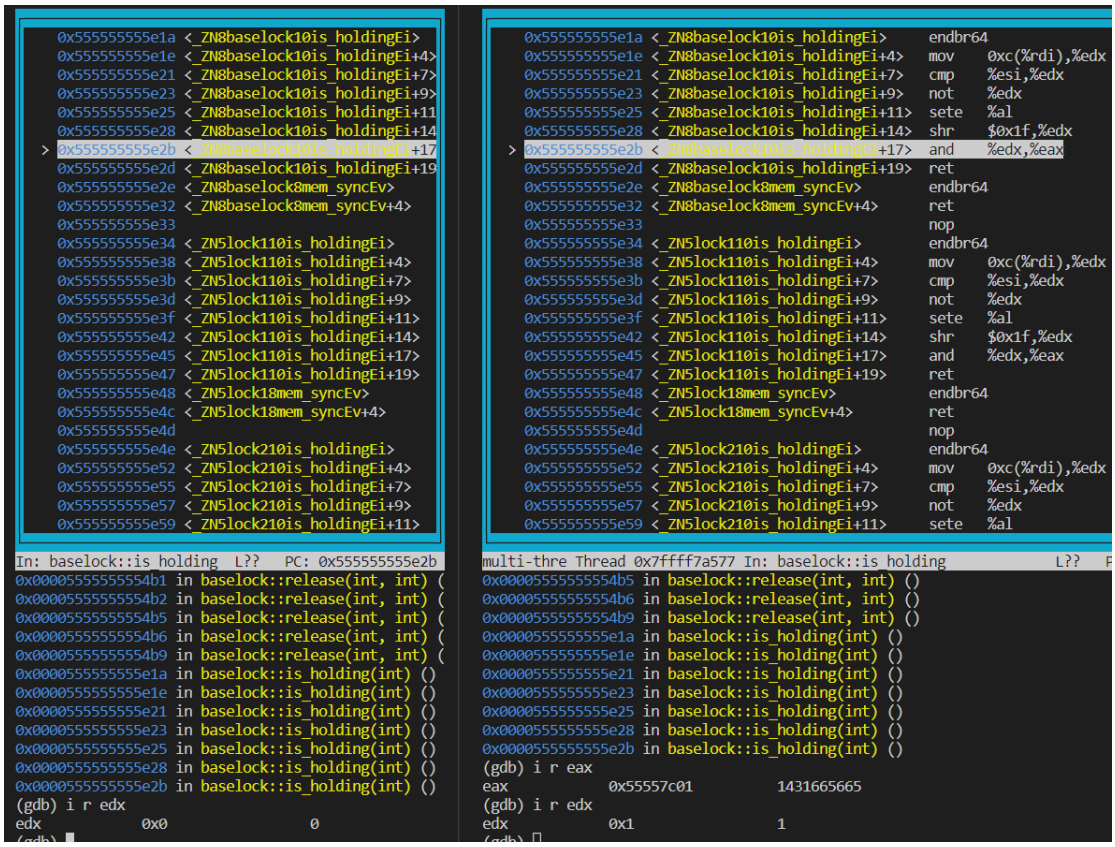
当大于 3 时，第三个数应该是 0xff0，即 4080：

```
> 0x5555555554bc <_ZN8baselock7releaseEii+18>   cmp    $0xff0,%ebp
  0x5555555554c2 <_ZN8baselock7releaseEii+24>   jne    0x5555555554d9 <_ZN8baselock7releaseEii+47>
  0x5555555554c4 <_ZN8baselock7releaseEii+26>   dec    %al
  0x5555555554c6 <_ZN8baselock7releaseEii+28>   jne    0x5555555554d9 <_ZN8baselock7releaseEii+47>
  0x5555555554c8 <_ZN8baselock7releaseEii+30>   mov    $0xffffffff,%eax
  0x5555555554cd <_ZN8baselock7releaseEii+35>   shl    $0x20,%rax
```

当等于 8 时，与上面一种情况类似，也是 4080，但是会多执行一步 movq 的操作：

```
0x555555555cd6 <_Z7phase_5lll+114>      lea    0x8(%rsp),%rdi
0x555555555cdb <_Z7phase_5lll+119>      jne    0x555555555ceb <_Z7phase_5lll+135>
0x555555555cdd <_Z7phase_5lll+121>      call   0x555555555a93 <_Z13run_lock_testP8baselockii>
0x555555555ce2 <_Z7phase_5lll+126>      movb   $0x79,0x234f(%rip)        # 0x555555558038 <phase_2_nums+24>
0x555555555ce9 <_Z7phase_5lll+133>      jmp    0x555555555cf0 <_Z7phase_5lll+140>
0x555555555ceb <_Z7phase_5lll+135>      call   0x555555555a93 <_Z13run_lock_testP8baselockii>
```

