《计算机组成原理实验》

实验报告

（实验一）

学 院 名 称 ：	数据科学与计算机学院
专业（班级）：	18 计教学 3 班
学 生 姓 名 ：	余傲泰
学　　号	18340199
时　　间 ：	2019　年　10　月　11　日

成 绩 ：

实验一 ： X86汇编基础二进制炸弹

一. 实验目的

1. 初步认识 X86 汇编语言；
2. 掌握阅读程序反汇编代码的方法，了解程序在机器上运行的实质；
3. 熟悉 Linux 环境、 掌握调试器 gdb 和反汇编工具 objdump 的使用。

二. 实验内容

使用课程知识拆除一个“Binary Bomb”（简称炸弹）来增强对程序的机器级表示、汇编语言、调试器和逆向工程等理解。 二进制炸弹是一个 Linux 可执行 C 程序，包含 phase_1~phase_6 共 6 个阶段和一个隐藏阶段 secret_phase。每人将获得一个唯一且每位同学差异化的炸弹程序。炸弹运行各阶段要求输入一个字符串，若输入符合程序预期，该阶段炸弹被“拆除”，否则“爆炸”。实验目标是需要拆除尽可能多的炸弹。

每个炸弹阶段考察机器级语言程序不同方面，难度递增。

阶段 1：字符串比较；

阶段 2：循环；

阶段 3：条件/分支：含 switch 语句；

阶段 4：递归调用和栈；

阶段 5：指针；

阶段 6：链表/指针/结构；

隐藏阶段，第 4 阶段的之后附加一特定字符串后才会出现。

拆弹技术：为了完成二进制炸弹拆除任务，需要：

1.使用 gdb 调试器和 objdump 反汇编工具。

2.单步跟踪调试每一阶段的机器代码。

3.理解汇编语言代码的行为或作用。

4.进而设法“推断”出拆除炸弹所需的目标字符串。

5.需要在每一阶段的开始代码前和引爆炸弹的函数前设置断点，便于调试。

三. 实验器材

1.PC机一台。

2.装有Linux操作系统的虚拟机一套。

3.炸弹文件（内容如下：）

（1）bomb：bomb 的可执行程序。

（2）bomb.c：bomb 程序的 main 函数。

（3）README：用文本编辑器打开即可查看其内容。

四. 实验过程与结果

说明：根据需要书写相关内容，如：

程序流程图、设计的思想与方法、分析、实验步骤和实验结果及分析等。

4.0 准备工作

（1）先用 objump 反汇编工具，将反汇编代码存入 dump.asm 文件中。

（2）查看 bomb.c 文件，记录 phase_1~phase_6 的位置，分别为：74、82、89、95、101、108。

（3）用 gdb 打开 bomb 程序，设置上述断点。

（4）新建 ans.txt 用于存放每一关的答案。

4.1 第一关

（1）找到第一关的反汇编代码所在位置：08048b5a <phase_1> 。

（2）查看 <phase_1> 发现调用了 <strings_not_equal> 函数，从函数名猜测这是一个判断字符串是否相同的函数，并且程序将一个固定地址 0x804a244 传入该函数。

```
8048b60:   68 44 a2 04 08          push   $0x804a244
8048b65:   ff 75 08                pushl  0x8(%ebp)
8048b68:   e8 eb 04 00 00          call   8049058 <strings_not_equal>
```

（3）用 gdb 查看地址 0x804a244 的内容，发现正是一个字符串，所以我猜测第一关的密码就是这个字符串。

```
(gdb) x/s 0x804a244
0x804a244:      "And they have no disregard for human life."
```

（4）运行 bomb 程序，并输入上面的字符串。

```
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
And they have no disregard for human life.
Phase 1 defused. How about the next one?
```

成功拆除第一关炸弹。

4.2 第二关

（1）找到第二关的反汇编代码所在位置：08048b7d <phase_2> 。

（2）查看 <phase_2> 发现调用了 <read_six_numbers> 函数，从函数名猜测这是一个读入 6 个数字的函数。

```
8048b96:   e8 31 07 00 00          call   80492cc <read_six_numbers>
```

（3）查看 <read_six_numbers> 发现调用了 <__isoc99_sscanf@plt> 函数，这应该是 C 语言里面的 sscanf 函数，是输入函数，并且程序将一个固定地址 0x804a4c5 传入 sscanf 里面。

```
80492ea:   68 c5 a4 04 08          push   $0x804a4c5
80492ef:   ff 75 08                pushl  0x8(%ebp)
80492f2:   e8 19 f5 ff ff          call   8048810 <__isoc99_sscanf@plt>
```

（4）用 gdb 查看 0x804a4c5 的内容如下：

```
(gdb) x/s 0x804a4c5
0x804a4c5:      "%d %d %d %d %d %d"
```

由此可见，第二关的密码应该是 6 个整数，整数之间用空格隔开。

（5）先在程序中随便输入“1 2 3 4 5 6”，然后继续查看反汇编代码，寄存器 ebx 的值被初始化为 1，然后跳转到 8048bba。

```
8048ba4:   bb 01 00 00 00          mov    $0x1,%ebx
8048ba9:   eb 0f                   jmp    8048bba <phase_2+0x3d>
```

（6）查看 8048bba 所在位置，发现在此之后如果不引爆炸弹的话，会跳转回 8048bb2，所以判断这是一个循环语句。如果 ebx = 6 则跳转到 8048bd2，否则如果不引爆炸弹的话，会跳转回 8048bb2，从 ebx 的变化规律可以看出循环次数是 6-1=5 次。因此，我们的目的应该是程序在这 5 次循环里面不引爆炸弹，可以安全跳转到 8048bd2。

```
8048bb2:   83 c3 01              add    $0x1,%ebx
8048bb5:   83 fb 06              cmp    $0x6,%ebx
8048bb8:   74 18                 je     8048bd2 <phase_2+0x55>

8048bba:   8b 44 9d d8           mov    -0x28(%ebp,%ebx,4),%eax
8048bbe:   89 45 d4              mov    %eax,-0x2c(%ebp)
8048bc1:   89 d9                 mov    %ebx,%ecx
8048bc3:   d3 e0                 shl    %cl,%eax
8048bc5:   39 44 9d dc           cmp    %eax,-0x24(%ebp,%ebx,4)
8048bc9:   74 e7                 je     8048bb2 <phase_2+0x35>

8048bcb:   e8 bc 06 00 00        call   804928c <explode_bomb>
8048bd0:   eb e0                 jmp    8048bb2 <phase_2+0x35>

8048bd2:   8b 45 f4              mov    -0xc(%ebp),%eax
```

（7）注意到 ebx=1 时，有如下指令：

```
8048bba:   8b 44 9d d8           mov    -0x28(%ebp,%ebx,4),%eax
```

这是将 ebp-ebx*4-0x28 间接寻址的值赋值给 eax，此时便是 ebp-0x24，用 gdb 查看之后发现，ebp-0x24 ~ ebp-0x10 便是存放我们输入的 6 个整数。

```
(gdb) x/6d $ebp-0x24
0xbffff5d4:     1       2       3       4
0xbffff5e4:     5       6
```

（8）再次查看反汇编代码：

当 ebx=1 时，eax 里为我们输入的第 1 个数；

当 ebx=2 时，eax 里为我们输入的第 2 个数；

……

当 ebx=6 时，退出循环。

可以知道这个循环在遍历我们输入的前 5 个数，赋值给 eax。同时在循环体内，ecx 的值与 ebx 是相等的，cl 此时就是 ecx。程序将 eax 里的值左移 cl 位。

```
8048bc3:   d3 e0                 shl    %cl,%eax
```

