Bomb Lab 实验报告

计 23 鲁逸沁 2012011314

【Bomb 信息】

Bomb ID：537

【Phase_1】

1. 获得汇编代码

```
<+0>:  sub    $0x8,%rsp
<+4>:  mov    $0x402670,%esi
<+9>:  callq  0x401408 <strings_not_equal>
<+14>: test   %eax,%eax
<+16>: je     0x400f67 <phase_1+23>
<+18>: callq  0x401698 <explode_bomb>
<+23>: add    $0x8,%rsp
<+27>: retq
```

2. 可以发现汇编代码中，调用了函数 string_not_equal。从名字上来看是判断两个 string 是否相等的函数。又观察到函数在调用前有一句 mov 命令，这句很可能是与输入串判断是否一样的串的地址。查看 0x402670 中的串

```
All your base are belong to us.
```

3. 输入以上串，通过调试发现%eax 等于 0，因此经过跳转它会进入<+23>的内容而跳过 explode_bomb，因此答案就是

```
All your base are belong to us.
```

【Phase_2】

1. 获得汇编代码

```
<+0>:  push   %rbp
<+1>:  push   %rbx
<+2>:  sub    $0x28,%rsp
<+6>:  mov    %rsp,%rsi
<+9>:  callq  0x4016ce <read_six_numbers>
<+14>: cmpl   $0x1,(%rsp)
<+18>: je     0x400fa0 <phase_2+52>
<+20>: callq  0x401698 <explode_bomb>
<+25>: jmp    0x400fa0 <phase_2+52>
<+27>: mov    -0x4(%rbx),%eax
<+30>: add    %eax,%eax
<+32>: cmp    %eax,(%rbx)
<+34>: je     0x400f95 <phase_2+41>
<+36>: callq  0x401698 <explode_bomb>
<+41>: add    $0x4,%rbx
<+45>: cmp    %rbp,%rbx
<+48>: jne    0x400f87 <phase_2+27>
<+50>: jmp    0x400fac <phase_2+64>
<+52>: lea    0x4(%rsp),%rbx
<+57>: lea    0x18(%rsp),%rbp
<+62>: jmp    0x400f87 <phase_2+27>
<+64>: add    $0x28,%rsp
<+68>: pop    %rbx
<+69>: pop    %rbp
<+70>: retq
```

2. 可以发现调用了 read_six_numbers，意思估计是读入六个整数，查看其调用的参数，发现确实是六个%d。读完后将六个整数保存在%rsp+0 到%rsp+24 的地方
3. 观察<+14>，发现第一个数必须要是 1
4. 观察<+41>到<+62>，发现这段代码执行 5 次，第 i 次%rbx 指向第 i 个整数
5. 观察<+27>到<+36>，发现每次检查后一个数是否等于前一个数的两倍

【Phase_3】

1. 获得汇编代码

```
<+0>:     sub    $0x18,%rsp
<+4>:     lea    0xc(%rsp),%r8
<+9>:     lea    0x7(%rsp),%rcx
<+14>:    lea    0x8(%rsp),%rdx
<+19>:    mov    $0x4026b6,%esi
<+24>:    mov    $0x0,%eax
<+29>:    callq  0x400c70 <__isoc99_sscanf@plt>
<+34>:    cmp    $0x2,%eax
<+37>:    jg     0x400fdf <phase_3+44>
<+39>:    callq  0x401698 <explode_bomb>
<+44>:    cmpl   $0x7,0x8(%rsp)
<+49>:    ja     0x4010dc <phase_3+297>
<+55>:    mov    0x8(%rsp),%eax
<+59>:    jmpq   *0x4026c0(,%rax,8)
<+66>:    mov    $0x62,%eax
<+71>:    cmpl   $0xc0,0xc(%rsp)
<+79>:    je     0x4010e6 <phase_3+307>
<+85>:    callq  0x401698 <explode_bomb>
……
<+307>:   cmp    0x7(%rsp),%al
<+311>:   je     0x4010f1 <phase_3+318>
<+313>:   callq  0x401698 <explode_bomb>
<+318>:   add    $0x18,%rsp
<+322>:   retq
```

2. 看到<+29>是读入，查看%esi 中的参数是%d %c %d，因此是按顺序读入三个参数：一个数字、一个字符、一个数字

3. 看到<+59>是一个 switch 跳转表的东西，根据%rax 来跳转，而根据<+44>到<+49>的内容得知%rax 的范围应该是 0~7。于是我让第一个参数为 0，则查看内存 0x4026c0+0*8 的内容，发现跳到<+66>

