汇编实践实验报告

实验名称：	二进制炸弹
姓　　名：	张绍磊
学　　号：	2016211392
班　　级：	2016211310
专　　业：	计算机科学与技术
指导教师：	周峰

2018 年　　7 月　　25 日

目录

一、实验目的

1.理解 C 语言程序的机器级表示

2.初步掌握 GDB 调试器的用法

3.阅读 C 编译器生成的 x86-64 机器代码，理解不同控制结构生成的基本指令模式，过程的实现。

二、实验环境

Window 10;

UltraEdit 二进制文件编辑器;

Visual Studio 2017;

Linux 服务器;

GDB 调试工具;

三、实验内容

输入学号和密码下载为你定制的可运行的 bomb，该程序是由 C 语言代码编译生成的，运行该程序要求你输入 6 个字符串，若输入错误则炸弹爆炸；要求阅读汇编代码，从而推测出所要求的字符串。

四、实验步骤

1. 输入学号和密码下载定制的可运行的 bomb。
2. 将.tar 文件上传至服务器 http://10.105.222.108:16310/。
3. 输入命令 tar -xvf bomb22.tar 将 bomb22.tar 中的文件减压到当前文件夹。
4. 输入命令 objdump -d bomb22 -d bomb22 > bomb22。
5. 将 bomb22.s 下载到本地。
6. 阅读.s 文件，分析推测程序要求输入的字符串格式。
7. 在服务器上，运行 bomb22，输入相关字符。
8. 判断测试结果是否正确。
9. 撰写实验报告。

五、实验原理

X86-84 寄存器含义：

GDB 调试方法：

```
gcc -g    main.c                          //在目标文件加入源代码的信息

(gdb) start                               //开始调试

(gdb) n                                   //一条一条执行

(gdb) step/s                              //执行下一条，如果函数进入函数

(gdb) backtrace/bt                        //查看函数调用栈帧

(gdb) info/i locals                       //查看当前栈帧局部变量
```

(gdb) frame/f //选择栈帧，再查看局部变量

(gdb) print/p //打印变量的值

(gdb) finish //运行到当前函数返回

(gdb) set var sum=0 //修改变量值

(gdb) list/l 行号或函数名 //列出源码

(gdb) display/undisplay sum //每次停下显示变量的值/取消跟踪

(gdb) break/b 行号或函数名 //设置断点

(gdb) continue/c //连续运行

(gdb) info/i breakpoints //查看已经设置的断点

(gdb) delete breakpoints 2 //删除某个断点

(gdb) disable/enable breakpoints 3 //禁用/启用某个断点

(gdb) break 9 if sum != 0 //满足条件才激活断点

(gdb) run/r //重新从程序开头连续执行

(gdb) watch input[4] //设置观察点

(gdb) info/i watchpoints //查看设置的观察点

(gdb) x/7b input //打印存储器内容，b--每个字节一组，7--7 组

(gdb) disassemble //反汇编当前函数或指定函数

(gdb) si // 一条指令一条指令调试 而 s 是一行一行代码

(gdb) info registers // 显示所有寄存器的当前值

(gdb) x/20 $esp //查看内存中开始的 20 个数

破解流程：

（1）反汇编成.s 文件；

（2）阅读源码，逐句翻译；

（3）对源码进行功能分析和逻辑分析；

（4）找到防止爆炸的密码；

（5）设置断点，开始测试。

六、破译过程

主函数分析

从主函数，得知程序一共分为 6 关，每关包含 read_line()、phase_1(input)和 phase_defused()三部分。

通过调用库函数 readline 读入一行数据，字符串地址通过 eax 返回，然后将字符串传给 phase_1 函数。同样，其后几个阶段遵循这种处理方式：读取一行字符串，调用 phase_n，传入字符串参数。

