Bomblab 实验报告

郭丹琪　2018202067

Phase_1:

密码: I can see Russia from my house!

解题过程:在 phase_1 的汇编代码中发现用到 strings_not_equal 函数，推断 phase_1 函数是要判断两个字符串是否相等。在 strings_not_equal 函数的汇编代码中，有

```
mov    %rdi,%rbx
mov    %rsi,%rbp
```

所以得知该函数有两个参数，分别存在%rdi 和%rsi 中，且通过该语句存到了%rbx 和%rbp 里。所以推测%rdi 和%rsi 其中一个存的是正确密码，另一存的是输入的字符串，所以密码就在%rdi 或%rsi 中。再看函数 phase_1，发现语句 lea　0x18c1(%rip),%rsi，即在该操作后%rsi 中存了一个内存地址，推测%rsi 中存的是正确密码的地址。在调试时，直接在 strings_not_equal 处设置断点，用 info registers 得到寄存器里存的内容，得到

```
(gdb) info registers
rax            0x5555557586c0   93824994346688
rbx            0x0      0
rcx            0x5      5
rdx            0x5555557586c0   93824994346688
rsi            0x555555556b20   93824992242464
rdi            0x5555557586c0   93824994346688
```

，再用 print 打印出两个寄存器存的地址里对应的字符串，即

```
(gdb) print (char*)(0x5555557586c0)
$1 = 0x5555557586c0 <input_strings> "stepi"
(gdb) print (char*)(0x555555556b20)
$2 = 0x555555556b20 "I can see Russia from my house!"
```

，发生正确密码的地址存在%rsi 中，且密码为 I can see Russia from my house!

Phase_2:

密码: 0 1 3 6 10 15

解题过程: 在 phase_2 的汇编代码里发现调用了函数 read_six_numbers，所以判断该题的密码是 6 个数字。在 read_six_numbers 函数中，有这样一段代码

```
mov    $0x0,%eax
callq  f30 <__isoc99_sscanf@plt>
add    $0x10,%rsp
cmp    $0x5,%eax
jle    1b63 <read_six_numbers+0x3c>
add    $0x8,%rsp
retq
callq  1aeb <explode_bomb>
```

即为先把%eax 里的值设为 0，再调用 sscanf，然后再调用后%eax 上的值与 5 的大小，若%eax 上的值 < 5 则爆炸。所以可以推断这里%eax 是在记录输入了几个数字，如果输入的数字没有 6 个则可以直接爆炸。所以可以确定 read_six_numbers 函数只是读入并确定是 6 个数字，没有对数字的值进行判断。再看 phase_2，在读完 6 个数字后，先用 cmpl　$0x0,(%rsp)比较了(%rsp)上存的值与 0 的大小，

即为判断输入的第一个数,小于 0 则跳转爆炸。再往下看发现在 mov $0x1,%ebx 语句后，有个循环

```
add    $0x1,%rbx
cmp    $0x6,%rbx
je     12c6 <phase_2+0x52>
mov    %ebx,%eax
add    -0x4(%rbp,%rbx,4),%eax
cmp    %eax,0x0(%rbp,%rbx,4)
je     12a9 <phase_2+0x35>
```

，即为每次对%rbx 的值+1，然后将%rbx 的值与 6 比较，若%rbx<6 则进入接下来的语句，否则跳出循环。所以可以得知此处是在对输入的 6 个数字进行遍历。所以可以将输入的 6 个数字假设对应 c 语言为存在数组 a[1~6]中，在循环语句里%rbx 里存的是下标 i。而在循环中，-0x4(%rbp,%rbx,4)和 0x0(%rbp,%rbx,4)内存里存的就是数组对应的值，且 a [1]=(%rbp)。所以该段汇编代码对应的 c 语言代码可以写为

```
for(int i=1;i<6;i++)
{
    if(a[i]+i!=a[i+1])
        break;
}
```

即得满足 a[1]+1==a[2]，a[2]+2==a[3]，a[3]+3==a[4]，a[4]+4==a[5]，a[5]+5==a[6]，否则爆炸。再根据该循环前的比较,即为 a[1]<0 则爆炸。所以假设 a[1]=0，则 a[2]=a[1]+1=1，a[3]=a[2]+2=3，a[4]=a[3]+3=6，a[5]=a[4]+4=10，a[6]=a[5]+5=15。测试时发现密码正确，所以得到结果。

Phase_3:

密码: 7 g 227

解答过程:在调用的输入的函数前有 xor %eax,%eax 将%eax 里的值设为 0，在输入后有

```
cmp    $0x2,%eax
jle    1335 <phase_3+0x53>
```

