BombLab
实验报告

姓名：黄昊

学号：21300240011

Phase_1

```
   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>:    callq  0x555555555a60 <_Z16string_not_equalPcS_>
   0x0000555555555b2d <+27>:    test   %al,%al
   0x0000555555555b2f <+29>:    jne    0x555555555b36 <_Z7phase_1Pc+36>
   0x0000555555555b31 <+31>:    callq  0x555555555a46 <_Z12explode_bombv>
   0x0000555555555b36 <+36>:    pop    %rax
   0x0000555555555b37 <+37>:    retq
End of assembler dump.
(gdb) si
0x0000555555555b28 in phase_1(char*) ()
(gdb) i r rsi
rsi            0x555555558146      93824992248134
(gdb) x/s 0x555555558146
0x555555558146 <w1+230>:        "each line is important"
```

进入 string_not_equal 函数之前，给 rsi 赋值为“each line is important”，在 string_not_equal 函数中逐个字符比较输入和 rsi 是否相同，相同则返回 1，即拆除炸弹，返回 0 则会跳到炸弹引爆函数

Phase_2

```
   0x0000555555555b3c <+4>:     push   %rdx
=> 0x0000555555555b3d <+5>:     lea    0x24dc(%rip),%rsi        # 0x555555558020 <phase_2_nums>
   0x0000555555555b44 <+12>:    callq  0x555555555ad0 <_Z16read_six_numbersPcPi>
   0x0000555555555b49 <+17>:    lea    0x24d0(%rip),%rax        # 0x555555558020 <phase_2_nums>
```

这里调用函数 read_six_numbers 可以推测输入为 6 个数字

```
   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>:    callq  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>:    retq
End of assembler dump.
(gdb) i r esi
esi            0x2                 2
(gdb) i r ecx
ecx            0xfffffffd          -3
(gdb)
```

然后给寄存器 esi 赋值为第一个输入的数字，同时给 ecx 赋值为-3，将两寄存器的值相乘保存在 esi 中。接着判断相乘结果和下一个输入是否相等，不相等则炸弹引爆，相等就跳转到 49 行。这里将 rax 中的内存加 4 个 byte 并和 rdx 比较，然后跳转，可以看出是一个循环，每次循环都把 esi（-3）乘 ecx（rax 的内存）并判断是否和下一个输入相等。所以输入的六个数只要满足公比为-3 即可。

Phase_3

```
(gdb) si 3
0x0000555555555b83 in phase_3(char*) ()
(gdb) x/s 0x55555555603a
0x55555555603a: "%d %c %d"
(gdb)
```

查看 rsi 中地址内容可以推测出输入的类型

```
0x0000555555555ba2 <+46>:    callq  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>
```

```
(gdb) i r rsp
rsp            0x7fffffffdc20      0x7fffffffdc20
(gdb) x/s 0x7fffffffdc30
0x7fffffffdc30: "\004"
(gdb)
```

接着判断输入是否为 3 个，通过查看 rsp 地址，并查看 0x10（%rsp）中的内容可以看出是第一个输入，因此第一个输入要小于 7；通过尝试输入 1~7 得到不同输入跳转到的不同语句：

1 -> 95

2 -> 109

3,4,5 -> 123

6 -> 134

7 -> 145

这里第一个输入为 4，跳转到 123 行

```
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>
```

跳转后比较 0x14(%rsp)（第三个输入）和 0x10 是否相等，因此第三个输入要和 cmpl 前面的立即数相等，这里是十进制的 16，然后给 al 赋值 0x74

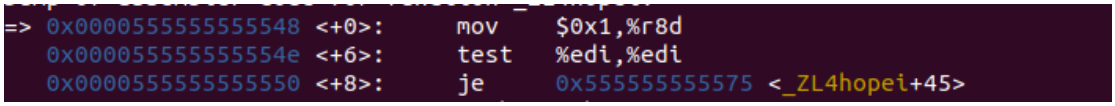
```
=> 0x0000555555555c16 <+162>:   cmp    %al,0xf(%rsp)
   0x0000555555555c1a <+166>:   jne    0x555555555c11 <_Z7phase_3Pc+157>
   0x0000555555555c1c <+168>:   mov    0x18(%rsp),%rax
```

跳转到 162 行后又将 al 的值和 0xf(%rsp)（第二个输入）比较是否相等，第二个输入为字符，通过查 ASCII 码表 0x74 对应的是 t，所以得到第三个输入为字符 t