4. 观察<+66>到<+85>可以发现它是用来判断第三个参数是否等于 0xc0

5. 跳转到<+307>后，发现它判断第二个参数是否等于%al，而%al 在<+66>处被赋

值为 0x62=98。由于第二个参数是字符，因此得到 ASCII 码为 98 的字符 b，因此最终答案为（由于第一个参数，答案其实并不唯一）

```
0 b 192
```

【Phase_4】

1. 获得汇编代码

```
<+0>:  sub    $0x18,%rsp
<+4>:  lea    0xc(%rsp),%rcx
<+9>:  lea    0x8(%rsp),%rdx
<+14>: mov    $0x402955,%esi
<+19>: mov    $0x0,%eax
<+24>: callq  0x400c70 <__isoc99_sscanf@plt>
<+29>: cmp    $0x2,%eax
<+32>: jne    0x40115d <phase_4+41>
<+34>: cmpl   $0xe,0x8(%rsp)
<+39>: jbe    0x401162 <phase_4+46>
<+41>: callq  0x401698 <explode_bomb>
<+46>: mov    $0xe,%edx
<+51>: mov    $0x0,%esi
<+56>: mov    0x8(%rsp),%edi
<+60>: callq  0x4010f6 <func4>
<+65>: cmp    $0x3,%eax
<+68>: jne    0x401181 <phase_4+77>
<+70>: cmpl   $0x3,0xc(%rsp)
<+75>: je     0x401186 <phase_4+82>
<+77>: callq  0x401698 <explode_bomb>
<+82>: add    $0x18,%rsp
<+86>: retq
```

2. 看到<+24>是读入，查看%esi 后发现是读入两个整数
3. 观察<+34>，发现第一个整数不能>14
4. 观察<+70>到<+75>，发现第二个整数=3
5. 观察<+46>到<+68>，将一系列参数传入后运行 func4 函数，其结果需要=3，并且这些参数中包含第一个整数。获得 func4 的汇编代码

```
<+0>:  sub    $0x8,%rsp
<+4>:  mov    %edx,%eax
<+6>:  sub    %esi,%eax
<+8>:  mov    %eax,%ecx
<+10>: shr    $0x1f,%ecx
<+13>: add    %ecx,%eax
<+15>: sar    %eax
<+17>: lea    (%rax,%rsi,1),%ecx
<+20>: cmp    %edi,%ecx
<+22>: jle    0x40111a <func4+36>
<+24>: lea    -0x1(%rcx),%edx
<+27>: callq  0x4010f6 <func4>
<+32>: add    %eax,%eax
<+34>: jmp    0x40112f <func4+57>
<+36>: mov    $0x0,%eax
<+41>: cmp    %edi,%ecx
<+43>: jge    0x40112f <func4+57>
<+45>: lea    0x1(%rcx),%esi
<+48>: callq  0x4010f6 <func4>
<+53>: lea    0x1(%rax,%rax,1),%eax
<+57>: add    $0x8,%rsp
<+61>: retq
```

6. 函数 func4 是一个很无聊的函数，还有递归，因此我将它写成 C 代码形式

```
void func4() {
    rax = rdx; rax -= rsi; rcx = rax;
    rcx = (rax >= 0) ? 0 : 1;
    rax += rcx; rax >>= 1; rcx = rax + rsi;
    if (rcx <= rdi) {
        rax = 0;
        if (rcx >= rdi) return; else {
            rsi = rcx + 1;
            func4();
            rax = rax * 2 + 1;
            return;
        }} else {
        rdx = rcx - 1;
        func4();
        rax *= 2;
        return;
    }
}
```

通过令 rdx=14，rsi=0，枚举 rdi，查看每个 rax 的返回

```
rdi = 0   rax = 0
rdi = 1   rax = 0
rdi = 2   rax = 4
rdi = 3   rax = 0
rdi = 4   rax = 2
rdi = 5   rax = 2
rdi = 6   rax = 6
rdi = 7   rax = 0
rdi = 8   rax = 1
rdi = 9   rax = 1
rdi = 10  rax = 5
rdi = 11  rax = 1
rdi = 12  rax = 3
rdi = 13  rax = 3
rdi = 14  rax = 7
rdi = 15  rax = 15
```