然后比较 eax 和 ebp+ebx*4-0x24 间接寻址的值，从上文可以知道，这是比较我们输入的第 i 个数左移 cl 位后与第 i+1 个数的大小，不相等就引爆炸弹。

```
8048bc5:   39 44 9d dc           cmp    %eax,-0x24(%ebp,%ebx,4)
8048bc9:   74 e7                 je     8048bb2 <phase_2+0x35>

8048bcb:   e8 bc 06 00 00        call   804928c <explode_bomb>
```

（9）由于 cl 的值与 ebx 是相等的，ebx 是从 1~5 的，所以我们可以写出大致的

代码来模拟，如下：

```
void phase_2(int a[])
{
    for(int i=0; i<5; i++)
    {
        if(a[i]<<i == a[i+1]) continue;
        else explode_bomb();
    }
}
```

因此，输入的 6 个数字应该满足：第 i+1 个数是第 i 个数左移 i 位的结果。

于是，可以得到很多组数组，它们的通项公式为：

$$A \quad A*2 \quad A*2^3 \quad A*2^6 \quad A*2^{10} \quad A*2^{15}$$

取 A=1 可以得到其中的一组“1 2 8 64 1024 32768”

（10）运行 bomb 程序，在第二关输入“1 2 8 64 1024 32768”。

```
Phase 1 defused. How about the next one?
1 2 8 64 1024 32768
That's number 2.  Keep going!
```

成功拆除第二关炸弹。

4.3 第三关

（1）找到第三关的反汇编代码所在位置：<phase_3>

（2）查看<phase_3>发现调用了<__isoc99_sscanf@plt>函数即 sscanf 函数，同时将一个固定地址 0x804a296 传入 sscanf 函数。

```
8048c05:   68 96 a2 04 08          push   $0x804a296
8048c0a:   ff 75 08                pushl  0x8(%ebp)
8048c0d:   e8 fe fb ff ff          call   8048810 <__isoc99_sscanf@plt>
```

（3）用 gdb 查看 0x804a296 的内容如下：

```
(gdb) x/s 0x804a296
0x804a296:      "%d %c %d"
```

由此可知第三关的密码是“整数 字符 整数”的形式。

（4）ebp-0x14 间接寻址的值便是我们输入的第一个整数，如果大于 7 则跳转到 8048cd5。

```
8048c1a:   83 7d ec 07             cmpl   $0x7,-0x14(%ebp)
8048c1e:   0f 87 b1 00 00 00       ja     8048cd5 <phase_3+0xed>
```

往后查找发现 8048cd5 执行的语句是引爆炸弹，所以第一个整数要小于或等于 7。

```
8048cd5:   e8 b2 05 00 00          call   804928c <explode_bomb>
```

（5）接着，输入的第一个整数被存入 eax 中，根据 eax 的值进行跳转，是典型的 switch-case 语句。

```
8048c24:   8b 45 ec                mov    -0x14(%ebp),%eax
8048c27:   ff 24 85 a0 a2 04 08    jmp    *0x804a2a0(,%eax,4)
```

（6）用 gdb 查看 eax=0~7 时对应跳转的地址如下：

```
(gdb) x/d *0x804a2a0
0x8048c35 <phase_3+77>: 31416
(gdb) x/d *0x804a2a4
0x8048cd5 <phase_3+237>:        373480
(gdb) x/d *0x804a2a8
0x8048c56 <phase_3+110>:        30904
(gdb) x/d *0x804a2ac
0x8048cd5 <phase_3+237>:        373480
(gdb) x/d *0x804a2b0
0x8048c6d <phase_3+133>:        25016
(gdb) x/d *0x804a2b4
0x8048c87 <phase_3+159>:        29112
(gdb) x/d *0x804a2b8
0x8048ca1 <phase_3+185>:        27320
(gdb) x/d *0x804a2bc
0x8048cbb <phase_3+211>:        27576
```

（7）以 eax=2 为例，此时程序跳转到 8048c56。将 0x78（即十进制的 120）赋值给 eax，之后判断 ebp-0x10 间接寻址即输入的第二个整数是否等于 0xfffffff2（即十进制的 -14），若相等则跳转到 8048cdf，不等则引爆炸弹。由此可知若第一个数字输入 2，那么第二个数字应该是 -14。

```
8048c56:   b8 78 00 00 00          mov    $0x78,%eax
8048c5b:   83 7d f0 f2             cmpl   $0xfffffff2,-0x10(%ebp)
8048c5f:   74 7e                   je     8048cdf <phase_3+0xf7>
8048c61:   e8 26 06 00 00          call   804928c <explode_bomb>
8048c66:   b8 78 00 00 00          mov    $0x78,%eax
8048c6b:   eb 72                   jmp    8048cdf <phase_3+0xf7>
```

（8）跳转到 8048cdf 后，判断 al 与 ebp-0x15 间接寻址（即输入的字符）是否相等。由于 al 此时便是 eax 的值 120，而要做判断的是一个字符，所以应该是将这个字符的 ASCII 码与 120 比较，如果相等就跳转到 8048ce9，不等就引爆炸弹。字符‘x’的 ASCII 码是 120，所以判断输入的字符为‘x’。

```
8048cdf:   38 45 eb                cmp    %al,-0x15(%ebp)
8048ce2:   74 05                   je     8048ce9 <phase_3+0x101>
8048ce4:   e8 a3 05 00 00          call   804928c <explode_bomb>
```

（9）综上可知，第三关的密码为“2 x -14”。运行 bomb 程序，在第三关输入该密码，如下：

```
That's number 2.  Keep going!
2 x -14
Halfway there!
```

成功拆除第三关的炸弹。

（10）当 eax 为 0~7 其他值时，程序运行过程类似，我们可以用代码模拟这个 switch-case 语句：

```
void phase_3(int num1, char c, int num2)
{
    switch (num1)
    {
    case 0:
        {
            num1 = 0x7a;
            if(num2 == 0x228 && num1 == c) return ;
            else explode_bomb();
            break;
        }
    case 1:
        {
            explode_bomb();
            break;
        }
    case 2:
        {
            num1 = 0x78;
            if(num2 == 0xfffffff2 && num1 == c) return ;
            else explode_bomb();
            break;
        }
    case 3:
        {
            explode_bomb();
            break;
        }
    case 4:
        {
            num1 = 0x61;
            if(num2 == 0x1d8 && num1 == c) return ;
            else explode_bomb();
            break;
        }
    case 5:
        {
            num1 = 0x71;
            if(num2 == 0x23d && num1 == c) return ;
            else explode_bomb();
            break;
        }
    case 6:
        {
            num1 = 0x6a;
            if(num2 == 0x439 && num1 == c) return ;
            else explode_bomb();
            break;
        }
    case 7:
        {
            num1 = 0x6b;
            if(num2 == 0x285 && num1 == c) return ;
            else explode_bomb();
            break;
        }
    }
}
```

4.4 第四关

（1）找到第四关的反汇编代码所在位置：<phase_4>

（2）同样调用了 sscanf 函数。

```
8048d6d:   68 d1 a4 04 08          push   $0x804a4d1
8048d72:   ff 75 08                pushl  0x8(%ebp)
8048d75:   e8 96 fa ff ff          call   8048810 <__isoc99_sscanf@plt>
```

（3）用 gdb 查看 0x804a4d1 的内容如下：

```
(gdb) x/s 0x804a4d1
0x804a4d1:      "%d %d"
```

由此可知，第四关的密码是 2 个整数，中间用空格隔开。

（4）ebp-0x14 间接寻址的值便是输入的第一个整数，如果小于或等于 0xe（即十进制的 14）就跳转到 8048d8d，否则引爆炸弹。所以输入的第一个整数要小于或等于 14。

```
8048d82:   83 7d ec 0e             cmpl   $0xe,-0x14(%ebp)
8048d86:   76 05                   jbe    8048d8d <phase_4+0x39>
8048d88:   e8 ff 04 00 00          call   804928c <explode_bomb>
```