Phase_4

```
   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>:    callq  0x555555555a46 <_Z12explode_bombv>
   0x0000555555555c56 <+32>:    callq  0x555555555548 <_ZL4hopei>
   0x0000555555555c5b <+37>:    cmp    $0x1000000,%eax
   0x0000555555555c60 <+42>:    jne    0x555555555c51 <_Z7phase_4l+27>
   0x0000555555555c62 <+44>:    pop    %rax
   0x0000555555555c63 <+45>:    retq
End of assembler dump.
(gdb) i r eax
eax            0x8                 8
(gdb) i r edx
edx            0xb                 11
```

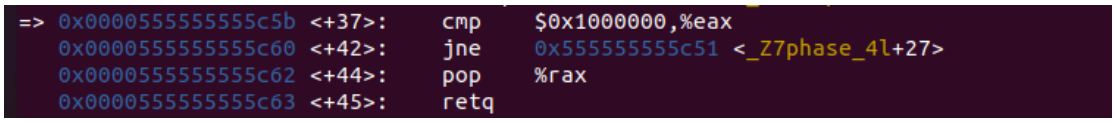
首先把输入赋给 rax 然后把输入的数算术右移 32 位，接着两个比较，分别比较前 32 位的值-1 和后 32 位的值-1 是否小于等于 13，不是则炸弹引爆。接着进入递归函数 hope



首先是递归结束条件判断输入（rdi）是否为 0，为 0 则返回 r8d（1）

每次递归都用 ebx 保存当前输入的值，然后把输入算术右移 1 位，即除以 2，然后继续递归调用，直到返回 1.

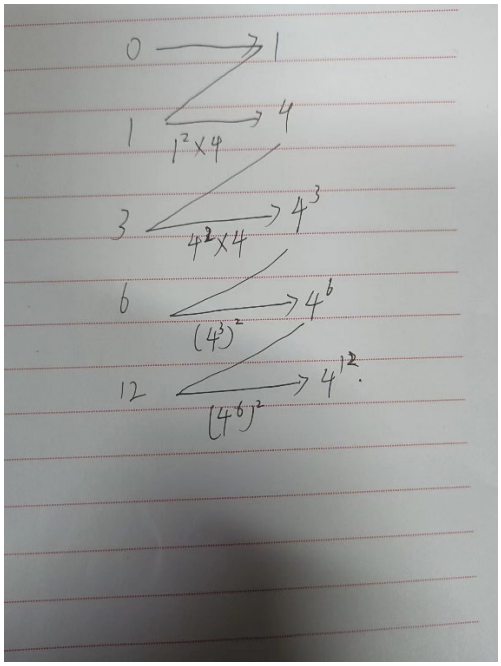
递归返回后每次都会将返回的结果进行平方，并且判断当前的输入为奇数还是偶数，奇数则将要返回的结果乘 4；



最后将递归返回的结果和 0x1000000（4^{12}）比较是否相等。

因为递归函数的输入为输入值的前 32 位，通过倒退可以知道前 32 位对应的十进制值应为 12，所以最后结果只要保证 64 位二进制数的前 32 位是十进制的 12，后 32 位的十进制数小于 13，我的输入为：

51539607560=1100_0000_0000_0000_0000_0000_0000_1000



推测递归函数 hope 的作用是计算 4 的幂次

Phase_5

首先将父类和两个子类的 acquire 函数的地址压入程序栈中；

```
0x0000555555555c6c <+8>:     lea    0x2045(%rip),%rcx        # 0x555555557cb8 <_ZTV5lock1+16>
0x0000555555555c73 <+15>:    mov    %fs:0x28,%rax
0x0000555555555c7c <+24>:    mov    %rax,0x38(%rsp)
0x0000555555555c81 <+29>:    xor    %eax,%eax
0x0000555555555c83 <+31>:    lea    0x1ffe(%rip),%rax        # 0x555555557c88 <_ZTV8baselock+16>
0x0000555555555c8a <+38>:    mov    %rcx,0x18(%rsp)
0x0000555555555c8f <+43>:    lea    0x2052(%rip),%rcx        # 0x555555557ce8 <_ZTV5lock2+16>
0x0000555555555c96 <+50>:    mov    %rax,0x8(%rsp)
```