具体主函数如下所示：

```
#include <stdio.h>
#include <stdlib.h>
#include "support.h"
#include "phases.h"

FILE *infile;
```

```c
int main(int argc, char *argv[])
{
    char *input;

    /* Note to self: remember to port this bomb to Windows and put a
     * fantastic GUI on it. */

    /* When run with no arguments, the bomb reads its input lines
     * from standard input. */
    if (argc == 1) {
        infile = stdin;
    }


    else if (argc == 2) {
        if (!(infile = fopen(argv[1], "r"))) {
            printf("%s: Error: Couldn't open %s\n", argv[0], argv[1]);
            exit(8);
        }
    }

    /* You can't call the bomb with more than 1 command line argument. */
    else {
        printf("Usage: %s [<input_file>]\n", argv[0]);
        exit(8);
    }

    /* Do all sorts of secret stuff that makes the bomb harder to defuse.
*/
    initialize_bomb();

    printf("Welcome to my fiendish little bomb. You have 6 phases
with\n");
    printf("which to blow yourself up. Have a nice day!\n");

    /* Hmm...  Six phases must be more secure than one phase! */
    input = read_line();             /* Get input                   */
    phase_1(input);                  /* Run the phase               */
    phase_defused();                 /* Drat!  They figured it out!
                                      * Let me know how they did it. */
    printf("Phase 1 defused. How about the next one?\n");

    input = read_line();
    phase_2(input);
```

```
    phase_defused();
    printf("That's number 2.  Keep going!\n");

    input = read_line();
    phase_3(input);
    phase_defused();
    printf("Halfway there!\n");

    /* Oh yeah?  Well, how good is your math?  Try on this saucy problem!
*/
    input = read_line();
    phase_4(input);
    phase_defused();
    printf("So you got that one.  Try this one.\n");

    /* Round and 'round in memory we go, where we stop, the bomb blows!
*/
    input = read_line();
    phase_5(input);
    phase_defused();
    printf("Good work!  On to the next...\n");


    input = read_line();
    phase_6(input);
    phase_defused();

    return 0;
}
```

反汇编

在服务器，输入命令 objdump -d bomb22 -d bomb22 > bomb22，使二进制文件转化成.s 汇编文件，方便阅读理解代码。

拆除炸弹

1. Phase_1

（1）源代码

Phase_1 汇编源码如下：

```
329 ⊟0000000000400e80 <phase_1>:
330     400e80:  48 83 ec 08          sub    $0x8,%rsp
331     400e84:  be 60 24 40 00       mov    $0x402460,%esi
332     400e89:  e8 f0 03 00 00       callq  40127e <strings_not_equal>
333     400e8e:  85 c0                test   %eax,%eax
334     400e90:  74 05                je     400e97 <phase_1+0x17>
335     400e92:  e8 4d 06 00 00       callq  4014e4 <explode_bomb>
336     400e97:  48 83 c4 08          add    $0x8,%rsp
337     400e9b:  c3                   retq
```

（2）汇编语言翻译：

内存地址	汇编代码	含义
400e80:	sub $0x8,%rsp	对 rsp-0x8，保存回 rsp。
400e84:	mov $0x402460,%esi	将$0x402460 的值存入 esi 中。
400e89:	callq 40127e <strings_not_equal>	入口函数的地址 0x40127e，该函数要调用的参数为 esi（即 $0x402460 处的值），和 rsp（即我们的输入）。该函数的作用是 ： 判断字符串是否相等。
400e8e:	test %eax,%eax	函数返回值存在 eax 中，是判断字符串是否相等的结果。 这三句，如果 eax 不为 0，则引爆炸弹， 如果 eax 为 0，则跳转到 leave 语句函数结束。说明 eax 为 0 时，可以进入下一关
400e90:	je 400e97 <phase_1+0x17>	
400e92:	callq 4014e4 <explode_bomb>	
400e97:	add $0x8,%rsp	对 rsp+0x8，保存回 rsp。
400e9b:	retq	返回。

（3）源码分析

根据以上分析，phase_1 是从键盘输入一个值，放到 rsp-0x8 中，并将$0x402460 的值存入 esi 中。接着，程序对这两个参数进行函数调用，调用判断字符串是否相等的函数 <string not equal> 进行判断，如果二者相等，则返回值为 0，不引爆炸弹，反之，只要二者不相等，则炸弹爆炸。

故此处的密码存在地址$0x402460 中，我们只要查看该地址的值，即可找到密码。

通过 GDB，查找到$0x402460 中的值为“When a problem comes along, you must zip it!”。

答案：When a problem comes along, you must zip it!