（5）跳转后，程序将 0xe 0x0 和输入的第一个整数传入函数<func4>。

```
8048d90:   6a 0e                   push   $0xe
8048d92:   6a 00                   push   $0x0
8048d94:   ff 75 ec                pushl  -0x14(%ebp)
8048d97:   e8 60 ff ff ff          call   8048cfc <func4>
```

（4）查看<func4>后发现是一个递归函数。

```
8048d19:   39 c3                   cmp    %eax,%ebx
8048d1b:   7f 0b                   jg     8048d28 <func4+0x2c>
8048d1d:   39 c3                   cmp    %eax,%ebx
8048d1f:   7c 1c                   jl     8048d3d <func4+0x41>
```

当 eax < ebx 时，跳转到 8048d28，接着再次调用<func4>递归。

```
8048d28:   83 ec 04                sub    $0x4,%esp
8048d2b:   8d 4b ff                lea    -0x1(%ebx),%ecx
8048d2e:   51                      push   %ecx
8048d2f:   52                      push   %edx
8048d30:   50                      push   %eax
8048d31:   e8 c6 ff ff ff          call   8048cfc <func4>
```

当 eax > ebx 时，跳转到 8048d3d，接着再次调用<func4>递归。

```
8048d3d:   83 ec 04                sub    $0x4,%esp
8048d40:   ff 75 10                pushl  0x10(%ebp)
8048d43:   8d 53 01                lea    0x1(%ebx),%edx
8048d46:   52                      push   %edx
8048d47:   50                      push   %eax
8048d48:   e8 af ff ff ff          call   8048cfc <func4>
```

所以，退出递归的条件是 eax=ebx。

（5）分析反汇编代码可以知道这是 [0,14] 的一个二分查找的函数，输入的第一个整数就是要寻找的数字，用代码模拟<func4>函数如下：

```
int func4(int start, int end, int key)
{
    int mid = (start + end) / 2;
    if(key < mid)
    {
        return mid + func4(start, mid, key);
    }else if(key > mid)
    {
        return mid + func4(mid+1, end, key);
    }
    return mid;
}
```

（6）再次查看<phase_4>，寻找拆除炸弹的要点。

```
8048d9f:   83 f8 13              cmp    $0x13,%eax
8048da2:   75 06                 jne    8048daa <phase_4+0x56>
8048da4:   83 7d f0 13           cmpl   $0x13,-0x10(%ebp)
8048da8:   74 05                 je     8048daf <phase_4+0x5b>
8048daa:   e8 dd 04 00 00        call   804928c <explode_bomb>
8048daf:   8b 45 f4              mov    -0xc(%ebp),%eax
```

发现调用<func4>之后，如果 eax=0x13（即十进制的19）则不引爆炸弹，否则爆炸。而 eax 的值便是<func4>的返回值。所以我们的目的是在 [0,14] 中找到一个数字，使得<func4>的返回值是 19。回顾（5）中的<func4>可知，当输入为 4 时，返回值为 19。所以第四关的密码为“4 19”。

（7）运行 bomb 程序，在第四关输入“4 19”。

```
Halfway there!
4 19
So you got that one.  Try this one.
```

成功拆除第四关炸弹。

4.5 第五关

（1）找到第五关的反汇编代码所在位置：<phase_5>

（2）程序调用了<string_length>函数，从函数名猜测是计算字符串长度的函数。

```
8048dcd:   e8 64 02 00 00        call   8049036 <string_length>
```

如果字符串的长度不等于 6 则跳转到 8048e07。

```
8048dd5:    83 f8 06                cmp    $0x6,%eax
8048dd8:    75 2d                   jne    8048e07 <phase_5+0x45>
```

而 8048e07 是引爆炸弹，所以输入的字符串长度要为 6。

```
8048e07:    e8 80 04 00 00          call   804928c <explode_bomb>
```

（3）接着又进入一个循环，遍历输入的字符串，每个字符对应的 ASCII 码与 0xf 按位与运算，便是取低4位，所以 edx 的范围是 [0,15]，共 16 个。

```
8048de4:    0f b6 10                movzbl (%eax),%edx
8048de7:    83 e2 0f                and    $0xf,%edx
8048dea:    03 0c 95 c0 a2 04 08    add    0x804a2c0(,%edx,4),%ecx
8048df1:    83 c0 01                add    $0x1,%eax
8048df4:    39 d8                   cmp    %ebx,%eax
8048df6:    75 ec                   jne    8048de4 <phase_5+0x22>
```

（4）根据 edx 的值，取出数组中的某个值，让 ecx 加上这个值，用 gdb 查看从 0x804a2c0 开始的 16 个数，如下：

```
(gdb) x/16d 0x804a2c0
0x804a2c0 <array.3034>: 2       10      6       1
0x804a2d0 <array.3034+16>:      12      16      9       3
0x804a2e0 <array.3034+32>:      4       7       14      5
0x804a2f0 <array.3034+48>:      11      8       15      13
```

（5）查看反汇编代码，如果 ecx=0x44（即十进制的 68）则拆除炸弹，否则爆炸，所以我们的目标是让 ecx=68。

```
8048df8:    83 f9 44                cmp    $0x44,%ecx
8048dfb:    74 05                   je     8048e02 <phase_5+0x40>
8048dfd:    e8 8a 04 00 00          call   804928c <explode_bomb>

8048e02:    8b 5d fc                mov    -0x4(%ebp),%ebx
8048e05:    c9                      leave
8048e06:    c3                      ret
```

（6）用代码模拟这个过程如下：

```
int phase_5(string s)
{
    int num[16]={2, 10, 6, 1, 12, 16, 9, 3, 4, 7, 14, 5, 11, 8, 15, 13};
    if(s.size() != 6) explode_bomb();
    int ecx = 0;
    for(int i=0; i<6; i++)
    {
        ecx += num[ s[i] & 0xf ];
    }
    return ecx;
}
```

（7）根据（4）中查找到的数组，我选择了 68=11*5+13*1。所以要有 5 个字符的 ASCII 码低四位为 1100，1 个字符的 ASCII 码低四位为 1111，我选择了 5 个‘1’（ACSII 码为 108,低四位为 1100)和 1 个‘o’(ASCII 码为 111,低四位为 1111)。

（8）运行 bomb 程序，在第五关输入“llllo”。

```
So you got that one.  Try this one.
llllo
Good work!  On to the next...
```

成功拆除第五关炸弹。

4.6 第六关

（1）找到第六关的反汇编代码所在位置：<phase_6>

（2）调用了<read_six_numbers>函数，从第二关可知，我们应该输入 6 个整数，整数之间用空格隔开。

```
8048e28:   e8 9f 04 00 00          call   80492cc <read_six_numbers>
```

（3）接着进入一个嵌套的 2 层循环：

```
8048e41:   83 c6 01                add    $0x1,%esi
8048e44:   83 fe 06                cmp    $0x6,%esi
8048e47:   74 51                   je     8048e9a <phase_6+0x8c>
8048e49:   89 f3                   mov    %esi,%ebx
8048e4b:   eb 0f                   jmp    8048e5c <phase_6+0x4e>

8048e4d:   e8 3a 04 00 00          call   804928c <explode_bomb>
8048e52:   eb ed                   jmp    8048e41 <phase_6+0x33>

8048e54:   83 c3 01                add    $0x1,%ebx
8048e57:   83 fb 05                cmp    $0x5,%ebx
8048e5a:   7f d9                   jg     8048e35 <phase_6+0x27>

8048e5c:   8b 44 9d c4             mov    -0x3c(%ebp,%ebx,4),%eax
8048e60:   39 44 b5 c0             cmp    %eax,-0x40(%ebp,%esi,4)
8048e64:   75 ee                   jne    8048e54 <phase_6+0x46>
8048e66:   e8 21 04 00 00          call   804928c <explode_bomb>
```