7. 只有 rdi=12 或 13，即输入第一个整数为 12 或 13 时才成功，因此最终答案是

```
12 3
```

【Phase_5】

1. 获得汇编代码

```
<+0>:  push   %rbx
<+1>:  mov    %rdi,%rbx
<+4>:  callq  0x4013eb <string_length>
<+9>:  cmp    $0x6,%eax
<+12>: je     0x40119e <phase_5+19>
<+14>: callq  0x401698 <explode_bomb>
<+19>: mov    $0x0,%eax
<+24>: mov    $0x0,%edx
<+29>: movzbl (%rbx,%rax,1),%ecx
<+33>: and    $0xf,%ecx
<+36>: add    0x402700(,%rcx,4),%edx
<+43>: add    $0x1,%rax
<+47>: cmp    $0x6,%rax
<+51>: jne    0x4011a8 <phase_5+29>
<+53>: cmp    $0x30,%edx
```

```
<+56>: je     0x4011ca <phase_5+63>
<+58>: callq  0x401698 <explode_bomb>
<+63>: pop    %rbx
<+64>: retq
```

2. 观察<+4>到<+12>可发现读入是一个字符串，且猜测 string_length 函数的功能，字符串长度应该恰好是 6
3. 观察<+29>到<+53>，发现%rax 从 1 枚举到 6；每次%rbx+%rax*4 的内容放入%rcx ,%rcx 则是字符串的每个字符 ;每次%edx 都累加 0x402700+(%rcx mod 16)*4 的值，最后判断%edx 累加和是否等于 48
4. 打印 0x402700 处的数字（因为每次%rcx 要模 16 因此总共只有 16 个数字有用）

```
2 10 6 1 12 16 9 3 4 7 14 5 11 8 15 13
```

5. 也就是说要挑 6 个数字使得它们加起来等于 48，那我干脆全挑 8。8 在第 13 位，字母的 ASCII 码模 16=13 的有 m，因此我构造的答案是

```
mmmmmm
```

【Phase_6】

1. 获得汇编代码

```
<+0>:     push   %r14
<+2>:     push   %r13
<+4>:     push   %r12
<+6>:     push   %rbp
<+7>:     push   %rbx
<+8>:     sub    $0x50,%rsp
<+12>:    mov    %rsp,%r13
<+15>:    mov    %rsp,%rsi
<+18>:    callq  0x4016ce <read_six_numbers>
<+23>:    mov    %rsp,%r14
<+26>:    mov    $0x0,%r12d
<+32>:    mov    %r13,%rbp
<+35>:    mov    0x0(%r13),%eax
```

```
<+39>:    sub    $0x1,%eax
<+42>:    cmp    $0x5,%eax
<+45>:    jbe    0x401200 <phase_6+52>
<+47>:    callq  0x401698 <explode_bomb>
<+52>:    add    $0x1,%r12d
<+56>:    cmp    $0x6,%r12d
<+60>:    je     0x40122b <phase_6+95>
<+62>:    mov    %r12d,%ebx
<+65>:    movslq %ebx,%rax
<+68>:    mov    (%rsp,%rax,4),%eax
<+71>:    cmp    %eax,0x0(%rbp)
<+74>:    jne    0x40121d <phase_6+81>
<+76>:    callq  0x401698 <explode_bomb>
<+81>:    add    $0x1,%ebx
<+84>:    cmp    $0x5,%ebx
<+87>:    jle    0x40120d <phase_6+65>
<+89>:    add    $0x4,%r13
<+93>:    jmp    0x4011ec <phase_6+32>
```

2. 调用了函数 read_six_number，猜测应该读入六个整数
3. 在<+93>有跳到<+32>，这应该是个循环，通过<+60>可知%r12d 是迭代器，当其等于 6 时退出；同时在<+87>有跳到<+65>，这是个小循环，通过<+84>可知%ebx 是迭代器，当起大于 5 时退出
4. 观察<+32>到<+47>以及<+89>，每次%r13 指向下一个读入的整数，%eax 取出这个数，减 1 判断是否小于等于 5；即判断读入的 6 个整数是否都在 1~6 之间
5. 观察<+65>到<+76>，每次枚举%ebx，取出%rbp 指向数字时候的数字，依次和%rbp 指向的数字比较，若相同则爆炸。即这两个大小循环在判断读入的 6 个数是否两两不同。因此综合上一点读入的 6 个整数应该是一个 1~6 的排列
6. 再看接下来的代码

```
<+95>:    lea    0x18(%rsp),%rsi
<+100>:   mov    %r14,%rax
<+103>:   mov    $0x7,%ecx
<+108>:   mov    %ecx,%edx
```

```
<+110>:    sub    (%rax),%edx
<+112>:    mov    %edx,(%rax)
<+114>:    add    $0x4,%rax
<+118>:    cmp    %rsi,%rax
<+121>:    jne    0x401238 <phase_6+108>
<+123>:    mov    $0x0,%esi
<+128>:    jmp    0x40126f <phase_6+163>
```