在分别将数 0xffffffff00000000 压入每个 acquire 函数的地址+0xc；

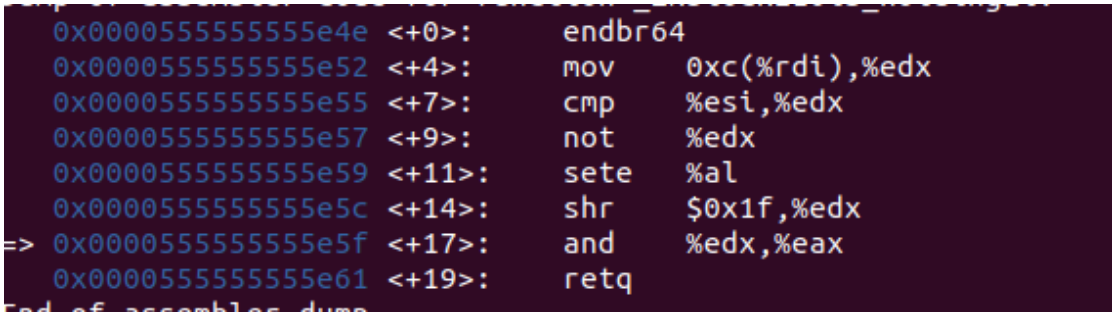
```
0x0000555555555c9b <+55>:    mov    $0xffffffff,%eax
0x0000555555555ca0 <+60>:    shl    $0x20,%rax
0x0000555555555ca4 <+64>:    mov    %rcx,0x28(%rsp)
0x0000555555555ca9 <+69>:    mov    %rax,0x10(%rsp)
0x0000555555555cae <+74>:    mov    %rax,0x20(%rsp)
0x0000555555555cb3 <+79>:    mov    %rax,0x30(%rsp)
```

rdi 中存储的是输入的第一个变量，将输入分别和 1，3，8 比较大小选择要运行的 acquire 函数；

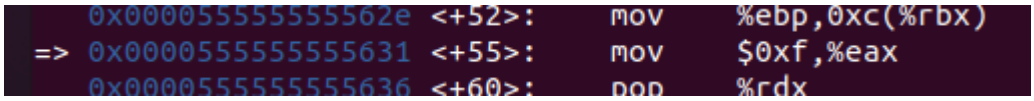
```
0x0000555555555cb8 <+84>:    cmp    $0x1,%rdi
0x0000555555555cbc <+88>:    jg     0x555555555cc5 <_Z7phase_5lll+97>
0x0000555555555cbe <+90>:    lea    0x28(%rsp),%rdi
0x0000555555555cc3 <+95>:    jmp    0x555555555ceb <_Z7phase_5lll+135>
0x0000555555555cc5 <+97>:    cmp    $0x3,%rdi
0x0000555555555cc9 <+101>:   jg     0x555555555cd2 <_Z7phase_5lll+110>
0x0000555555555ccb <+103>:   lea    0x18(%rsp),%rdi
0x0000555555555cd0 <+108>:   jmp    0x555555555ceb <_Z7phase_5lll+135>
0x0000555555555cd2 <+110>:   cmp    $0x8,%rdi
0x0000555555555cd6 <+114>:   lea    0x8(%rsp),%rdi
0x0000555555555cdb <+119>:   jne    0x555555555ceb <_Z7phase_5lll+135>
0x0000555555555cdd <+121>:   callq  0x555555555a93 <_Z13run_lock_testP8baselockii>
0x0000555555555ce2 <+126>:   movb   $0x79,0x234f(%rip)        # 0x555555558038 <phase_2_nums+24>
0x0000555555555ce9 <+133>:   jmp    0x555555555cf0 <_Z7phase_5lll+140>
0x0000555555555ceb <+135>:   callq  0x555555555a93 <_Z13run_lock_testP8baselockii>
```

进入 run_lock_test 函数后更具之前选择的地址进入相应的 acquire 函数，在 acquire 函数中

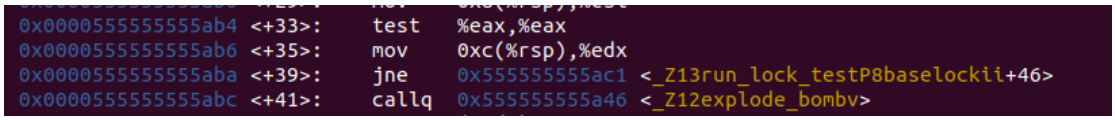
先会调用 is_holding 函数；在该函数中先给 edx 赋值为-1，然后比较 esi（第二个输入的后 32 位）和 edx 的大小，相等则 al = 1，否则 al = 0；再给 edx 取反变为 0，右移 31（0x1f）后仍为 0，和 eax 取与后为 0，即返回值为 0；