这个循环遍历了输入的 6 个数，主要在做 2 件事，第一是判断输入的数 -1 是否大于 5，实际上就是判断是否大于 6，如果大于 6 则爆炸；第二是判断这 6 个数字是否有重复，如果有重复就爆炸。因此，可以推断输入的 6 个数是 1~6 的某个排列。

（4）下面的操作有点迷乱，我看不太懂。。。翻来覆去看到了一个固定地址：

```
8048e8e:   ba 54 c1 04 08          mov    $0x804c154,%edx
```

出于好奇心，就用 gdb 看了一下这个东西里面是什么。

```
(gdb) x/20d 0x804c154
0x804c154 <node1>:      737     1       134529376       72
0x804c164 <node2+4>:    2       134529388       721     3
0x804c174 <node3+8>:    134529400       189     4       134529412
0x804c184 <node5>:      147     5       134529424       688
0x804c194 <node6+4>:    6       0       0       875771953
```

然后就发现这是一个带有 6 个节点的链表：

	node1	node2	node3	node4	node5	node6
value1	737	72	721	189	147	688
value2	1	2	3	4	5	6

但第六关到底想做什么还不清楚，于是我往后看，不引爆炸弹的条件是什么。

```
8048ec9:   be 05 00 00 00          mov    $0x5,%esi
8048ece:   eb 08                   jmp    8048ed8 <phase_6+0xca>

8048ed0:   8b 5b 08                mov    0x8(%ebx),%ebx
8048ed3:   83 ee 01                sub    $0x1,%esi
8048ed6:   74 10                   je     8048ee8 <phase_6+0xda>

8048ed8:   8b 43 08                mov    0x8(%ebx),%eax
8048edb:   8b 00                   mov    (%eax),%eax
8048edd:   39 03                   cmp    %eax,(%ebx)
8048edf:   7e ef                   jle    8048ed0 <phase_6+0xc2>
8048ee1:   e8 a6 03 00 00          call   804928c <explode_bomb>
8048ee6:   eb e8                   jmp    8048ed0 <phase_6+0xc2>
```

最后有一个循环，从 esi 的变化来看，循环次数是 5。每次循环后，ebx 会往后增大 0x8。ebx+0x8 间接寻址赋值给 eax，然后比较 eax 的值和 ebx 简介寻址的值。推测这是比较 6 个数中相邻的两个数之间的大小，如果（ebx）<=eax 则跳转，否则爆炸。所以这 6 个数应该是升序排列，炸弹才不会爆炸。

（5）从（3）（4）可以看到一些“巧合”：

<1>输入的数是 1~6 的某个排列；

<2>有一个链表里面有 6 个节点，每个节点有 2 个值，其中一个和节点的标号一样；

<3>最后比较的是 6 个数的大小，要求升序排列。

这几点都和“6”有关，所以猜测我们输入的 6 个数，是对应这个链表的 6 个节点，而且要通过这个数，重新排列这个链表使其变成升序排列。

在（3）中整理出来的表格中，可以看到“72<147<189<688<721<737”，对应的 6

个节点标号的排列是“2 5 4 6 3 1”。

（6）运行 bomb 程序，输入“2 5 4 6 3 1”。

```
Good work!  On to the next...
2 5 4 6 3 1
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
```

成功拆除第六关炸弹（虽然我还没完全搞懂）。

4.7 隐藏关

（1）拆完前面 6 个炸弹之后，程序就结束了，但是听说有隐藏关，于是重新查看反汇编代码：

```
08048f54 <secret_phase>:
```

发现真的有。

（2）于是查看哪里会跳转到 08048f54 这里，然后在<phase_defused>函数里面找到了这一条调用信息：

```
80494c2:   e8 8d fa ff ff          call   8048f54 <secret_phase>
```

所以在每一关通过之后，都有可能进入隐藏关，那么应该有一把“钥匙”决定了能不能进入。

（3）在<phase_defused>里发现又调用了 sscanf 函数：

```
804945c:   68 2b a5 04 08          push   $0x804a52b
8049461:   68 f0 c8 04 08          push   $0x804c8f0
8049466:   e8 a5 f3 ff ff          call   8048810 <__isoc99_sscanf@plt>
```

（4）在 gdb 里查看 0x804a52b 和 0x804c8f0。

```
(gdb) x/s 0x804a52b
0x804a52b:      "%d %d %s"
(gdb) x/s 0x804c8f0
0x804c8f0 <input_strings+240>:  ""
```

（5）所以有一关的输入格式为“整数 整数 字符串”便可以进入隐藏关，下面的空字符串是因为我没有输入字符串，所以为空，不能进入隐藏关。回想之前的关卡，发现要输入 2 个整数的只有第四关，所以这个字符串应该加在第四关的密码后面。

（6）回看反汇编代码发现调用了<strings_not_equal>函数，同时传入 0x804a534，由第一关可知，这是判断字符串是否相等的函数。

```
8049494:   68 34 a5 04 08          push   $0x804a534
8049499:   8d 45 a4                lea    -0x5c(%ebp),%eax
804949c:   50                      push   %eax
804949d:   e8 b6 fb ff ff          call   8049058 <strings_not_equal>
```

（7）用 gdb 查看 0x804a534 的内容如下：

```
(gdb) x/s 0x804a534
0x804a534:      "SecretSYSU"
```

由此可见，在第四关后面输入的字符串应该是“SecretSYSU”。

（8）在 ans.txt 里面的第四关密码后，加入该字符串，运行 bomb 程序，将 ans.txt 传入，结果如下：

```
sysu@debian:~/bomb42$ ./bomb ans.txt
Welcome to my fiendish little bomb. 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...
Curses, you've found the secret phase!
But finding it and solving it are quite different...
```

成功进入了隐藏关！但是还要找到它的答案。

（9）隐藏关调用<fun7>，同时传入 0x804c0a0。

```
8048f7f:   53                      push   %ebx
8048f80:   68 a0 c0 04 08          push   $0x804c0a0
8048f85:   e8 76 ff ff ff          call   8048f00 <fun7>
```

用 gdb 查看 0x804c0a0，如下

```
(gdb) x/d 0x804c0a0
0x804c0a0 <n1>: 36
(gdb) x/s 0x804c0a0
0x804c0a0 <n1>: "$"
```

不清楚是 36 还是 “$”，于是多查看之后几个，如下：

```
(gdb) x/24d 0x804c0a0
0x804c0a0 <n1>: 36      134529196       134529208       8
0x804c0b0 <n21+4>:      134529244       134529220       50      134529232
0x804c0c0 <n22+8>:      134529256       22      134529328       134529304
0x804c0d0 <n33>:        45      134529268       134529340       6
0x804c0e0 <n31+4>:      134529280       134529316       107     134529292
0x804c0f0 <n34+8>:      134529352       40      0       0
```

在一些小数字中间穿插了很多大数字，但是大数字看起来很接近，有点像地址，所以用 16 进制查看，如下：

```
(gdb) x/24x 0x804c0a0
0x804c0a0 <n1>: 0x00000024      0x0804c0ac      0x0804c0b8      0x00000008
0x804c0b0 <n21+4>:      0x0804c0dc      0x0804c0c4      0x00000032      0x0804c0d0
0x804c0c0 <n22+8>:      0x0804c0e8      0x00000016      0x0804c130      0x0804c118
0x804c0d0 <n33>:        0x0000002d      0x0804c0f4      0x0804c13c      0x00000006
0x804c0e0 <n31+4>:      0x0804c100      0x0804c124      0x0000006b      0x0804c10c
0x804c0f0 <n34+8>:      0x0804c148      0x00000028      0x00000000      0x00000000
```