2. Phase_2:

（1）源代码

Phase_2 汇编源码如下：

```
0000000000400e9c <phase_2>:
  400e9c:  55                   push   %rbp
  400e9d:  53                   push   %rbx
  400e9e:  48 83 ec 28          sub    $0x28,%rsp
  400ea2:  48 89 e6             mov    %rsp,%rsi
  400ea5:  e8 70 06 00 00       callq  40151a <read_six_numbers>
  400eaa:  83 3c 24 01          cmpl   $0x1,(%rsp)
  400eae:  74 20                je     400ed0 <phase_2+0x34>
  400eb0:  e8 2f 06 00 00       callq  4014e4 <explode_bomb>
  400eb5:  eb 19                jmp    400ed0 <phase_2+0x34>
  400eb7:  8b 43 fc             mov    -0x4(%rbx),%eax
  400eba:  01 c0                add    %eax,%eax
  400ebc:  39 03                cmp    %eax,(%rbx)
  400ebe:  74 05                je     400ec5 <phase_2+0x29>
  400ec0:  e8 1f 06 00 00       callq  4014e4 <explode_bomb>
  400ec5:  48 83 c3 04          add    $0x4,%rbx
  400ec9:  48 39 eb             cmp    %rbp,%rbx
  400ecc:  75 e9                jne    400eb7 <phase_2+0x1b>
  400ece:  eb 0c                jmp    400edc <phase_2+0x40>
  400ed0:  48 8d 5c 24 04       lea    0x4(%rsp),%rbx
  400ed5:  48 8d 6c 24 18       lea    0x18(%rsp),%rbp
  400eda:  eb db                jmp    400eb7 <phase_2+0x1b>
  400edc:  48 83 c4 28          add    $0x28,%rsp
  400ee0:  5b                   pop    %rbx
  400ee1:  5d                   pop    %rbp
  400ee2:  c3                   retq
```

（2）汇编语言翻译：

内存地址	汇编代码	含义
400e9c:	push %rbp	压栈%rbx、压栈%rbp，设置%rsp。
400e9d:	push %rbx	
400e9e:	sub $0x28,%rsp	
400ea2:	mov %rsp,%rsi	
400ea5:	callq 40151a <read_six_numbers>	入口函数的地址 0x40151a。该函数的作用是：输入六个数字，以上部分结束后，%rsp 即为输入的第一个数字的地址。
400eaa:	cmpl $0x1,(%rsp)	比较与 1 的大小。
400eae:	je 400ed0 <phase_2+0x34>	若等于 1，跳转。
400eb0:	callq 4014e4	爆炸。

	<explode_bomb>	
400eb5:	jmp 400ed0 <phase_2+0x34>	跳出。
400eb7:	mov -0x4(%rbx),%eax	%eax=%rbx-0x4。
400eba:	add %eax,%eax	当前数字乘 2。
400ebc:	cmp %eax,(%rbx)	判断是否等比
400ebe:	je 400ec5 <phase_2+0x29>	若相等，跳转。
400ec0:	callq 4014e4 <explode_bomb>	爆炸。
400ec5:	add $0x4,%rbx	%rbx=%rbx+4。
400ec9:	cmp %rbp,%rbx	判断是否跳出循环。
400ecc:	jne 400eb7 <phase_2+0x1b>	
400ece:	jmp 400edc <phase_2+0x40>	跳转至 400edc。
400ed0:	lea 0x4(%rsp),%rbx	指针后移 移到第二个数字。
400ed5:	lea 0x18(%rsp),%rbp	这里指向最后一个数。
400eda:	jmp 400eb7 <phase_2+0x1b>	跳转至 400eb7。
400edc:	add $0x28,%rsp	恢复%rsp，弹出%rbx、%rbp。
400ee0:	pop %rbx	
400ee1:	pop %rbp	
400ee2:	retq	返回。

（3）源码分析

根据汇编代码，得知需要输入六个数字，内存中输入的六个数字分布如下：

%rsp	%rsp+0x4	%rsp+0x8	%rsp+0xC	%rsp+0x10	%rsp+0x14
Number[0]	Number[1]	Number[2]	Number[3]	Number[4]	Number[5]