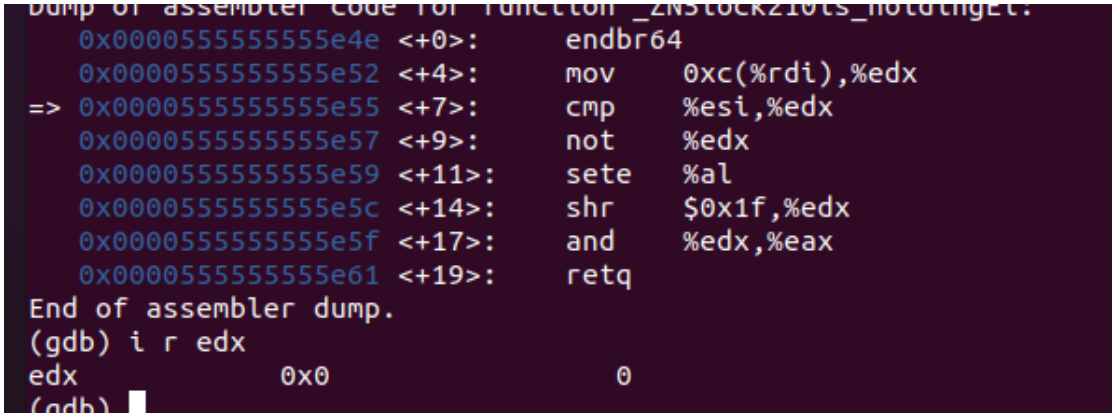
根据该返回值，acquire 函数则可返回 0xf（lock2）即为第三个输入；并且将栈顶 ebp（之前被调用函数保存的 rsi，即第二个输入）保存到 0xc(%rbx)，即下一次调用 is_holding 函数是赋给 edx 的值



第三个输入和在 acquire 函数中的返回值不相等会爆炸



相等就会调用 release 函数，并在 release 函数中再次调用 is_holding 函数，这时 edx 中会被赋值为第二个输入的后 32 位，并和第二个输入的后 32 位 esi 比较，结果一定相等，所以 al 会等于 1，并且 edx 取反等于 0xffffffff,逻辑右移 31 位后变成 1，再和 eax 取与后即可保留 al 的最后一位；



返回值 al = 1，第二个输入和 0xf 相等，并且 al-1=0，都不会触发 jne 指令跳转到+44 条指令，即让返回值为 0

然后让返回值为 0xffffffff00000001

```
=> 0x0000555555555547 <+49>:    retq
End of assembler dump.
(gdb) i r rax
rax            0xffffffff00000001  -4294967295
(gdb)
```

返回到 run_lock_test 函数后判断 al 是否为 0，不为 0 则拆弹成功

```
=> 0x00005555555555ac7 <+52>:    test   %al,%al
   0x00005555555555ac9 <+54>:    je     0x555555555abc <_Z13run_lock_testP8baselockii+41>
```

Phase_6

```
   0x00005555555555d0e <+4>:     push   %rdx
   0x00005555555555d0f <+5>:     callq  0x5555555555a7c <_Z10string_lenPc>
   0x00005555555555d14 <+10>:    lea    0x27f0(%rip),%rdx        # 0x55555555850b <w2+11>
   0x00005555555555d1b <+17>:    mov    %eax,%r8d
   0x00005555555555d1e <+20>:    xor    %eax,%eax
   0x00005555555555d20 <+22>:    cmp    $0x6,%r8d
   0x00005555555555d24 <+26>:    je     0x5555555555d2b <_Z7phase_6Pc+33>
   0x00005555555555d26 <+28>:    callq  0x5555555555a46 <_Z12explode_bombv>
```