7. 观察可发现，%rax 是循环枚举了这六个整数，每次通过%ecx=7 去减它们，所以这六个数最终都变成被 7 减完的余数

8. 观察<+130>到<+181>这段代码中，根据%rsp 中排列，分别给数字 1~6 分配 0x604300~0x604350，放在%rsp+0x20 之后的内存中，每次隔 8 个字节，相当于一个 Node。每个 Node 都有自己的 next，形成一个链表。每个 Node 还有数据内容

```
0x604300  0000 0001 0000 00d0
0x604310  0000 0002 0000 00de
0x604320  0000 0003 0000 0237
0x604330  0000 0004 0000 0058
0x604340  0000 0005 0000 00be
0x604350  0000 0006 0000 0036
```

9. 观察<+183>到<+257>这段，其中将 Node 按数据内容的后一个 byte 排序，于是最终答案就是通过排序的逆过程构造出原来的顺序，再将答案用 7 减

```
4 5 6 2 3 1
```

【Secret_phase】

1. 通过 objdump 工具获得整个汇编代码，在其中找到了 secret_phase 这个函数，应该就是传说中的隐藏关卡。通过查找其他函数的代码，在 phase_defused 函数（在某个关卡完成以后执行）下发现有进入 secret_phase 的入口。于是将 phase_defused 反汇编出来

```
<+39>:    lea    0x10(%rsp),%r8
<+44>:    lea    0xc(%rsp),%rcx
<+49>:    lea    0x8(%rsp),%rdx
<+54>:    mov    $0x40299f,%esi
<+59>:    mov    $0x604cb0,%edi
<+64>:    mov    $0x0,%eax
<+69>:    callq  0x400c70 <__isoc99_sscanf@plt>
<+74>:    cmp    $0x3,%eax
<+77>:    jne    0x4018b7 <phase_defused+128>
<+79>:    mov    $0x4029a8,%esi
<+84>:    lea    0x10(%rsp),%rdi
<+89>:    callq  0x401408 <strings_not_equal>
<+94>:    test   %eax,%eax
<+96>:    jne    0x4018b7 <phase_defused+128>
<+98>:    mov    $0x402800,%edi
<+103>:   callq  0x400b80 <puts@plt>
<+108>:   mov    $0x402828,%edi
<+113>:   callq  0x400b80 <puts@plt>
<+118>:   mov    $0x0,%eax
<+123>:   callq  0x40131a <secret_phase>
```

2. 查看 sscanf 的内容发现 phase_4 实际上可以读三个参数，第三个参数是字符串。

   看到后面有 strings_not_equal 函数 ,应该是判断输入字符串和默认字符串是否相

   同。根据<+79>查看内存 0x4029a8，发现字符串为

```
EastTk
```

   因此第四关完整的答案应该是

```
12 3 EastTk
```

3. 进入 secret_phase，查看汇编代码，主要内容有

```
<+42>: mov    %ebx,%esi
<+44>: mov    $0x604120,%edi
<+49>: callq  0x4012dc <fun7>
<+54>: cmp    $0x7,%eax
<+57>: je     0x40135a <secret_phase+64>
```

   即进入 fun7 函数让其返回值等于 7

4. 查看 fun7 函数的汇编代码，发现其又是一个无聊的递归，翻译成 C 代码

```
void fun7() {
    if (rdi == 0) {
        rax = 0xffffffff; return;
    }
    rdx = *rdi;
    if (rdx <= rsi) {
        rax = 0;
        if (rsi == rdx) return; else {
            rdi = *(rdi + 0x10);
            fun7();
            rax = rax * 2 + 1;
            return;
        }
    } else {
        rdi = *rdi + 8;
        fun7();
        rax *= 2;
        return;
    }
}
```

5. 每次%rax 都乘 2 或乘 2 加 1，初始值为 0，因此最终答案要是 7，%rax 必须经过的轨迹是 0→1→3→7，每次都经过乘 2 加 1 这一步，即每次都执行%rdx=*rdi 以及%rdi=*(%rdi+0x10)。而边界条件是%rsi==%rdi。因此通过三次%rdx=*rdi 以及%rdi=*(%rdi+0x10)操作后查看%rdi 的值，就是输入%rsi

6. 最终答案为

```
1001
```

【完整答案】

```
All your base are belong to us.
1 2 4 8 16 32
0 b 192
12 3 EastTk
mmmmmm
4 5 6 2 3 1
1001
```