来比较%eax 与 2 的大小，小于等于则跳转爆炸，所以判断该题要输入三个值。且在调用输入的函数前,有 lea 0x185a(%rip),%rsi，则%rsi 里存了一个参数，在调试的时候输出得%rsi 存的地址对应内存上的内容为"%d %c %d"，所以可以得知该题密码为一个数字+一个字符+一个数字。在接下来的汇编代码中，先有

```
cmpl   $0x7,0x10(%rsp)
ja     142d <phase_3+0x14b>
```

，表示 0x10(%rsp)当成无符号数大于 7 时跳转爆炸，小于等于 7 时进行

```
mov    0x10(%rsp),%eax
lea    0x1854(%rip),%rdx
movslq (%rdx,%rax,4),%rax
add    %rdx,%rax
jmpq   *%rax
```

，即为根据 0x10(%rsp)的值跳转到不同的地方，所以可以得知该段语句及接下来跳转的内容是在进行 c 语言里的 switch 操作。可以推断输入的第一个数要小于等于 7，且根据不同的值进行不同的操作。这里先选择第一个数字为 7，此时跳转到

```
mov     $0x67,%eax
cmpl    $0xe3,0x14(%rsp)

je      1437 <phase_3+0x155>
callq   1aeb <explode_bomb>
```

，即判断第三个数是否为 0xe3,即为 227。

若相等，跳转到

```
cmp     %al,0xf(%rsp)
je      1442 <phase_3+0x160>
callq   1aeb <explode_bomb>
```

。此时的%al=0x67，即为 103，在 ascii 表里对应为 g。所以得到该题密码为 7 g 227，同时可以推测当第一个数为 0~7 的其他数时，会有不同的结果。

Phase_4:

密码: 10　37

解题过程：在 phase_4 的汇编代码中看到语句

```
xor     %eax,%eax
lea     0x4(%rsp),%rcx
mov     %rsp,%rdx
lea     0x194a(%rip),%rsi        #
callq   f30 <__isoc99_sscanf@plt>
cmp     $0x2,%eax
jne     14c3 <phase_4+0x33>
```

，%eax 里记录输入了几个数或字符等，所以这题要输入两个数或字符，且输入的类型存在%rsi 里。在测试时输出此处%rsi 里的地址对应存的内容为 "%d %d"，所以得知该题要输入两个数。由

```
cmpl    $0xe,(%rsp)
jbe     14c8 <phase_4+0x38>
callq   1aeb <explode_bomb>
```

得输入的第一个数得小于 0xe，即 14。之后

```
mov     $0xe,%edx
mov     $0x0,%esi
mov     (%rsp),%edi
```

，且在这 3 条语句后调用了函数 func4，%edx,%esi,%edi 里存了函数 func4 的三个参数，且已确定一个参数是 14，一个参数是 0，一个参数是输入的第一个数。且在调用了这个函数后，判断了%eax 里的值是否等于 0x25，即 37，不相等则跳转爆炸，所以可以判断 func4 函数的返回值得是 37。此后又用

```
cmpl   $0x25,0x4(%rsp)
je     14eb <phase_4+0x5b>
callq  1aeb <explode_bomb>
```

判断了输入的第二个数是否等于 37，相等则跳转不爆炸，所以可以判断输入的第二个数是 37。而第一个数要取小于等于 14 的什么值得通过函数 func4 判断。所以此时看向 func4。得保证函数的返回值等于 37，写出对应 c 语言代码。把%edx 对应为 x，把%esi 对应为 y，把%edi 对应为 z，把%eax 对应为返回值 t，把%ebx 对应为 k。

```
push   %rbx
mov    %edx,%eax
sub    %esi,%eax
mov    %eax,%ebx
shr    $0x1f,%ebx
add    %eax,%ebx
sar    %ebx
add    %esi,%ebx
```

对应

```
int func(int x,int y,int z)
{
    int t;
    t=x;
    t=t-y;
    int k=t;
    if(k>=0) k=0;
    else k=1;
    k=k+t;
    k=k>>1;
    k=k+y;
```

进行值的加减和移位，把 shr $0x1f,%ebx 的移位过程直接简化为对 k 正负的判断。

```
cmp    %edi,%ebx
jg     1478 <func4+0x1c>
cmp    %edi,%ebx
jl     1484 <func4+0x28>
mov    %ebx,%eax
pop    %rbx
retq

lea    -0x1(%rbx),%edx
callq  145c <func4>
add    %eax,%ebx
jmp    1474 <func4+0x18>

lea    0x1(%rbx),%esi
callq  145c <func4>
add    %eax,%ebx
jmp    1474 <func4+0x18>
```