发现这些大数字其实是指向这附近的地址，而且一个小数字跟着 2 个大数字，所以猜测这是一颗二叉树。

（10）查看<fun7>发现又是一个递归函数，用代码模拟如下：

```
void fun7(int& eax, int key, list* head)
{
    eax = 0;
    if(head == NULL) return ;
    if(head->value == key) return ;
    if(head->value > key)
    {
        eax *= 2;
        fun7(eax, key, head->left);
    }
    if(head->value < key)
    {
        eax += 1;
        fun7(eax, key, head->right);
    }
}
```

（11）炸弹不爆炸的条件时 eax=2，于是这就是我们的目标。

```
8048f8d:    83 f8 02                cmp    $0x2,%eax
8048f90:    74 05                   je     8048f97 <secret_phase+0x43>
8048f92:    e8 f5 02 00 00          call   804928c <explode_bomb>
```

由（10）可知，只要往右子树移动 2 次就行了，根据（9）中各个节点的地址，可以知道，向右子树移动 2 次后对应的值是 22，所以这就是隐藏关的密码。

（12）运行 bomb 程序，隐藏关处输入 22，如下：

```
Curses, you've found the secret phase!
But finding it and solving it are quite different...
22
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
```

成功拆除隐藏关炸弹。

五. 实验心得

（1）实验前对 x86 指令完全不熟悉，所以花了 2 天看了 x86 的各种指令，在网上找了很多资料，然后回忆实验课上助教是如何演示第一关的。然后渐渐熟悉了，如何判断数与数的大小，如何跳转，怎么查看地址，怎么查看值。

（2）但是实际操作的时候，经常会把直接寻址与间接寻址搞错，所以饶了很多弯路，比如查看值的时候，喜欢在地址前面加“$”，然后就看到一堆“void”，怀疑人生，后来干脆就都试一遍，现在渐渐熟悉了。

（3）开始使用 gdb 的时候很不舒服，因为我只能看到下面一条指令是什么，但是没什么大作用，后来在网上查到可以用“display /10i $pc”查看即将运行的 10 条指令，这样调试起来就有了很大的进展，知道后面几步是要干什么，不会说一不小心就跳过了。

（4）刚开始在虚拟机上操作，很生疏，不断地切换 gdb 和 vim 调试和查看反汇编代码，后来知道可以把反汇编代码传到主机上，这样一边看反汇编代码，一边调试就很方便，查找值、看跳转到哪一步就很方便。

