

# ICS Lab2-BombLab 实验报告

——邹懿 21302010073

## 实验思路

### phase\_1

首先可以发现，phase\_1的参数是一个字符串，而string\_not\_equal的参数为两个字符串，根据函数名可以知道其实是在判断输入的字符串是否和string\_not\_equal的第二个参数是否相同，然后就在进入string\_not\_equal前打印%rsi储存的值查看，如图

```
0x0000555555555555d20 in phase_1(char*)()
(gdb) si
0x00005555555555a60 in string_not_equal(char*, char*)()
(gdb) x/s $rsi
0x5555555558146 <w1+230>:          "each line is important"
(gdb) |
```

发现为“each line is important”，然后将其作为第一关的password输入发现猜想正确！

### phase\_2

同理，我们知道phase\_2参数为字符串，同时函数调用了read\_six\_numbers检查输入字符串是否符合某种要求，通过函数名同样可以知道实际上这道题我们需要输入的是六个数字。紧接着探索这六个数字满足的关系，如下图，我发现%rax保存的是我们输入的第一数字num1，%ecx储存的是一个定值k，然后在1b5c这一行将%rax和%ecx表示的值相乘，并和0x4(%rax)代表的值比较，而联想到int为4个字节可以猜测0x4(%rax)也就是输入的第二个数num2，并且由1b62这一行我们知道必须满足num2=k\*num1，继续下去也就是一个循环判断相邻两个数是否满足同样的关系，因此这道题就是输入一个以k为公比的等比数列

1b49:	48 8d 05 d0 24 00 00	lea	0x24d0(%rip),%rax	# 4020		
<phase_2_nums>						
1b50:	8b 0d e6 24 00 00	mov	0x24e6(%rip),%ecx	# 403c		
<phase_2_nums+0x1c>						
1b56:	48 8d 50 14	lea	0x14(%rax),%rdx			
1b5a:	8b 30	mov	(%rax),%esi			
1b5c:	0f af f1	imul	%ecx,%esi			
1b5f:	39 70 04	cmp	%esi,0x4(%rax)			
1b62:	74 05	je	1b69 <phase_2(char*)+0x31>			
1b64:	e8 dd fe ff ff	callq	1a46 <explode_bomb()>			
1b69:	48 83 c0 04	add	\$0x4,%rax			
1b6d:	48 39 c2	cmp	%rax,%rdx			
1b70:	75 e8	jne	1b5a <phase_2(char*)+0x22>			
1b72:	58	pop	%rax			
1b73:	c3	retq				

打印%ecx代表的值如下图，发现k=-3

```
(gdb) p $ecx  
$2 = -3  
(gdb) |
```

## phase\_3

phase\_3我认为最重要去理解下面这几行(如下图), 前三行我最先很困惑, 不知道在干嘛, 我先是跳过这几行观察1ba2这一行发现实际上在判断输入的字符串是否有三个字符, 然后我便猜测前三行应该是在把输入的三个参数传入对应的寄存器, 其中1st--0x10(%rsp),2nd--0xf(%rsp),3th--0x14(%rsp),当然也可以实际验证,了解了这个之后就在后面寻找对应的满足条件。

```
1b93: 48 8d 4c 24 0f          lea    0xf(%rsp),%rcx  
1b98: 48 8d 54 24 10          lea    0x10(%rsp),%rdx  
1b9d: 4c 8d 44 24 14          lea    0x14(%rsp),%r8  
1ba2: e8 c9 f5 ff ff          callq 1170 <__isoc99_sscanf@plt>  
1ba7: 83 f8 03                cmp    $0x3,%eax  
1baa: 75 65                  jne    1c11 <phase_3(char*)+0x9d>
```

### 1st parameter:

```
1bac: 83 7c 24 10 07          cmpl   $0x7,0x10(%rsp)  
1bb1: 77 5e                   ja    1c11 <phase_3(char*)+0x9d>
```

我们发现第一个数字首先应该满足小于等于7

### 3th parameter:

紧接着继续运行我发现给出了第三个参数需要满足的条件

```
1bef: 83 7c 24 14 10          cmpl   $0x10,0x14(%rsp)  
1bf4: b0 74                   mov    $0x74,%al  
1bf6: 74 1e                   je    1c16 <phase_3(char*)+0xa2>  
1bf8: eb 17                   jmp    1c11 <phase_3(char*)+0x9d>
```