对应

```
if(k>z)
{
    x=k-1;
    t=func(x,y,z);
    k=k+t;
}
else
if(k<z)
{
    y=k+1;
    t=func(x,y,z);
    k=k+t;
}
t=k;
return t;
```

即先把%ebx（也就是 k）与%edi（也就是 z）进行比较，在等于的时候可以直接让%eax=%ebx（t=k）返回，在大于小于的部分分别进行操作，并再调用此函数进行递归。再在主函数里对 z 的值（即输入的第一个数）进行从 0 到 14 的遍历，输出 z 的值和对应 func4 的结果，得到

```
0 11
1 11
2 13
3 10
4 19
5 15
6 21
7 7
8 35
9 27
10 37
11 18
12 43
13 31
14 45
```

因为需要函数的返回值是 37，所以可以确定输入的第一个数应该是 10。

Phase_5

密码: ztoefw

解题过程:

```
xor    %eax,%eax
callq  17ca <string_length>
cmp    $0x6,%eax
jne    157c <phase_5+0x77>
```

得知本题要输入一个字符串,且用%eax 记录字符串长度，得知密码字符串的长度是 6。在该段语句

```
mov    $0x0,%eax
lea    0x166d(%rip),%rcx
movzbl (%rbx,%rax,1),%edx
and    $0xf,%edx
movzbl (%rcx,%rdx,1),%edx
mov    %dl,0x1(%rsp,%rax,1)
add    $0x1,%rax
cmp    $0x6,%rax
jne    1533 <phase_5+0x2e>
```

通过一开始把%eax 设为 0，并在最后对%rax+1,再判断%rax 是否等于 6,如果不等于则跳回第 3 行的语句，得知该段语句在进行对输入的字符串内容的进行循环。且其中

```
movzbl (%rbx,%rax,1),%edx
and    $0xf,%edx
movzbl (%rcx,%rdx,1),%edx
mov    %dl,0x1(%rsp,%rax,1)
```

在对输入的字符串进行操作。在

```
lea     0x166d(%rip),%rcx
```
的语句后调试输出%rcx 里的内容得到 maduiersnfotvbyl，所以可以写出该段语句对应的 c 语言,所以可以通过对 26 个字母分别操作得到不同的字母在该段操作后的结果

```
char rcx[16]={'m','a','d','u','i','e',
'r','s','n','f','o','t','v','b','y','l'};
for(int i=0;i<=25;i++)
{
    char a=97+i;
    int b=a;
    b=b&0xf;
    char c=rcx[b];
    int d=c;
    char e=d;
    cout<<a<<" "<<e<<endl;
}
```

```
a a
b d
c u
d i
e e
f r
g s
h n
i f
j o
k t
l v
```

```
m b
n y
o l
p m
q a
r d
s u
t i
u e
v r
w s
x n
y f
z o
```

。而之后通过

```
lea     0x1(%rsp),%rdi
lea     0x1612(%rip),%rsi
callq   17e7 <strings_not_equal>
```
来判断输入的字符串在经过操作后是否符合需要。所以在调试时输出%rsi 里的内容，得到的字符串为 oilers。所以根据刚才算出的 26 个字母分别对应的结果，得到应输入 ztoefw。

Phase_6:

密码: 1 2 3 6 4 5

```
xor     %eax,%eax
mov     %rsp,%r12
mov     %r12,%rsi
callq   1b27 <read_six_numbers>
```
解题过程:由得知本题要输入 6 个数字，且%eax 用来记输入了几个数，即在 call 语句后%eax 里的值是 6。把 6 个数字当成 C 语言中的 a[1-6]。在这段语句后把%r13d 设为 0，再用

```
mov     %r12,%rbp
mov     (%r12),%eax
sub     $0x1,%eax
cmp     $0x5,%eax
ja      15bc <phase_6+0x2d>
```
判断 a[1]，即得满足 a[1]<=6。之后

```
add     $0x1,%r13d
cmp     $0x6,%r13d
je      162f <phase_6+0xa0>
mov     %r13d,%ebx
jmp     15cb <phase_6+0x3c>
```
，即为当%r13d 不等于 6 的时候，先让%ebx 里存的等于%r13d 里存的，再跳转到前面，而前面的语句为

```
15c3:   83 c3 01                add    $0x1,%ebx
15c6:   83 fb 05                cmp    $0x5,%ebx
15c9:   7f 12                   jg     15dd <phase_6+0x4e>

15cb:   48 63 c3                movslq %ebx,%rax
15ce:   8b 04 84                mov    (%rsp,%rax,4),%eax
15d1:   39 45 00                cmp    %eax,0x0(%rbp)
15d4:   75 ed                   jne    15c3 <phase_6+0x34>
15d6:   e8 10 05 00 00          callq  1aeb <explode_bomb>
15db:   eb e6                   jmp    15c3 <phase_6+0x34>
```