（5）很重要的一点就是要有耐心，看反汇编代码，特别是跳转指令多的时候，很容易混乱，然后就暴躁，这次实验也稍微提高了我的耐心，让人更能沉得住气了。

（6）最后感谢助教和老师的教导，还有同学的帮助，让我成功完成实验（=v=）。

【程序代码】

```
08048b5a <phase_1>:
 8048b5a:   55                      push   %ebp
 8048b5b:   89 e5                   mov    %esp,%ebp
 8048b5d:   83 ec 10                sub    $0x10,%esp
 8048b60:   68 44 a2 04 08          push   $0x804a244
 8048b65:   ff 75 08                pushl  0x8(%ebp)
 8048b68:   e8 eb 04 00 00          call   8049058 <strings_not_equal>
 8048b6d:   83 c4 10                add    $0x10,%esp
 8048b70:   85 c0                   test   %eax,%eax
 8048b72:   75 02                   jne    8048b76 <phase_1+0x1c>
 8048b74:   c9                      leave
 8048b75:   c3                      ret
 8048b76:   e8 11 07 00 00          call   804928c <explode_bomb>
 8048b7b:   eb f7                   jmp    8048b74 <phase_1+0x1a>

08048b7d <phase_2>:
```

```
 8048b7d:   55                      push   %ebp
 8048b7e:   89 e5                   mov    %esp,%ebp
 8048b80:   53                      push   %ebx
 8048b81:   83 ec 3c                sub    $0x3c,%esp
 8048b84:   65 a1 14 00 00 00       mov    %gs:0x14,%eax
 8048b8a:   89 45 f4                mov    %eax,-0xc(%ebp)
 8048b8d:   31 c0                   xor    %eax,%eax
 8048b8f:   8d 45 dc                lea    -0x24(%ebp),%eax
 8048b92:   50                      push   %eax
 8048b93:   ff 75 08                pushl  0x8(%ebp)
 8048b96:   e8 31 07 00 00          call   80492cc <read_six_numbers>       ;读入6个
数字
 8048b9b:   83 c4 10                add    $0x10,%esp
 8048b9e:   83 7d dc 05             cmpl   $0x5,-0x24(%ebp)
 8048ba2:   77 07                   ja     8048bab <phase_2+0x2e>           ;8048ba
b

 8048ba4:   bb 01 00 00 00          mov    $0x1,%ebx                        ;ebx =
1
 8048ba9:   eb 0f                   jmp    8048bba <phase_2+0x3d>           ;8048bb
a

 8048bab:   e8 dc 06 00 00          call   804928c <explode_bomb>           ;boom!
 8048bb0:   eb f2                   jmp    8048ba4 <phase_2+0x27>
;loop
 8048bb2:   83 c3 01                add    $0x1,%ebx                        ;ebx +=
 1
 8048bb5:   83 fb 06                cmp    $0x6,%ebx                        ;比较ebx
与6
 8048bb8:   74 18                   je     8048bd2 <phase_2+0x55>           ;出循环

 8048bba:   8b 44 9d d8             mov    -0x28(%ebp,%ebx,4),%eax          ;eax =
(ebp + 4*ebx - 0x28)
 8048bbe:   89 45 d4                mov    %eax,-0x2c(%ebp)                 ;eax =
(ebp - 0x2c)
 8048bc1:   89 d9                   mov    %ebx,%ecx                        ;ecx =
ebx
 8048bc3:   d3 e0                   shl    %cl,%eax                         ;eax左移
cl位
 8048bc5:   39 44 9d dc             cmp    %eax,-0x24(%ebp,%ebx,4)
 8048bc9:   74 e7                   je     8048bb2 <phase_2+0x35>           ;回到405
行循环
```

```
 8048bcb:   e8 bc 06 00 00          call   804928c <explode_bomb>           ;boom!
 8048bd0:   eb e0                   jmp    8048bb2 <phase_2+0x35>

 8048bd2:   8b 45 f4                mov    -0xc(%ebp),%eax
 8048bd5:   65 33 05 14 00 00 00    xor    %gs:0x14,%eax
 8048bdc:   75 05                   jne    8048be3 <phase_2+0x66>
 8048bde:   8b 5d fc                mov    -0x4(%ebp),%ebx
 8048be1:   c9                      leave
 8048be2:   c3                      ret
 8048be3:   e8 a8 fb ff ff          call   8048790 <__stack_chk_fail@plt>

08048be8 <phase_3>:
 8048be8:   55                      push   %ebp
 8048be9:   89 e5                   mov    %esp,%ebp
 8048beb:   83 ec 24                sub    $0x24,%esp
 8048bee:   65 a1 14 00 00 00       mov    %gs:0x14,%eax
 8048bf4:   89 45 f4                mov    %eax,-0xc(%ebp)
 8048bf7:   31 c0                   xor    %eax,%eax                        ;eax =
0
 8048bf9:   8d 45 f0                lea    -0x10(%ebp),%eax
 8048bfc:   50                      push   %eax
 8048bfd:   8d 45 eb                lea    -0x15(%ebp),%eax
 8048c00:   50                      push   %eax
 8048c01:   8d 45 ec                lea    -0x14(%ebp),%eax
 8048c04:   50                      push   %eax
 8048c05:   68 96 a2 04 08          push   $0x804a296
 8048c0a:   ff 75 08                pushl  0x8(%ebp)
 8048c0d:   e8 fe fb ff ff          call   8048810 <__isoc99_sscanf@plt>
 8048c12:   83 c4 20                add    $0x20,%esp
 8048c15:   83 f8 02                cmp    $0x2,%eax                        ;exa <=
 2 boom!
 8048c18:   7e 14                   jle    8048c2e <phase_3+0x46>           ;8048c2
e boom!

 8048c1a:   83 7d ec 07             cmpl   $0x7,-0x14(%ebp)                 ;参数1大
于7时boom!
 8048c1e:   0f 87 b1 00 00 00       ja     8048cd5 <phase_3+0xed>           ;8048cd
5 boom!
 8048c24:   8b 45 ec                mov    -0x14(%ebp),%eax
 8048c27:   ff 24 85 a0 a2 04 08    jmp    *0x804a2a0(,%eax,4)
 /*
```

```
 eax==0 -> 8048c35
 eax==1 -> 8048cd5 boom!
 eax==2 -> 8048c56
 eax==3 -> 8048cd5 boom!
 eax==4 -> 8048c6d
 eax==5 -> 8048c87
 eax==6 -> 8048ca1
 eax==7 -> 8048cbb
 */

 8048c2e:   e8 59 06 00 00          call   804928c <explode_bomb>           ;boom!
 8048c33:   eb e5                   jmp    8048c1a <phase_3+0x32>           ;回到447
行循环

 8048c35:   b8 7a 00 00 00          mov    $0x7a,%eax                       ;eax =
122
 8048c3a:   81 7d f0 28 02 00 00    cmpl   $0x228,-0x10(%ebp)               ;552与参
数3
 8048c41:   0f 84 98 00 00 00       je     8048cdf <phase_3+0xf7>
 8048c47:   e8 40 06 00 00          call   804928c <explode_bomb>           ;boom!
 8048c4c:   b8 7a 00 00 00          mov    $0x7a,%eax
 8048c51:   e9 89 00 00 00          jmp    8048cdf <phase_3+0xf7>

 8048c56:   b8 78 00 00 00          mov    $0x78,%eax                       ;eax =
120
 8048c5b:   83 7d f0 f2             cmpl   $0xfffffff2,-0x10(%ebp)          ;-14与参
数3
 8048c5f:   74 7e                   je     8048cdf <phase_3+0xf7>
 8048c61:   e8 26 06 00 00          call   804928c <explode_bomb>           ;boom!
 8048c66:   b8 78 00 00 00          mov    $0x78,%eax
 8048c6b:   eb 72                   jmp    8048cdf <phase_3+0xf7>

 8048c6d:   b8 61 00 00 00          mov    $0x61,%eax                       ;eax =
97
 8048c72:   81 7d f0 d8 01 00 00    cmpl   $0x1d8,-0x10(%ebp)               ;472与参
数3
 8048c79:   74 64                   je     8048cdf <phase_3+0xf7>
 8048c7b:   e8 0c 06 00 00          call   804928c <explode_bomb>           ;boom!
 8048c80:   b8 61 00 00 00          mov    $0x61,%eax
 8048c85:   eb 58                   jmp    8048cdf <phase_3+0xf7>
```

```
 8048c87:   b8 71 00 00 00          mov    $0x71,%eax                  ;eax =
113
 8048c8c:   81 7d f0 3d 02 00 00    cmpl   $0x23d,-0x10(%ebp)          ;573 与参
数 3
 8048c93:   74 4a                   je     8048cdf <phase_3+0xf7>
 8048c95:   e8 f2 05 00 00          call   804928c <explode_bomb>      ;boom!
 8048c9a:   b8 71 00 00 00          mov    $0x71,%eax
 8048c9f:   eb 3e                   jmp    8048cdf <phase_3+0xf7>

 8048ca1:   b8 6a 00 00 00          mov    $0x6a,%eax                  ;eax =
106
 8048ca6:   81 7d f0 39 04 00 00    cmpl   $0x439,-0x10(%ebp)          ;1081 与
参数 3
 8048cad:   74 30                   je     8048cdf <phase_3+0xf7>
 8048caf:   e8 d8 05 00 00          call   804928c <explode_bomb>      ;boom!
 8048cb4:   b8 6a 00 00 00          mov    $0x6a,%eax
 8048cb9:   eb 24                   jmp    8048cdf <phase_3+0xf7>

 8048cbb:   b8 6b 00 00 00          mov    $0x6b,%eax                  ;eax =
107
 8048cc0:   81 7d f0 85 02 00 00    cmpl   $0x285,-0x10(%ebp)          ;645 与参
数 3
 8048cc7:   74 16                   je     8048cdf <phase_3+0xf7>
 8048cc9:   e8 be 05 00 00          call   804928c <explode_bomb>      ;boom!
 8048cce:   b8 6b 00 00 00          mov    $0x6b,%eax