首先从 string_len 函数和返回值与 6 比较判断输入的字符串长度一个为 6

```
   0x00005555555555d2b <+33>:    mov    (%rdi,%rax,1),%cl
   0x00005555555555d2e <+36>:    inc    %rax
   0x00005555555555d31 <+39>:    mov    %cl,-0x1(%rax,%rdx,1)
   0x00005555555555d35 <+43>:    cmp    $0x7,%rax
=> 0x00005555555555d39 <+47>:    jne    0x5555555555d2b <_Z7phase_6Pc+33>
   0x00005555555555d3b <+49>:    mov    $0x11,%esi
   0x00005555555555d40 <+54>:    lea    0x27b9(%rip),%rdi        # 0x555555558500 <w2>
   0x00005555555555d47 <+61>:    callq  0x5555555555765 <_Z31build_candidate_expression_treePci>
   0x00005555555555d4c <+66>:    mov    $0x11,%edx
   0x00005555555555d51 <+71>:    lea    0x26ac(%rip),%rdi        # 0x555555558404 <ans+132>
   0x00005555555555d58 <+78>:    mov    %rax,%rsi
   0x00005555555555d5b <+81>:    callq  0x5555555555922 <_Z28compare_answer_and_candidateP9tree_nodeS0_i>
   0x00005555555555d60 <+86>:    test   %al,%al
   0x00005555555555d62 <+88>:    je     0x5555555555d26 <_Z7phase_6Pc+28>
   0x00005555555555d64 <+90>:    pop    %rax
   0x00005555555555d65 <+91>:    retq
End of assembler dump.
(gdb) i r cl
cl             0x2f                47
(gdb) x/s 0x555555558500
0x555555558500 <w2>:    "(1+2)*(9-0)/"
```

```
(gdb) x/s 0x555555558500
0x555555558500 <w2>:    "(1+2)*(9-0)/(3-4)"
(gdb)
```

接着进入一个循环，一共循环 6 次，把输入的 6 个字符接在原来的表达式后面

```
> 0x0000555555555765 <+0>:     endbr64
  0x0000555555555769 <+4>:     push   %r14
  0x000055555555576b <+6>:     lea    0x31ef(%rip),%rdx        # 0x555555558961 <cand_stack+1>
  0x0000555555555772 <+13>:    push   %r13
  0x0000555555555774 <+15>:    push   %r12
  0x0000555555555776 <+17>:    push   %rbp
  0x0000555555555777 <+18>:    push   %rbx
  0x0000555555555778 <+19>:    sub    $0x10,%rsp
  0x000055555555577c <+23>:    mov    %fs:0x28,%rax
  0x0000555555555785 <+32>:    mov    %rax,0x8(%rsp)
  0x000055555555578a <+37>:    xor    %eax,%eax
  0x000055555555578c <+39>:    movl   $0x0,(%rsp)
  0x0000555555555793 <+46>:    movl   $0x0,0x4(%rsp)
  0x000055555555579b <+54>:    movl   $0x1,0x2d7b(%rip)        # 0x555555558520 <cand_node_idx>
```

进入 build_candidate_expression_tree 函数，rdx 保存 cand_stack 栈，(%rsp)记录 digit_stack 和 flag_stack 栈的元素个数，0x4(%rsp)记录 op_stack 栈中元素的个数，并给 cand_node_idx 赋初值 1；

```
   0x00005555555557a5 <+64>:    cmp    %eax,%esi
   0x00005555555557a7 <+66>:    jg     0x5555555557bf <_Z31build_candidate_expression_treePci+90>
   0x00005555555557a9 <+68>:    mov    $0x0,%ebx
   0x00005555555557ae <+73>:    test   %esi,%esi
   0x00005555555557b0 <+75>:    lea    0x31a9(%rip),%rbp        # 0x555555558960 <cand_stack>
   0x00005555555557b7 <+82>:    cmovs  %ebx,%esi
   0x00005555555557ba <+85>:    movslq %esi,%rbx
   0x00005555555557bd <+88>:    jmp    0x5555555557fc <_Z31build_candidate_expression_treePci+151>
=> 0x00005555555557bf <+90>:    mov    (%rdi,%rax,1),%cl
   0x00005555555557c2 <+93>:    inc    %rax
   0x00005555555557c5 <+96>:    mov    %cl,-0x1(%rax,%rdx,1)
   0x00005555555557c9 <+100>:   jmp    0x5555555557a5 <_Z31build_candidate_expression_treePci+64>
```