对于第二个数，发现在 block:: is_holding 函数中会取第二个输入的值的符号，即最高位，只有当最高位为 0，即第二个数非负的时候，才满足条件：



运行最终通过：

```
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /mnt/d/my_project/ICS/lab2/bomb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
You have 6 phases with which to blow yourself up. Have a nice day!
each line is important
1 -3 9 -27 81 -243
7 w 512
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
51539607566
1 2 15
So you got that one.  Try this one.
Good work!  On to the next...
```

phase_6：

断点运行，发现调用了 string_len 函数，猜测是求字符串长度，并且在之后会与 6 比较，猜测需要输入一个 6 位字符串：

```
0x0000000000001d0f <+5>:     call   0x1a7c <_Z10string_lenPc>
0x0000000000001d14 <+10>:    lea    0x27f0(%rip),%rdx        # 0x450b <w2+11>
0x0000000000001d1b <+17>:    mov    %eax,%r8d
0x0000000000001d1e <+20>:    xor    %eax,%eax
0x0000000000001d20 <+22>:    cmp    $0x6,%r8d
0x0000000000001d24 <+26>:    je     0x1d2b <_Z7phase_6Pc+33>
0x0000000000001d26 <+28>:    call   0x1a46 <_Z12explode_bombv>
```

输入字符串，继续运行，发现 rdi 中放着一个算式加输入字符串，猜测应该输入算式：

```
(gdb) x/s $rdi
0x555555558500 <w2>:    "(1+2)*(9-0)abcdef"
(gdb)
```

之后继续运行，输入*（1+1）

在进入 compare_answer_and_candidate 函数后，发现从内存中加载了 2 个值到寄存器，并且发现这是个循环：

```
   0x0000000000001936 <+20>:    lea    0x2a43(%rip),%rcx        # 0x438
0 <ans>
   0x000000000000193d <+27>:    lea    0x2e3c(%rip),%rsi        # 0x478
0 <cand>
   0x0000000000001944 <+34>:    xor    %edi,%edi
   0x0000000000001946 <+36>:    cmp    %edi,%edx
   0x0000000000001948 <+38>:    jl     0x197b <_Z28compare_answer_and_c
```

并且每次循环中，rcx 和 rsi 的值分别会自增 12，edi 会自增 1。于是猜测 rcx 和 rsi 是输入的答案和谜底。于是遍历 2 者，发现 rsi 存放输入的被处理过的表达式，rcx 存放谜底：

```
(gdb) x/s $rsi+0
0x555555558780 <cand>:  ""
(gdb) x/s $rsi+12
0x55555555878c <cand+12>:       "+"
(gdb) x/s $rsi+24
0x555555558798 <cand+24>:       "1"
(gdb) x/s $rsi+36
0x5555555587a4 <cand+36>:       "1"
(gdb) x/s $rsi+48
0x5555555587b0 <cand+48>:       "-"
(gdb) x/s $rsi+60
0x5555555587bc <cand+60>:       "9"
(gdb) x/s $rsi+72
0x5555555587c8 <cand+72>:       "0"
(gdb) x/s $rsi+84
0x5555555587d4 <cand+84>:       "*"
(gdb) 96
Undefined command: "96".  Try "help".
(gdb) x/s $rsi+96
0x5555555587e0 <cand+96>:       "+"
(gdb) x/s $rsi+108
0x5555555587ec <cand+108>:      "1"
(gdb) x/s $rsi+120
0x5555555587f8 <cand+120>:      "2"
(gdb) x/s $rsi+134
0x555555558806 <cand+134>:      ""
(gdb)
```

```
(gdb) x/s $rcx+0
0x555555558380 <ans>:   ""
(gdb) x/s $rcx+12
0x55555555838c <ans+12>:        "-"
(gdb) x/s $rcx+24
0x555555558398 <ans+24>:        "3"
(gdb) x/s $rcx+36
0x5555555583a4 <ans+36>:        "4"
(gdb) x/s $rcx+48
0x5555555583b0 <ans+48>:        "-"
(gdb) x/s $rcx+60
0x5555555583bc <ans+60>:        "9"
(gdb) x/s $rcx+72
0x5555555583c8 <ans+72>:        "0"
(gdb) x/s $rcx+84
0x5555555583d4 <ans+84>:        "/"
(gdb) x/s $rcx+96
0x5555555583e0 <ans+96>:        "+"
(gdb) x/s $rcx+108
0x5555555583ec <ans+108>:       "1"
(gdb) x/s $rcx+120
0x5555555583f8 <ans+120>:       "2"
(gdb) x/s $rcx+134
0x555555558406 <ans+134>:       ""
(gdb)
```

于是猜测答案应该是/（3-4），验证后通过，提示进行隐藏关。

```
(gdb) r <password.txt
Starting program: /mnt/d/my_project/ICS/lab2/bomb <password.txt
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1"
.
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...
But isn't something... missing? Perhaps something was overlooked?
[Inferior 1 (process 14657) exited normally]
(gdb)
```

secret_phase：

注意到 phase_5 中有个神奇的地方，如果第一个输入是 8，就会干一件意义不明的事情，猜测为开启隐藏关卡的钥匙。修改输入后发现成功进入浮点世界：

```
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...
You've enter the float point world! It's not hard o(*^ _ ^*)m
```