还原成 C 语言，phase_2()大致如下所示：

```
void phase_2()
{//Number in %rsp,Edge in %rbp,(%register)表示寻址得到的值
    if((%rsp)==1)   //保证第一个数是1
    {
        goto Label_400f30;
    }
Label_400f17:
    %eax=(%rbx-0x4);
    %eax=2*%eax;
    if((%rbx)!=%eax)   //保证后一个数为前一个数的两倍
    {
        explode_bomb();
    }
    %rbx=%rbx+0x4;
    if(%rbx==%rbp)
    {
        return;
    }
    else
    {
        goto Label_400f17;
    }
Label_400f30:
    %rbx=%rsp+0x4;
    %rbp=%rsp+0x18;
    goto Label_400f17;
}
```

故输入为首项为 1，公比为 2 的等比数列。

答案：1 2 4 8 16 32

3. Phase_3

（1）源代码

Phase_3 汇编源码如下：

```
0000000000400ee3 <phase_3>:
  400ee3:  48 83 ec 18          sub    $0x18,%rsp
  400ee7:  48 8d 4c 24 08       lea    0x8(%rsp),%rcx
  400eec:  48 8d 54 24 0c       lea    0xc(%rsp),%rdx
  400ef1:  be 8d 27 40 00       mov    $0x40278d,%esi
  400ef6:  b8 00 00 00 00       mov    $0x0,%eax
  400efb:  e8 a0 fc ff ff       callq  400ba0 <__isoc99_sscanf@plt>
  400f00:  83 f8 01             cmp    $0x1,%eax
  400f03:  7f 05                jg     400f0a <phase_3+0x27>
  400f05:  e8 da 05 00 00       callq  4014e4 <explode_bomb>
  400f0a:  83 7c 24 0c 07       cmpl   $0x7,0xc(%rsp)
  400f0f:  77 64                ja     400f75 <phase_3+0x92>
  400f11:  8b 44 24 0c          mov    0xc(%rsp),%eax
  400f15:  ff 24 c5 c0 24 40 00 jmpq   *0x4024c0(,%rax,8)
  400f1c:  b8 00 00 00 00       mov    $0x0,%eax
  400f21:  eb 05                jmp    400f28 <phase_3+0x45>
  400f23:  b8 e5 03 00 00       mov    $0x3e5,%eax
  400f28:  2d 1d 02 00 00       sub    $0x21d,%eax
  400f2d:  eb 05                jmp    400f34 <phase_3+0x51>
  400f2f:  b8 00 00 00 00       mov    $0x0,%eax
  400f34:  83 c0 77             add    $0x77,%eax
  400f37:  eb 05                jmp    400f3e <phase_3+0x5b>
  400f39:  b8 00 00 00 00       mov    $0x0,%eax
  400f3e:  2d e8 02 00 00       sub    $0x2e8,%eax
  400f43:  eb 05                jmp    400f4a <phase_3+0x67>
  400f45:  b8 00 00 00 00       mov    $0x0,%eax
  400f4a:  05 e8 02 00 00       add    $0x2e8,%eax
  400f4f:  eb 05                jmp    400f56 <phase_3+0x73>
  400f51:  b8 00 00 00 00       mov    $0x0,%eax
  400f56:  2d e8 02 00 00       sub    $0x2e8,%eax
  400f5b:  eb 05                jmp    400f62 <phase_3+0x7f>
  400f5d:  b8 00 00 00 00       mov    $0x0,%eax
  400f62:  05 e8 02 00 00       add    $0x2e8,%eax
  400f67:  eb 05                jmp    400f6e <phase_3+0x8b>
  400f69:  b8 00 00 00 00       mov    $0x0,%eax
  400f6e:  2d e8 02 00 00       sub    $0x2e8,%eax
  400f73:  eb 0a                jmp    400f7f <phase_3+0x9c>
  400f75:  e8 6a 05 00 00       callq  4014e4 <explode_bomb>
  400f7a:  b8 00 00 00 00       mov    $0x0,%eax
  400f7f:  83 7c 24 0c 05       cmpl   $0x5,0xc(%rsp)
  400f84:  7f 06                jg     400f8c <phase_3+0xa9>
  400f86:  3b 44 24 08          cmp    0x8(%rsp),%eax
  400f8a:  74 05                je     400f91 <phase_3+0xae>
  400f8c:  e8 53 05 00 00       callq  4014e4 <explode_bomb>
  400f91:  48 83 c4 18          add    $0x18,%rsp
  400f95:  c3                   retq
```