我发现第三个数字必须等于0x10也就是16(虽然前期被炸傻后还以为0x10就是10orz)

### 2nd parameter:

```
//需要满足:  
1c16: 38 44 24 0f          cmp    %al,0xf(%rsp)  
1c1a: 75 f5                 jne    1c11 <phase_3(char*)+0x9d>
```

可以发现第二个参数对应的值应该和%al的值相同, 而从前面 mov \$0x74,%al 指令可以知道它等于116

然后! ! ! 我就认为第二个字符就是116可是一直错误! ! 后来我突然根据第一个参数和第二个参数栈指针偏移量只相差1推断2nd 参数是char类型, 116是其ascall码, 所以应该是t!!

## phase\_4

首先需要明确其参数是一个64位的整数，这里我们设为x,然后分析代码

第一个条件：

```
1c3a: 48 89 f8          mov    %rdi,%rax
1c3d: 48 c1 ff 20       sar    $0x20,%rdi
1c41: 52                 push   %rdx
1c42: 8d 57 ff          lea    -0x1(%rdi),%edx
1c45: 83 fa 0d          cmp    $0xd,%edx
1c48: 77 07              ja    1c51 <phase_4(long)+0x1b>
```

这里首先是将x的值先赋值给%rax储存，然后将x右移32位，之后比较( $x \ll 31-1$ )与0xd(13)的大小，**必须满足**( $x \ll 31-1 \leq 14$ ，即：x的前32位所代表的数字必须不大于14

第二个条件：

```
1c4a: ff c8              dec    %eax
1c4c: 83 f8 0d          cmp    $0xd,%eax
1c4f: 76 05              jbe    1c56 <phase_4(long)+0x20>
```

这里%eax储存的是x的后32位，这里表明其后32位代表的数字x'必须满足( $x'-1 \leq 13$ ,**也就是**x'不大于14

第三个条件：

第三个条件需要在美丽的递归中去体会，我们需要在hope(int)函数去“感悟”，不过在这之前这里需要注意传入的是32位，然后通过观察发现它实际上传入的是**x的前32位x1**，接下来就可以开启“递归体验之旅”了,以下是我反汇编后猜测的源代码

```
int hope(int x){
    int result;
    int num1 = 1;
    if(!x) {
        result = num1;
        return result;
    }
    int xx = x;
    x = x>>1;
    int value = hope(x);
    value = value*value;
    if(xx&1) value *= 4;
    result = value;
    return result;
}
```

然后我们返回看phase\_4后面是在比较hope函数返回值是否与0x1000000 (2的24次方) 相等，然后我就想怎么能凑出这个答案推导如下

$2^{24} \Rightarrow result1 = 1^2 = 1 \rightarrow x \text{右移5次末位} 0$   
 $result2 = 4 \times 1^2 = 2^2 \rightarrow x \text{右移4次末位} 1$   
 $result3 = 4 \times (2^2)^2 = 2^6 \rightarrow x \text{右移3次末位} 1$   
 $result4 = (2^6)^2 = 2^{12} \rightarrow x \text{右移2次末位} 0$   
 $result5 = (2^{12})^2 = 2^{24} \rightarrow x \text{右移1次末位} 0$

$\Rightarrow x = \underbrace{00 \dots \dots 0}_{\text{前32位}} \underbrace{1100}_{\leq 13} \underbrace{00 \dots \dots 001}_{\leq 13}$

$\Rightarrow x: 51539607555$

## phase\_5

对于phase\_5的评价是华而不实，但又却始终重要，这里就不分析另外两种情况，只分析开启secret\_phase的第三种情况(**第一个数字是8**)

首先需要明确它**有三个参数**，然后很简单的我们确定第一个数字为8之后进入对应的函数的极限套娃acquire(int), is\_holding(int), release(int, int), is\_holding(int), release(int).....这需要十分耐心去发现跳出的条件，否则就会进入死循环，这其中最重要的句子就是

```

0x5555555554bc <_ZN8baseLock7releaseEii+18>: cmp    $0xff0,%ebp
0x5555555554c2 <_ZN8baseLock7releaseEii+24>: jne    0x5555555554d9
<_ZN8baseLock7releaseEii+47>