，有对%ebx 与 5 的大小比较，大于则跳回对%r13d+1 和判断%r13d 与 6 的大小的语句，所以可以得知该段跳转语句是在对输入的 6 个数字进行遍历，且%r13d 里存的就相当于是 C 语言在遍历时的下标 i，且遍历从输入的第二个数开始，即从 a[2]开始。在循环体里，先让%eax 等于当前遍历到的数，再比较其与 0x0(%rbp)的大小（循环的第一轮即为与 a[1]比较大小），等于则跳转爆炸，不等于则在该次循环里再遍历后面的数，来与 0x0(%rbp)比较，且同样不能相等。在这次循环进行完后进行

```
add    $0x4,%r12
mov    %r12,%rbp
mov    (%r12),%eax
sub    $0x1,%eax
cmp    $0x5,%eax
ja     15bc <phase_6+0x2d>
```

即把%rbp 的位置往后移，且比较移完后存的值-1 后与 5 的大小，即存的值小于等于 6 的话进行下一轮的循环。所以可以得知该段循环语句是在确保输入的 6 个数字每一个都小于等于 6，且得保证这 6 个数字互不相等。出了该段循环后，先 mov　$0x0,%esi，再

```
mov    (%rsp,%rsi,4),%ecx
mov    $0x1,%eax
lea    0x202c08(%rip),%rdx
cmp    $0x1,%ecx
jg     15ff <phase_6+0x70>
jmp    160a <phase_6+0x7b>
```

让%ecx 等于 a[1]，让%eax=1，再把一个地址给%rdx，且该句备注了<node1>，然后比较%ecx 与 1 的大小，大于则跳转到

```
mov    0x8(%rdx),%rdx
add    $0x1,%eax
cmp    %ecx,%eax
jne    15ff <phase_6+0x70>
```

，让%rdx 里存的地址一直往后移 8 个字节，让%eax 一直+1 直到与%ecx 相等。相等后进行 mov　%rdx,0x20(%rsp,%rsi,8)，将%rdx 里存的地址存入栈中，然后对%rsi+1，比较%rsi 与 6 的大小，小于则又回到

```
mov    (%rsp,%rsi,4),%ecx
mov    $0x1,%eax
lea    0x202c08(%rip),%rdx
cmp    $0x1,%ecx
jg     15ff <phase_6+0x70>
jmp    160a <phase_6+0x7b>
```

。所以得知该段是按顺序对输入的 6 个数的遍历，且根据数的大小对应让%rdx 里存不同的地址，然后再把这个地址按顺序存入栈中。之后进行

```
mov    0x20(%rsp),%rbx
mov    0x28(%rsp),%rax
mov    %rax,0x8(%rbx)
mov    0x30(%rsp),%rdx
mov    %rdx,0x8(%rax)
mov    0x38(%rsp),%rax
mov    %rax,0x8(%rdx)
mov    0x40(%rsp),%rdx
mov    %rdx,0x8(%rax)
mov    0x48(%rsp),%rax
mov    %rax,0x8(%rdx)
movq   $0x0,0x8(%rax)
```

，相当于是把刚才得到到每个数对应的地址连起来，再由刚才备注里的 node 推测是在把输入的 6 个数存到链表里，这里的操作即为把整个链表连起来。然后 mov $0x5,%ebp，再跳转到

```
mov    0x8(%rbx),%rax
mov    (%rax),%eax
cmp    %eax,(%rbx)
jge    1677 <phase_6+0xe8>
```

即在比较第 1 个结点里的值和第 2 个结点里的值，第 1 个大于等于第 2 个才不会爆炸，跳转到

```
mov    0x8(%rbx),%rbx
sub    $0x1,%ebp
je     1691 <phase_6+0x102>
```

即进入下一个结点，然后再进行刚才的比较，即为比较第 2 个结点和第 3 个结点里的值。这里就是在遍历每一个结点，保证每一个结点里存的值比下一个结点里存的值大，全都成立后则该密码正确。所以得保证 6 个结点里依次存 6 5 4 3 2 1。在调试的时候依次输出各个结点的地址，得到输入时需按 1 2 3 6 4 5 排列，才能使该处判断成立。所以该题密码为 1 2 3 6 4 5。