输入 123，测试。先是一个格式转换，接着打印 xmm0 寄存器，在 float 格式中发现输入的数据：

```
(gdb) p/f $xmm0
$4 = {v8_bfloat16 = {0, 123, 1.464e+13, 0, -96, 1.471e+13, 1.464e+13, 0}, v8_half = {0,
    3.4805, 85.312, 0, -3.375, 85.375, 85.312, 0}, v4_float = {123, 3.0611365e-41,
    1.47582458e+13, 3.0611365e-41}, v2_double = {4.6355553095931429e-310,
    4.635570543178489e-310}, v16_int8 = {0, 0, -10, 66, 85, 85, 0, 0, -64, -62, 86, 85,
    85, 85, 0, 0}, v8_int16 = {0, 17142, 21845, 0, -15680, 21846, 21845, 0}, v4_int32 = {
    123, 3.0611365e-41, 1.47582458e+13, 3.0611365e-41}, v2_int64 = {
    4.6355553095931429e-310, 4.635570543178489e-310}, uint128 = <invalid float value>}
```

之后输入的数会乘以 2，然后加上一个数，扩充为双精度，再次打印 xmm0，发现变成 256，相当于+10，：

```
(gdb) p/f $xmm0
$7 = {v8_bfloat16 = {0, 0, 0, 3.75, -96, 1.471e+13, 1.464e+13, 0}, v8_half = {0, 0, 0,
    2.2188, -3.375, 85.375, 85.312, 0}, v4_float = {0, 3.75, 1.47582458e+13,
    3.0611365e-41}, v2_double = {256, 4.635570543178489e-310}, v16_int8 = {0, 0, 0, 0,
    0, 0, 112, 64, -64, -62, 86, 85, 85, 85, 0, 0}, v8_int16 = {0, 0, 0, 16496, -15680,
    21846, 21845, 0}, v4_int32 = {0, 3.75, 1.47582458e+13, 3.0611365e-41}, v2_int64 = {
    256, 4.635570543178489e-310}, uint128 = <invalid float value>}
```

接着，接着发现和 rip+0x411 中浮点数进行比较，要求小于等于某个数：

```
Dump of assembler code for function _Z12secret_phasel:
   0x0000000000001d66 <+0>:     endbr64
   0x0000000000001d6a <+4>:     cvtsi2ss %rdi,%xmm0
   0x0000000000001d6f <+9>:     addss  %xmm0,%xmm0
   0x0000000000001d73 <+13>:    addss  0x475(%rip),%xmm0        # 0x21f0
   0x0000000000001d7b <+21>:    cvtss2sd %xmm0,%xmm0
   0x0000000000001d7f <+25>:    comisd 0x471(%rip),%xmm0        # 0x21f8
   0x0000000000001d87 <+33>:    jae    0x1d97 <_Z12secret_phasel+49>
   0x0000000000001d89 <+35>:    movsd  0x46f(%rip),%xmm1        # 0x2200
   0x0000000000001d91 <+43>:    comisd %xmm0,%xmm1
   0x0000000000001d95 <+47>:    jb     0x1d9d <_Z12secret_phasel+55>
   0x0000000000001d97 <+49>:    push   %rax
   0x0000000000001d98 <+50>:    call   0x1a46 <_Z12explode_bombv>
   0x0000000000001d9d <+55>:    ret
End of assembler dump.
```

暂时不知道取值范围，但是 123 正常通过了。然后 xmm1 的 double 加载了一个值，打印出来为 3819.9999990000001：

```
(gdb) p/lf $xmm1
$16 = {v8_bfloat16 = {2.872e+30, -nan(0x5e), -5.608e+14, 5.406, 0, 0, 0, 0}, v8_half = {
    12424, -nan(0x3de), -127.94, 2.3379, 0, 0, 0, 0}, v4_float = {-nan(0x5e7211),
    5.43261671, 0, 0}, v2_double = {3819.9999990000001, 0}, v16_int8 = {17, 114, -34,
    -1, -1, -41, -83, 64, 0, 0, 0, 0, 0, 0, 0, 0}, v8_int16 = {29201, -34, -10241,
    16557, 0, 0, 0, 0}, v4_int32 = {-nan(0x5e7211), 5.43261671, 0, 0}, v2_int64 = {
    3819.9999990000001, 0}, uint128 = 1.698888504486317949 96e-4932}
(gdb)
```

然后要求之前得到的值要大于它，当输入 1905，得到 3820，是第一个大于它的整数，发现 1905 可以通过样例，并且 1904、1906 均不行。于是猜测之前那个比较是要求输入小于等于 1905：

```
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
You've enter the float point world! It's not hard o(*^ _ ^*)m

Breakpoint 5, 0x0000555555555d66 in secret_phase(long) ()
(gdb) c
Continuing.
Congratulations!
```