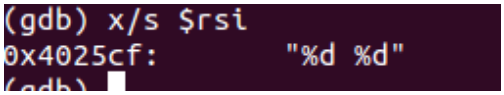
（2）汇编语言翻译：

内存地址	汇编代码	含义
400ee3:	sub $0x18,%rsp	设置%rsp，设置两个输入的地址。
400ee7:	lea 0x8(%rsp),%rcx	
400eec:	lea 0xc(%rsp),%rdx	
400ef1:	mov $0x40278d,%esi	
400ef6:	mov $0x0,%eax	
400efb:	callq 400ba0 <__isoc99_sscanf@plt>	输入两个整数。返回输入个数。
400f00:	cmp $0x1,%eax	若输入的个数小于等于 1，则爆炸。
400f03:	jg 400f0a <phase_3+0x27>	
400f05:	callq 4014e4 <explode_bomb>	
400f0a:	cmpl $0x7,0xc(%rsp)	将输入的第一个数和 7 比较，大于 7 则爆炸。
400f0f:	ja 400f75 <phase_3+0x92>	
400f11:	mov 0xc(%rsp),%eax	Switch-Case 选择跳转
400f15:	jmpq *0x4024c0(,%rax,8)	
400f1c:	mov $0x0,%eax	当 a=1 时，跳转至此。
400f21:	jmp 400f28 <phase_3+0x45>	
400f23:	mov $0x3e5,%eax	当 a=0 时，跳转至此。
400f28:	sub $0x21d,%eax	
400f2d:	jmp 400f34 <phase_3+0x51>	
400f2f:	mov $0x0,%eax	当 a=2 时，跳转至此。
400f34:	add $0x77,%eax	
400f37:	jmp 400f3e <phase_3+0x5b>	
400f39:	mov $0x0,%eax	当 a=3 时，跳转至此。
400f3e:	sub $0x2e8,%eax	
400f43:	jmp 400f4a <phase_3+0x67>	
400f45:	mov $0x0,%eax	当 a=4 时，跳转至此。
400f4a:	add $0x2e8,%eax	
400f4f:	jmp 400f56 <phase_3+0x73>	
400f51:	mov $0x0,%eax	
400f56:	sub $0x2e8,%eax	当 a=5 时，跳转至此。
400f5b:	jmp 400f62 <phase_3+0x7f>	
400f5d:	mov $0x0,%eax	当 a=6 时，跳转至此。
400f62:	add $0x2e8,%eax	

400f67:	jmp 400f6e <phase_3+0x8b>	
400f69:	mov $0x0,%eax	当 a=7 时，跳转至此。
400f6e:	sub $0x2e8,%eax	
400f73:	jmp 400f7f <phase_3+0x9c>	跳转至 400f7f。
400f75:	callq 4014e4 <explode_bomb>	爆炸。
400f7a:	mov $0x0,%eax	
400f7f:	cmpl $0x5,0xc(%rsp)	将输入的第一个数和 5 比较，大于 5 则爆炸。
400f84:	jg 400f8c <phase_3+0xa9>	
400f86:	cmp 0x8(%rsp),%eax	比较输入的第二个数和%eax，若不相等则爆炸，相等则成功。
400f8a:	je 400f91 <phase_3+0xae>	
400f8c:	callq 4014e4 <explode_bomb>	
400f91:	add $0x18,%rsp	恢复%rsp。
400f95:	retq	返回。

（3）源码分析

可以看出 sscanf 的具体作用，用 x/s $rsi 查看格式串，得出我们需要输入 2 个整数 a、b，通过 sscanf 分别读到%rsp + 0x8，%rsp + 0xc 中。



然后将输入的第一个数和 7 比较，大于 7 炸弹爆炸，大于 5 时炸弹爆炸。因此第一个数为 1-5 之间。之后通过 GDB 查看 1-5 之间的数字对于跳转位置的映射。