接着进入一个循环，把之前连接在一起的表达式依次压入 cand_stack 栈（rbx）中

```
   0x00005555555557cb <+102>:   mov    0x0(%rbp,%rbx,1),%r12b
   0x00005555555557d0 <+107>:   lea    -0x30(%r12),%eax
>  0x00005555555557d5 <+112>:   cmp    $0x9,%al
   0x00005555555557d7 <+114>:   ja     0x555555555808 <_Z31build_candidate_expression_treePci+163>
   0x00005555555557d9 <+116>:   mov    (%rsp),%eax
   0x00005555555557dc <+119>:   lea    0x315d(%rip),%rdx        # 0x555555558940 <digit_stack>
   0x00005555555557e3 <+126>:   inc    %eax
   0x00005555555557e5 <+128>:   mov    %eax,(%rsp)
   0x00005555555557e8 <+131>:   cltq
   0x00005555555557ea <+133>:   mov    %r12b,(%rdx,%rax,1)
   0x00005555555557ee <+137>:   lea    0x312b(%rip),%rdx        # 0x555555558920 <flag_stack>
   0x00005555555557f5 <+144>:   movb   $0x0,(%rdx,%rax,1)
   0x00005555555557f9 <+148>:   dec    %rbx
   0x00005555555557fc <+151>:   test   %ebx,%ebx
   0x00005555555557fe <+153>:   jg     0x5555555557cb <_Z31build_candidate_expression_treePci+102>
```

压栈后进入一个大循环，循环每次从 cand_stack 栈中取栈顶字符赋给 r12b，然后将 r12b-0x30（48）的值赋给 eax，最后将 al 的值和 0x9 比较；这一步是判断栈顶字符是数字还是操作符，查表得知运算符的 ASCII 码是 40~47，数字是 48~57,如果该字符是操作符，减去 48 后变成负数，按照无符号数的比较一定大于 9，则跳转到 163 行进行运算符的操作；如果是数字相减的结果就是对应的数字。

```
=> 0x00005555555557d9 <+116>:   mov    (%rsp),%eax
   0x00005555555557dc <+119>:   lea    0x315d(%rip),%rdx        # 0x555555558940 <digit_stack>
   0x00005555555557e3 <+126>:   inc    %eax
   0x00005555555557e5 <+128>:   mov    %eax,(%rsp)
   0x00005555555557e8 <+131>:   cltq
   0x00005555555557ea <+133>:   mov    %r12b,(%rdx,%rax,1)
   0x00005555555557ee <+137>:   lea    0x312b(%rip),%rdx        # 0x555555558920 <flag_stack>
   0x00005555555557f5 <+144>:   movb   $0x0,(%rdx,%rax,1)
   0x00005555555557f9 <+148>:   dec    %rbx
   0x00005555555557fc <+151>:   test   %ebx,%ebx
   0x00005555555557fe <+153>:   jg     0x5555555557cb <_Z31build_candidate_expression_treePci+102>
```

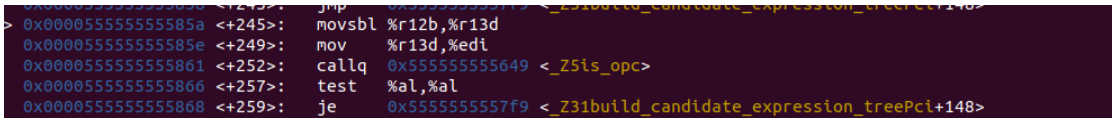
如果是数字，同样压入 digit_stack 栈，并用 flag==0 表示正数，flag==1 表示负数，压入 flag_stack 栈，并给计数器加 1。

```
   0x0000555555555808 <+163>:   cmp    $0x29,%r12b
   0x000055555555580c <+167>:   jne    0x555555555827 <_Z31build_candidate_expression_treePci+194>
   0x000055555555580e <+169>:   mov    0x4(%rsp),%eax
   0x0000555555555812 <+173>:   lea    0x30e7(%rip),%rdx        # 0x555555558900 <op_stack>
   0x0000555555555819 <+180>:   inc    %eax
   0x000055555555581b <+182>:   mov    %eax,0x4(%rsp)
   0x000055555555581f <+186>:   cltq
   0x0000555555555821 <+188>:   movb   $0x29,(%rdx,%rax,1)
   0x0000555555555825 <+192>:   jmp    0x5555555557f9 <_Z31build_candidate_expression_treePci+148>
```