 8048cd3:   eb 0a                   jmp    8048cdf <phase_3+0xf7>

 8048cd5:   e8 b2 05 00 00          call   804928c <explode_bomb>      ;boom!
 8048cda:   b8 69 00 00 00          mov    $0x69,%eax

 8048cdf:   38 45 eb                cmp    %al,-0x15(%ebp)
 8048ce2:   74 05                   je     8048ce9 <phase_3+0x101>
 8048ce4:   e8 a3 05 00 00          call   804928c <explode_bomb>      ;boom!
 8048ce9:   8b 45 f4                mov    -0xc(%ebp),%eax
 8048cec:   65 33 05 14 00 00 00    xor    %gs:0x14,%eax
 8048cf3:   75 02                   jne    8048cf7 <phase_3+0x10f>     ;exit
 8048cf5:   c9                      leave
 8048cf6:   c3                      ret
 8048cf7:   e8 94 fa ff ff          call   8048790 <__stack_chk_fail@plt>

08048cfc <func4>:
 8048cfc:   55                      push   %ebp
```

```
8048cfd:   89 e5                mov    %esp,%ebp
8048cff:   53                   push   %ebx
8048d00:   83 ec 04             sub    $0x4,%esp
8048d03:   8b 45 08             mov    0x8(%ebp),%eax
8048d06:   8b 55 0c             mov    0xc(%ebp),%edx
8048d09:   8b 4d 10             mov    0x10(%ebp),%ecx
8048d0c:   29 d1                sub    %edx,%ecx
8048d0e:   89 cb                mov    %ecx,%ebx
8048d10:   c1 eb 1f             shr    $0x1f,%ebx
8048d13:   01 cb                add    %ecx,%ebx
8048d15:   d1 fb                sar    %ebx
8048d17:   01 d3                add    %edx,%ebx
8048d19:   39 c3                cmp    %eax,%ebx                ;if ebx
> eax -> 544 行
8048d1b:   7f 0b                jg     8048d28 <func4+0x2c>
8048d1d:   39 c3                cmp    %eax,%ebx                ;if ebx
< eax -> 554 行
8048d1f:   7c 1c                jl     8048d3d <func4+0x41>

8048d21:   89 d8                mov    %ebx,%eax                ;eax =
ebx
8048d23:   8b 5d fc             mov    -0x4(%ebp),%ebx          ;ebx =
-0x4(ebp)
8048d26:   c9                   leave
8048d27:   c3                   ret

8048d28:   83 ec 04             sub    $0x4,%esp
8048d2b:   8d 4b ff             lea    -0x1(%ebx),%ecx
8048d2e:   51                   push   %ecx
8048d2f:   52                   push   %edx
8048d30:   50                   push   %eax
8048d31:   e8 c6 ff ff ff       call   8048cfc <func4>          ;->521
行递归
8048d36:   83 c4 10             add    $0x10,%esp
8048d39:   01 c3                add    %eax,%ebx
8048d3b:   eb e4                jmp    8048d21 <func4+0x25>     ;->539
行

8048d3d:   83 ec 04             sub    $0x4,%esp
8048d40:   ff 75 10             pushl  0x10(%ebp)
8048d43:   8d 53 01             lea    0x1(%ebx),%edx
8048d46:   52                   push   %edx
```

```
 8048d47:   50                      push   %eax
 8048d48:   e8 af ff ff ff          call   8048cfc <func4>                 ;->521
行递归
 8048d4d:   83 c4 10                add    $0x10,%esp
 8048d50:   01 c3                   add    %eax,%ebx
 8048d52:   eb cd                   jmp    8048d21 <func4+0x25>            ;->539
行

08048d54 <phase_4>:
 8048d54:   55                      push   %ebp
 8048d55:   89 e5                   mov    %esp,%ebp
 8048d57:   83 ec 18                sub    $0x18,%esp
 8048d5a:   65 a1 14 00 00 00       mov    %gs:0x14,%eax
 8048d60:   89 45 f4                mov    %eax,-0xc(%ebp)
 8048d63:   31 c0                   xor    %eax,%eax
 8048d65:   8d 45 f0                lea    -0x10(%ebp),%eax
 8048d68:   50                      push   %eax
 8048d69:   8d 45 ec                lea    -0x14(%ebp),%eax
 8048d6c:   50                      push   %eax
 8048d6d:   68 d1 a4 04 08          push   $0x804a4d1
 8048d72:   ff 75 08                pushl  0x8(%ebp)
 8048d75:   e8 96 fa ff ff          call   8048810 <__isoc99_sscanf@plt>
 8048d7a:   83 c4 10                add    $0x10,%esp
 8048d7d:   83 f8 02                cmp    $0x2,%eax                       ;eax !=
 2 boom!
 8048d80:   75 06                   jne    8048d88 <phase_4+0x34>          ;boom!
 8048d82:   83 7d ec 0e             cmpl   $0xe,-0x14(%ebp)                ;参数
1 <= 14 时跳转
 8048d86:   76 05                   jbe    8048d8d <phase_4+0x39>
 8048d88:   e8 ff 04 00 00          call   804928c <explode_bomb>          ;boom!

 8048d8d:   83 ec 04                sub    $0x4,%esp
 8048d90:   6a 0e                   push   $0xe
 8048d92:   6a 00                   push   $0x0
 8048d94:   ff 75 ec                pushl  -0x14(%ebp)
 8048d97:   e8 60 ff ff ff          call   8048cfc <func4>                 ;func4
 8048d9c:   83 c4 10                add    $0x10,%esp
 8048d9f:   83 f8 13                cmp    $0x13,%eax
 8048da2:   75 06                   jne    8048daa <phase_4+0x56>          ;boom!
 8048da4:   83 7d f0 13             cmpl   $0x13,-0x10(%ebp)
 8048da8:   74 05                   je     8048daf <phase_4+0x5b>
 8048daa:   e8 dd 04 00 00          call   804928c <explode_bomb>          ;boom!
```

```
 8048daf:   8b 45 f4                mov    -0xc(%ebp),%eax
 8048db2:   65 33 05 14 00 00 00    xor    %gs:0x14,%eax
 8048db9:   75 02                   jne    8048dbd <phase_4+0x69>
 8048dbb:   c9                      leave
 8048dbc:   c3                      ret
 8048dbd:   e8 ce f9 ff ff          call   8048790 <__stack_chk_fail@plt>

08048dc2 <phase_5>:
 8048dc2:   55                      push   %ebp
 8048dc3:   89 e5                   mov    %esp,%ebp
 8048dc5:   53                      push   %ebx
 8048dc6:   83 ec 10                sub    $0x10,%esp
 8048dc9:   8b 5d 08                mov    0x8(%ebp),%ebx
 8048dcc:   53                      push   %ebx
 8048dcd:   e8 64 02 00 00          call   8049036 <string_length>
 8048dd2:   83 c4 10                add    $0x10,%esp
 8048dd5:   83 f8 06                cmp    $0x6,%eax
 8048dd8:   75 2d                   jne    8048e07 <phase_5+0x45>          ;boom!
 8048dda:   89 d8                   mov    %ebx,%eax
 8048ddc:   83 c3 06                add    $0x6,%ebx
 8048ddf:   b9 00 00 00 00          mov    $0x0,%ecx

 8048de4:   0f b6 10                movzbl (%eax),%edx
 8048de7:   83 e2 0f                and    $0xf,%edx
 8048dea:   03 0c 95 c0 a2 04 08    add    0x804a2c0(,%edx,4),%ecx
 8048df1:   83 c0 01                add    $0x1,%eax
 8048df4:   39 d8                   cmp    %ebx,%eax
 8048df6:   75 ec                   jne    8048de4 <phase_5+0x22>
 8048df8:   83 f9 44                cmp    $0x44,%ecx
 8048dfb:   74 05                   je     8048e02 <phase_5+0x40>          ;exit!
 8048dfd:   e8 8a 04 00 00          call   804928c <explode_bomb>          ;boom!

 8048e02:   8b 5d fc                mov    -0x4(%ebp),%ebx
 8048e05:   c9                      leave
 8048e06:   c3                      ret

 8048e07:   e8 80 04 00 00          call   804928c <explode_bomb>          ;boom!
 8048e0c:   eb cc                   jmp    8048dda <phase_5+0x18>

08048e0e <phase_6>:
 8048e0e:   55                      push   %ebp
 8048e0f:   89 e5                   mov    %esp,%ebp
```

```
8048e11:   56                    push   %esi
8048e12:   53                    push   %ebx
8048e13:   83 ec 48              sub    $0x48,%esp
8048e16:   65 a1 14 00 00 00     mov    %gs:0x14,%eax
8048e1c:   89 45 f4              mov    %eax,-0xc(%ebp)
8048e1f:   31 c0                 xor    %eax,%eax
8048e21:   8d 45 c4              lea    -0x3c(%ebp),%eax
8048e24:   50                    push   %eax
8048e25:   ff 75 08              pushl  0x8(%ebp)
8048e28:   e8 9f 04 00 00        call   80492cc <read_six_numbers>       ;读入6个
数字
8048e2d:   83 c4 10              add    $0x10,%esp
8048e30:   be 00 00 00 00        mov    $0x0,%esi

8048e35:   8b 44 b5 c4           mov    -0x3c(%ebp,%esi,4),%eax
8048e39:   83 e8 01              sub    $0x1,%eax