第一个数	条件跳转地址
0	0x 400F23
1	0x 400F1C
2	0x 400F2F
3	0x 400F39
4	0x 400F45

5	0x 400F51
6	0x 400F5D
7	0x 400F69

通过加减运算，若最终%eax 等于 b 则成功，否则爆炸。

以 a=5 为例，程序跳转至 400f51 处：0 - 0x2e8 + 0x2e8 - 0x2e8 = b。故 b=-0x2e8=-744。此题答案不唯一。

答案：5 -744

4. Phase_4

（1）源代码

Phase_4 汇编源码如下：

```
441 0000000000400fce <phase_4>:
442   400fce:  48 83 ec 18        sub    $0x18,%rsp
443   400fd2:  48 8d 4c 24 0c        lea    0xc(%rsp),%rcx
444   400fd7:  48 8d 54 24 08        lea    0x8(%rsp),%rdx
445   400fdc:  be 8d 27 40 00        mov    $0x40278d,%esi
446   400fe1:  b8 00 00 00 00        mov    $0x0,%eax
447   400fe6:  e8 b5 fb ff ff     callq  400ba0 <__isoc99_sscanf@plt>
448   400feb:  83 f8 02           cmp    $0x2,%eax
449   400fee:  75 0c              jne    400ffc <phase_4+0x2e>
450   400ff0:  8b 44 24 0c        mov    0xc(%rsp),%eax
451   400ff4:  83 e8 02           sub    $0x2,%eax
452   400ff7:  83 f8 02           cmp    $0x2,%eax
453   400ffa:  76 05              jbe    401001 <phase_4+0x33>
454   400ffc:  e8 e3 04 00 00        callq  4014e4 <explode_bomb>
455   401001:  8b 74 24 0c        mov    0xc(%rsp),%esi
456   401005:  bf 08 00 00 00        mov    $0x8,%edi
457   40100a:  e8 87 ff ff ff     callq  400f96 <func4>
458   40100f:  3b 44 24 08        cmp    0x8(%rsp),%eax
459   401013:  74 05              je     40101a <phase_4+0x4c>
460   401015:  e8 ca 04 00 00        callq  4014e4 <explode_bomb>
461   40101a:  48 83 c4 18        add    $0x18,%rsp
462   40101e:  c3                 retq
```

```
413 0000000000400f96 <func4>:
414   400f96:  41 54              push   %r12
415   400f98:  55                 push   %rbp
416   400f99:  53                 push   %rbx
417   400f9a:  89 fb              mov    %edi,%ebx
418   400f9c:  85 ff              test   %edi,%edi
419   400f9e:  7e 24                 jle    400fc4 <func4+0x2e>
420   400fa0:  89 f5              mov    %esi,%ebp
421   400fa2:  89 f0              mov    %esi,%eax
422   400fa4:  83 ff 01              cmp    $0x1,%edi
423   400fa7:  74 20                 je     400fc9 <func4+0x33>
424   400fa9:  8d 7f ff              lea    -0x1(%rdi),%edi
425   400fac:  e8 e5 ff ff ff        callq  400f96 <func4>
426   400fb1:  44 8d 24 28           lea    (%rax,%rbp,1),%r12d
427   400fb5:  8d 7b fe              lea    -0x2(%rbx),%edi
428   400fb8:  89 ee                 mov    %ebp,%esi
429   400fba:  e8 d7 ff ff ff        callq  400f96 <func4>
430   400fbf:  44 01 e0              add    %r12d,%eax
431   400fc2:  eb 05                 jmp    400fc9 <func4+0x33>
432   400fc4:  b8 00 00 00 00           mov    $0x0,%eax
433   400fc9:  5b                 pop    %rbx
434   400fca:  5d                 pop    %rbp
435   400fcb:  41 5c                 pop    %r12
436   400fcd:  c3                 retq
```

（2）汇编语言翻译：

Phase_4：

内存地址	汇编代码	含义
400fce:	sub $0x18,%rsp	设置%rsp，设置两个输入的地址。
400fd2:	lea 0xc(%rsp),%rcx	
400fd7:	lea 0x8(%rsp),%rdx	
400fdc:	mov $0x40278d,%esi	
400fe1:	mov $0x0,%eax	
400fe6:	callq 400ba0 <__isoc99_sscanf@plt>	输入两个整数 a、b。返回输入个数。
400feb:	cmp $0x2,%eax	若输入个数不等于 2，则爆炸。
400fee:	jne 400ffc <phase_4+0x2e>	
400ff0:	mov 0xc(%rsp),%eax	若输入的第二个数大于 4，则爆炸。
400ff4:	sub $0x2,%eax	
400ff7:	cmp $0x2,%eax	
400ffa:	jbe 401001 <phase_4+0x33>	
400ffc:	callq 4014e4 <explode_bomb>	
401001:	mov 0xc(%rsp),%esi	调用 func(8，b)，并返回结果。
401005:	mov $0x8,%edi	
40100a:	callq 400f96 <func4>	
40100f:	cmp 0x8(%rsp),%eax	比较输入的第一个数和 func(8，b)，若相等则成功，否则爆炸。
401013:	je 40101a <phase_4+0x4c>	
401015:	callq 4014e4 <explode_bomb>	
40101a:	add $0x18,%rsp	恢复%rsp
40101e:	retq	返回。