```

其中%ebp储存的是%edx也就是第三个参数的值，着这行语句要求**第三个参数值必须等于0xff0(4080)**，而对第二个参数值未作要求

## phase\_6

```

0x555555555d0f <_Z7phase_6Pc+5>:    callq  0x555555555a7c <_Z10string_lenPc>
0x555555555d14 <_Z7phase_6Pc+10>:   lea     0x27f0(%rip),%rdx      #
0x555555555850b <w2+11>
0x555555555d1b <_Z7phase_6Pc+17>:   mov     %eax,%r8d
0x555555555d1e <_Z7phase_6Pc+20>:   xor     %eax,%eax
0x555555555d20 <_Z7phase_6Pc+22>:   cmp     $0x6,%r8d
0x555555555d24 <_Z7phase_6Pc+26>:   je      0x555555555d2b <_Z7phase_6Pc+33>
0x555555555d26 <_Z7phase_6Pc+28>:   callq  0x555555555a46
<_Z12explode_bombv>

```

首先phase\_6参数同样为字符串，从调用string\_len函数以及比较返回值与6的大小我们可以知道是在要求输入字符串长度必须为6，但其实我们仍然对这个字符串的内容十分茫然，但是！下面的这两行给了我灵感

```
1d40: 48 8d 3d b9 27 00 00    lea    0x27b9(%rip),%rdi      # 4500 <w2>
1d47: e8 19 fa ff ff        callq 1765
<build_candidate_expression_tree(char*, int)>
```

通过bulid\_candidate\_expression\_tree我大概能知道这个字符串应该与计算表达式有关，然后打印0x27b9(%rip)验证（其中/(3-4)是我们自己输入的）

```
(gdb) x/s 0x5555555558500  
0x5555555558500 <w2>:      "(1+2)*(9-0)/(3-4)"  
(gdb) |
```

这说明我们的猜想正确，然后通过build\_candidate\_expression\_tree函数名可以知道，这是将中缀表达式搭建为二叉树，然后在phase\_6后半部分中可以知道，实际上在判断我们的candidate\_tree和answer\_tree时候相同，这就考验我们的对比能力了

```
(gdb) x/10i $pc
=> 0x555555555929 <_Z28compare_answer_and_candidateP9tree_nodeS0_i+7>: sete %al
0x55555555592c <_Z28compare_answer_and_candidateP9tree_nodeS0_i+10>: test %rsi,%rsi
0x55555555592f <_Z28compare_answer_and_candidateP9tree_nodeS0_i+13>: sete %cl
0x555555555932 <_Z28compare_answer_and_candidateP9tree_nodeS0_i+16>: or %cl,%al
0x555555555934 <_Z28compare_answer_and_candidateP9tree_nodeS0_i+18>:
jne 0x555555555978 <_Z28compare_answer_and_candidateP9tree_nodeS0_i+86>
0x555555555936 <_Z28compare_answer_and_candidateP9tree_nodeS0_i+20>: lea 0x2a43(%rip),%rcx
0x55555555593d <_Z28compare_answer_and_candidateP9tree_nodeS0_i+27>: lea 0x2e3c(%rip),%rsi
0x555555555944 <_Z28compare_answer_and_candidateP9tree_nodeS0_i+34>: xor %edi,%edi
0x555555555946 <_Z28compare_answer_and_candidateP9tree_nodeS0_i+36>: cmp %edi,%edx
0x555555555948 <_Z28compare_answer_and_candidateP9tree_nodeS0_i+38>:
jl 0x555555555978 <_Z28compare_answer_and_candidateP9tree_nodeS0_i+89>
```

红色圈的是对应的地址位置

其中answer\_tree为

```
(gdb) x/100c 0x55555555558380
0x55555555558380 <ans>: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x55555555558388 <ans+8>: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 45 '-' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x55555555558390 <ans+16>: 2 '\002' 0 '\000' 0 '\000' 0 '\000' 3 '\003' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x55555555558398 <ans+24>: 51 '3' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x555555555583a0 <ans+32>: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 52 '4' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x555555555583a8 <ans+40>: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x555555555583b0 <ans+48>: 45 '-' 0 '\000' 0 '\000' 0 '\000' 5 '\005' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x555555555583b8 <ans+56>: 6 '\006' 0 '\000' 0 '\000' 0 '\000' 57 '9' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x555555555583c0 <ans+64>: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x555555555583c8 <ans+72>: 48 '0' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x555555555583d0 <ans+80>: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 47 '/' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x555555555583d8 <ans+88>: 4 '\004' 0 '\000' 0 '\000' 0 '\000' 1 '\001' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
0x555555555583e0 <ans+96>: 43 '+' 0 '\000' 0 '\000' 0 '\000'
```