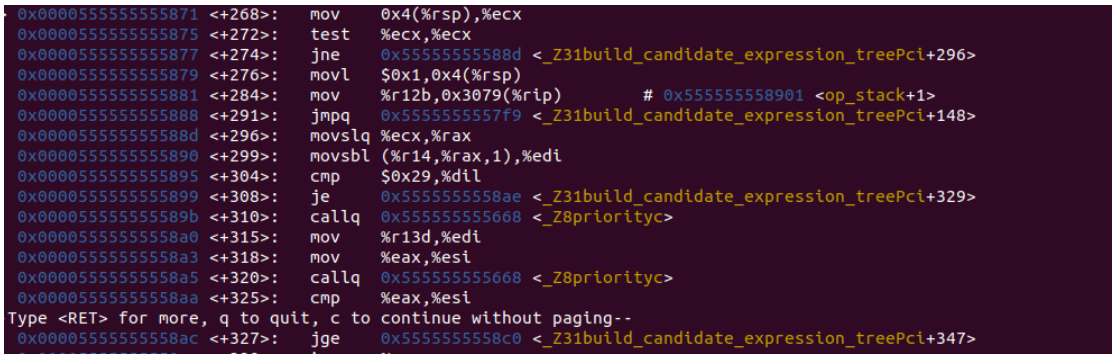
```
   0x0000555555555827 <+194>:   cmp    $0x28,%r12b
   0x000055555555582b <+198>:   jne    0x55555555585a <_Z31build_candidate_expression_treePci+245>
   0x000055555555582d <+200>:   lea    0x30cc(%rip),%r12        # 0x555555558900 <op_stack>
   0x0000555555555834 <+207>:   movslq 0x4(%rsp),%rdx
   0x0000555555555839 <+212>:   cmpb   $0x29,(%r12,%rdx,1)
   0x000055555555583e <+217>:   mov    %rdx,%rax
   0x0000555555555841 <+220>:   je     0x555555555852 <_Z31build_candidate_expression_treePci+237>
   0x0000555555555843 <+222>:   mov    %rsp,%rsi
   0x0000555555555846 <+225>:   lea    0x4(%rsp),%rdi
   0x000055555555584b <+230>:   callq  0x55555555569a <_Z11attach_nodePiS_>
   0x0000555555555850 <+235>:   jmp    0x555555555834 <_Z31build_candidate_expression_treePci+207>
   0x0000555555555852 <+237>:   dec    %eax
   0x0000555555555854 <+239>:   mov    %eax,0x4(%rsp)
   0x0000555555555858 <+243>:   jmp    0x5555555557f9 <_Z31build_candidate_expression_treePci+148>
```

如果是运算符，就会跳转到 163 行，这里先判断该操作符是否为‘）’（0x29），是则直接压入

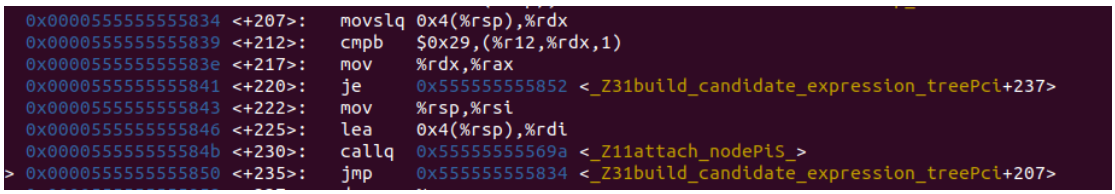
op_stack 栈，同时 op 计数加 1；不是则判断是否为‘（’(0x28)，如果是‘（’则判断此时 op_stack 栈顶是否为‘（’，是就把‘（’弹出栈；



如果都不是则跳到 245 行判断是不是运算符，不是运算符直接跳过，是运算符则进入一个循环



循环内先看 op_stack 内是否有元素，没有就将该操作符压栈然后结束循环；有则先判断栈顶是不是‘）’,是就直接将操作符压栈，不是则分别将当前操作符和栈顶操作符输入 priority 函数，该函数返回操作符的优先级；如果上一个操作符的优先级大于或等于当前操作符的优先级则结束循环，否则跳转到 347 行，即执行 attach_node 函数；

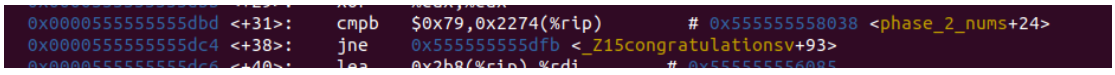


attach_node 函数将数据储存在 cnad 这段连续的空间内，先将当前操作符存入再将 digit_stack 的上面两个操作数存入循环调用直到遇到‘）’，并把‘）’弹出，或者操作符栈为空。

结束后 phase_6 继续调用 compare_answer_and_candidate 函数，显然该函数是比较上一个函数得到的保存在 cand 中的结果和 ans 是否一致，查看 ans 中的值为- 3 4 – 9 0 / + 1 2;与输入前的字符串相比并结合前缀表达式，推测答案为/(3-4)