8048e3c:   83 f8 05              cmp    $0x5,%eax
8048e3f:   77 0c                 ja     8048e4d <phase_6+0x3f>           ;boom!

8048e41:   83 c6 01              add    $0x1,%esi
8048e44:   83 fe 06              cmp    $0x6,%esi
8048e47:   74 51                 je     8048e9a <phase_6+0x8c>
8048e49:   89 f3                 mov    %esi,%ebx
8048e4b:   eb 0f                 jmp    8048e5c <phase_6+0x4e>

8048e4d:   e8 3a 04 00 00        call   804928c <explode_bomb>           ;boom!
8048e52:   eb ed                 jmp    8048e41 <phase_6+0x33>

8048e54:   83 c3 01              add    $0x1,%ebx
8048e57:   83 fb 05              cmp    $0x5,%ebx
8048e5a:   7f d9                 jg     8048e35 <phase_6+0x27>

8048e5c:   8b 44 9d c4           mov    -0x3c(%ebp,%ebx,4),%eax
8048e60:   39 44 b5 c0           cmp    %eax,-0x40(%ebp,%esi,4)
8048e64:   75 ee                 jne    8048e54 <phase_6+0x46>
8048e66:   e8 21 04 00 00        call   804928c <explode_bomb>           ;boom!
8048e6b:   eb e7                 jmp    8048e54 <phase_6+0x46>

8048e6d:   8b 52 08              mov    0x8(%edx),%edx
8048e70:   83 c0 01              add    $0x1,%eax
8048e73:   39 c8                 cmp    %ecx,%eax
8048e75:   75 f6                 jne    8048e6d <phase_6+0x5f>
```

```
8048e77:   89 54 b5 dc             mov    %edx,-0x24(%ebp,%esi,4)
8048e7b:   83 c3 01                add    $0x1,%ebx
8048e7e:   83 fb 06                cmp    $0x6,%ebx
8048e81:   74 1e                   je     8048ea1 <phase_6+0x93>

8048e83:   89 de                   mov    %ebx,%esi
8048e85:   8b 4c 9d c4             mov    -0x3c(%ebp,%ebx,4),%ecx
8048e89:   b8 01 00 00 00          mov    $0x1,%eax
8048e8e:   ba 54 c1 04 08          mov    $0x804c154,%edx
8048e93:   83 f9 01                cmp    $0x1,%ecx
8048e96:   7f d5                   jg     8048e6d <phase_6+0x5f>
8048e98:   eb dd                   jmp    8048e77 <phase_6+0x69>

8048e9a:   bb 00 00 00 00          mov    $0x0,%ebx
8048e9f:   eb e2                   jmp    8048e83 <phase_6+0x75>

8048ea1:   8b 5d dc                mov    -0x24(%ebp),%ebx
8048ea4:   8b 45 e0                mov    -0x20(%ebp),%eax
8048ea7:   89 43 08                mov    %eax,0x8(%ebx)
8048eaa:   8b 55 e4                mov    -0x1c(%ebp),%edx
8048ead:   89 50 08                mov    %edx,0x8(%eax)
8048eb0:   8b 45 e8                mov    -0x18(%ebp),%eax
8048eb3:   89 42 08                mov    %eax,0x8(%edx)
8048eb6:   8b 55 ec                mov    -0x14(%ebp),%edx
8048eb9:   89 50 08                mov    %edx,0x8(%eax)
8048ebc:   8b 45 f0                mov    -0x10(%ebp),%eax
8048ebf:   89 42 08                mov    %eax,0x8(%edx)
8048ec2:   c7 40 08 00 00 00 00    movl   $0x0,0x8(%eax)
8048ec9:   be 05 00 00 00          mov    $0x5,%esi
8048ece:   eb 08                   jmp    8048ed8 <phase_6+0xca>

8048ed0:   8b 5b 08                mov    0x8(%ebx),%ebx
8048ed3:   83 ee 01                sub    $0x1,%esi
8048ed6:   74 10                   je     8048ee8 <phase_6+0xda>

8048ed8:   8b 43 08                mov    0x8(%ebx),%eax
8048edb:   8b 00                   mov    (%eax),%eax
8048edd:   39 03                   cmp    %eax,(%ebx)
8048edf:   7e ef                   jle    8048ed0 <phase_6+0xc2>
8048ee1:   e8 a6 03 00 00          call   804928c <explode_bomb>          ;boom!
8048ee6:   eb e8                   jmp    8048ed0 <phase_6+0xc2>
```

```
 8048ee8:   8b 45 f4                mov    -0xc(%ebp),%eax
 8048eeb:   65 33 05 14 00 00 00    xor    %gs:0x14,%eax
 8048ef2:   75 07                   jne    8048efb <phase_6+0xed>          ;exit
 8048ef4:   8d 65 f8                lea    -0x8(%ebp),%esp
 8048ef7:   5b                      pop    %ebx
 8048ef8:   5e                      pop    %esi
 8048ef9:   5d                      pop    %ebp
 8048efa:   c3                      ret
 8048efb:   e8 90 f8 ff ff          call   8048790 <__stack_chk_fail@plt>

08048f00 <fun7>:
 8048f00:   55                      push   %ebp
 8048f01:   89 e5                   mov    %esp,%ebp
 8048f03:   53                      push   %ebx
 8048f04:   83 ec 04                sub    $0x4,%esp
 8048f07:   8b 55 08                mov    0x8(%ebp),%edx
 8048f0a:   8b 4d 0c                mov    0xc(%ebp),%ecx
 8048f0d:   85 d2                   test   %edx,%edx
 8048f0f:   74 3c                   je     8048f4d <fun7+0x4d>
 8048f11:   8b 1a                   mov    (%edx),%ebx
 8048f13:   39 cb                   cmp    %ecx,%ebx
 8048f15:   7f 0e                   jg     8048f25 <fun7+0x25>
 8048f17:   b8 00 00 00 00          mov    $0x0,%eax
 8048f1c:   39 cb                   cmp    %ecx,%ebx
 8048f1e:   75 18                   jne    8048f38 <fun7+0x38>

 8048f20:   8b 5d fc                mov    -0x4(%ebp),%ebx
 8048f23:   c9                      leave
 8048f24:   c3                      ret

 8048f25:   83 ec 08                sub    $0x8,%esp
 8048f28:   51                      push   %ecx
 8048f29:   ff 72 04                pushl  0x4(%edx)
 8048f2c:   e8 cf ff ff ff          call   8048f00 <fun7>
 8048f31:   83 c4 10                add    $0x10,%esp
 8048f34:   01 c0                   add    %eax,%eax
 8048f36:   eb e8                   jmp    8048f20 <fun7+0x20>
 8048f38:   83 ec 08                sub    $0x8,%esp
 8048f3b:   51                      push   %ecx
 8048f3c:   ff 72 08                pushl  0x8(%edx)
 8048f3f:   e8 bc ff ff ff          call   8048f00 <fun7>
```

```
 8048f44:   83 c4 10                add    $0x10,%esp
 8048f47:   8d 44 00 01             lea    0x1(%eax,%eax,1),%eax
 8048f4b:   eb d3                   jmp    8048f20 <fun7+0x20>

 8048f4d:   b8 ff ff ff ff          mov    $0xffffffff,%eax                ;eax =
-1
 8048f52:   eb cc                   jmp    8048f20 <fun7+0x20>

08048f54 <secret_phase>:
 8048f54:   55                      push   %ebp
 8048f55:   89 e5                   mov    %esp,%ebp
 8048f57:   53                      push   %ebx
 8048f58:   83 ec 04                sub    $0x4,%esp
 8048f5b:   e8 a6 03 00 00          call   8049306 <read_line>
 8048f60:   83 ec 04                sub    $0x4,%esp
 8048f63:   6a 0a                   push   $0xa
 8048f65:   6a 00                   push   $0x0
 8048f67:   50                      push   %eax
 8048f68:   e8 13 f9 ff ff          call   8048880 <strtol@plt>
 8048f6d:   89 c3                   mov    %eax,%ebx
 8048f6f:   8d 40 ff                lea    -0x1(%eax),%eax
 8048f72:   83 c4 10                add    $0x10,%esp
 8048f75:   3d e8 03 00 00          cmp    $0x3e8,%eax
 8048f7a:   77 35                   ja     8048fb1 <secret_phase+0x5d>      ;boom!
 8048f7c:   83 ec 08                sub    $0x8,%esp
 8048f7f:   53                      push   %ebx
 8048f80:   68 a0 c0 04 08          push   $0x804c0a0
 8048f85:   e8 76 ff ff ff          call   8048f00 <fun7>
 8048f8a:   83 c4 10                add    $0x10,%esp
 8048f8d:   83 f8 02                cmp    $0x2,%eax
 8048f90:   74 05                   je     8048f97 <secret_phase+0x43>
 8048f92:   e8 f5 02 00 00          call   804928c <explode_bomb>           ;boom!

 8048f97:   83 ec 0c                sub    $0xc,%esp
 8048f9a:   68 70 a2 04 08          push   $0x804a270
 8048f9f:   e8 1c f8 ff ff          call   80487c0 <puts@plt>
 8048fa4:   e8 6e 04 00 00          call   8049417 <phase_defused>
 8048fa9:   83 c4 10                add    $0x10,%esp
 8048fac:   8b 5d fc                mov    -0x4(%ebp),%ebx
 8048faf:   c9                      leave
 8048fb0:   c3                      ret
```

```
8048fb1:   e8 d6 02 00 00          call   804928c <explode_bomb>          ;boom!
8048fb6:   eb c4                   jmp    8048f7c <secret_phase+0x28>
```