candidate\_tree为

```
(gdb) x/100c 0x5555555558780
0x5555555558780 <cand>: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
'\000'
0x5555555558788 <cand+8>: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 45 '-' 0 '\000' 0 '\000' 0 '\000'
'\000'
0x5555555558790 <cand+16>: 2 '\002' 0 '\000' 0 '\000' 0 '\000' 3 '\003' 0 '\000' 0 '\000' 0 '\000'
'0'
0x5555555558798 <cand+24>: 57 '9' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
'\000'
0x55555555587a0 <cand+32>: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 48 '0' 0 '\000' 0 '\000' 0 '\000'
'\000'
0x55555555587a8 <cand+40>: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
'0'
0x55555555587b0 <cand+48>: 45 '-' 0 '\000' 0 '\000' 0 '\000' 5 '\005' 0 '\000' 0 '\000' 0 '\000'
'\000'
0x55555555587b8 <cand+56>: 6 '\006' 0 '\000' 0 '\000' 0 '\000' 57 '9' 0 '\000' 0 '\000' 0 '\000'
'\000'
0x55555555587c0 <cand+64>: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
'0'
0x55555555587c8 <cand+72>: 48 '0' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
'\000'
0x55555555587d0 <cand+80>: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 42 '*' 0 '\000' 0 '\000' 0 '\000'
'\000'
0x55555555587d8 <cand+88>: 4 '\004' 0 '\000' 0 '\000' 0 '\000' 1 '\001' 0 '\000' 0 '\000' 0 '\000'
'0'
0x55555555587e0 <cand+96>: 43 '+' 0 '\000' 0 '\000' 0 '\000'

(gdb)
```

然后我们对比这两棵树发现，答案树中和我们搭建的树相同位置的值有几点不同：51'3'——57'9'，52'4'——48'0'，47''——42'\*'，而我们输入的'-'号能够相对应说明输入正确，对应调整后可以得出正确答案为/(3-4)

## secret\_phase

主要是打印查看答案，secret\_phase主要就是两处比较大小满足即可，打印如下（此时输入为1000）

```
0x00000555555d87 in secret_phase(long) ()
(gdb) p $xmm0
$1 = {v4_float = {0, 4.981444531, -nan(0x7fdd40), 4.59163468e-41}, v2_double = {2010, 6.9533558073954835e-310}, v16_int8 = {0, 0, 0,
0, 0, 104, -97, 64, 64, -35, -1, -1, -1, 127, 0, 0}, v8_int16 = {0, 0, 26624, -33000, -8896, 32767}, v4_int32 = {0,
1084188672, -8896, 32767}, v2_int64 = {4656554888933670912, 140737488346432}, uint128 = 2596148429103316235540416927105024}
(gdb) x/lf 0x55555555561f8
0x5555555555561f8: 3.08148755e-39
(gdb) si
0x0000055555555d89 in secret_phase(long) ()
(gdb) si
0x000005555555555d91 in secret_phase(long) ()
(gdb) p $xmm1
$2 = {v4_float = {-nan(0x5e7211), 5.43261671, 0, 0}, v2_double = {3819.999999000001, 0}, v16_int8 = {17, 114, -34, -1, -1, -41,
-83, 64, 0, 0, 0, 0, 0, 0}, v8_int16 = {29201, -34, -1024, -1024, -1024, -1024, -1024, 0}, v4_int32 = {-2199023, 1085134847, 0, 0},
v2_int64 = {4660618683907731985, 0}, uint128 = 4660618683907731985}
(gdb) a
```

实际实验中知道，%xmm0代表的值首先被赋值为 $2x(x$ 为输入值的后32位所代表的数)，然后加10，我们发现需要满足 $2x+10 > 3819.9999...$ ，可以让它为3820然后得出 $x$ 为1905，然后试一下！！正确了！！拆弹结束！

## 实验结果

```
(gdb) run < password
Starting program: /mnt/d/学习资料/ics lab2/bomb < password
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 413) exited normally]
(gdb) |
```