Secret_phase

查看 congratulations 函数的汇编码，发现会判断 phase_2_nums+24 和 0x79 是否相等



返回查看发现在 phase_5 中当第一个输入为 8 时，即选择 baselock 类会给 phase_2_nums+24 赋值 0x79；通过之后即可进入 secret_phase 函数

进入 secret_phase 函数后，先将输入存进 xmm0 寄存器中，addss 指令 xmm0 寄存器加上它本身，相当于乘 2；

```
   0x0000555555555d66 <+0>:     endbr64
   0x0000555555555d6a <+4>:     cvtsi2ss %rdi,%xmm0
   0x0000555555555d6f <+9>:     addss  %xmm0,%xmm0
=> 0x0000555555555d73 <+13>:    addss  0x475(%rip),%xmm0        # 0x5555555561f0
   0x0000555555555d7b <+21>:    cvtss2sd %xmm0,%xmm0
```

查看 0x5555555561f0 中浮点内容为 10，即将 xmm0 加 10，cvtss2sd 把 xmm0 中的单精度浮点数转换为双精度浮点数

```
   0x0000555555555d66 <+0>:     endbr64
   0x0000555555555d6a <+4>:     cvtsi2ss %rdi,%xmm0
   0x0000555555555d6f <+9>:     addss  %xmm0,%xmm0
=> 0x0000555555555d73 <+13>:    addss  0x475(%rip),%xmm0        # 0x5555555561f0
   0x0000555555555d7b <+21>:    cvtss2sd %xmm0,%xmm0
   0x0000555555555d7f <+25>:    comisd 0x471(%rip),%xmm0        # 0x5555555561f8
   0x0000555555555d87 <+33>:    jae    0x555555555d97 <_Z12secret_phasel+49>
   0x0000555555555d89 <+35>:    movsd  0x46f(%rip),%xmm1        # 0x555555556200
   0x0000555555555d91 <+43>:    comisd %xmm0,%xmm1
   0x0000555555555d95 <+47>:    jb     0x555555555d9d <_Z12secret_phasel+55>
   0x0000555555555d97 <+49>:    push   %rax
   0x0000555555555d98 <+50>:    callq  0x555555555a46 <_Z12explode_bombv>
   0x0000555555555d9d <+55>:    retq
End of assembler dump.
(gdb) x/f 0x5555555561f0
0x5555555561f0: 10
```

查看和 xmm0 比较的数为 3820.000000999999

```
   0x0000555555555d7b <+21>:    cvtss2sd %xmm0,%xmm0
   0x0000555555555d7f <+25>:    comisd 0x471(%rip),%xmm0        # 0x5555555561f8
=> 0x0000555555555d87 <+33>:    jae    0x555555555d97 <_Z12secret_phasel+49>
   0x0000555555555d89 <+35>:    movsd  0x46f(%rip),%xmm1        # 0x555555556200
   0x0000555555555d91 <+43>:    comisd %xmm0,%xmm1
   0x0000555555555d95 <+47>:    jb     0x555555555d9d <_Z12secret_phasel+55>
   0x0000555555555d97 <+49>:    push   %rax
   0x0000555555555d98 <+50>:    callq  0x555555555a46 <_Z12explode_bombv>
   0x0000555555555d9d <+55>:    retq
End of assembler dump.
(gdb) x/f 0x5555555561f8
0x5555555561f8: 3820.0000009999999
```

查看 xmm1 中的浮点数

```
   0x0000555555555d7f <+25>:    comisd 0x471(%rip),%xmm0        # 0x5555555561f8
   0x0000555555555d87 <+33>:    jae    0x555555555d97 <_Z12secret_phasel+49>
   0x0000555555555d89 <+35>:    movsd  0x46f(%rip),%xmm1        # 0x555555556200
=> 0x0000555555555d91 <+43>:    comisd %xmm0,%xmm1
   0x0000555555555d95 <+47>:    jb     0x555555555d9d <_Z12secret_phasel+55>
   0x0000555555555d97 <+49>:    push   %rax
   0x0000555555555d98 <+50>:    callq  0x555555555a46 <_Z12explode_bombv>
   0x0000555555555d9d <+55>:    retq
End of assembler dump.
(gdb) x/f 0x555555556200
0x555555556200: 3819.9999990000001
```

最后结合两条比较指令，可以得出输入的数乘 2 加 10 的结果要小于 3820.0000009 并且大于 3819.999999，所以输入的数为 1905