func4：

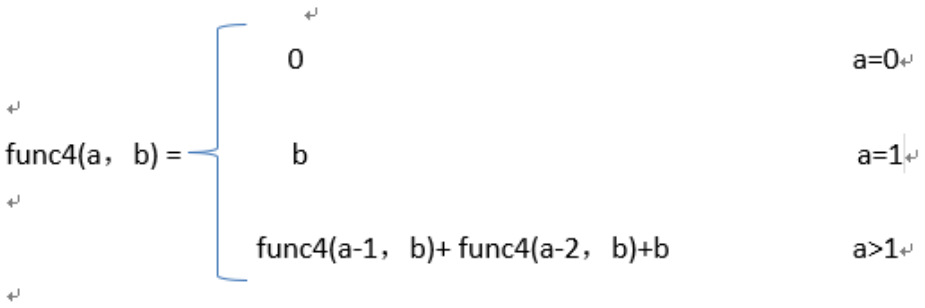
内存地址	汇编代码	含义
400f96:	push %r12	压栈%r12。
400f98:	push %rbp	压栈%rbp。
400f99:	push %rbx	压栈%rbx。

400f9a:	mov %edi,%ebx	%ebx = %edi。
400f9c:	test %edi,%edi	若%edi=0，返回 0。
400f9e:	jle 400fc4 <func4+0x2e>	
400fa0:	mov %esi,%ebp	若%edi=1，返回%rbp。
400fa2:	mov %esi,%eax	
400fa4:	cmp $0x1,%edi	
400fa7:	je 400fc9 <func4+0x33>	
400fa9:	lea -0x1(%rdi),%edi	递归调用 func(a-1，b)。
400fac:	callq 400f96 <func4>	
400fb1:	lea (%rax,%rbp,1),%r12d	递归调用 func(a-2，b)。
400fb5:	lea -0x2(%rbx),%edi	
400fb8:	mov %ebp,%esi	
400fba:	callq 400f96 <func4>	
400fbf:	add %r12d,%eax	返回 func4(a-1，b)+ func4(a-2，b)+b。
400fc2:	jmp 400fc9 <func4+0x33>	跳转至 400fc9
400fc4:	mov $0x0,%eax	返回 0。
400fc9:	pop %rbx	弹栈%rbx。
400fca:	pop %rbp	弹栈%rbp。
400fcb:	pop %r12	弹栈%r12。
400fcd:	retq	返回。

（3）源码分析

可以看出 sscanf 的具体作用，用 x/s $rsi 查看格式串，得出我们需要输入 2 个整数 a、b，通过 sscanf 分别读到%rsp + 0x8，%rsp + 0xc 中。当 a=func4(8，b)时，破解成功，否则爆炸。

其中 func4(a，b)为递归程序，递归方式为：



递归计算结果如下所示：

Fun4(0,n)	Fun4(1,n)	Fun4(2,n)	Fun4(3,n)	Fun4(4,n)	Fun4(5,n)	Fun4(6,n)	Fun4(7,n)	Fun4(8,n)
0	n	2n	4n	7n	12n	20n	33n	54n

a=func4(8，b)，故 a=54b。

答案：162 3

5. Phase_5

（1）源代码

Phase_5 汇编源码如下：

```
000000000040101f <phase_5>:
  40101f:  53                   push   %rbx
  401020:  48 89 fb             mov    %rdi,%rbx
  401023:  e8 39 02 00 00       callq  401261 <string_length>
  401028:  83 f8 06             cmp    $0x6,%eax
  40102b:  74 05                je     401032 <phase_5+0x13>
  40102d:  e8 b2 04 00 00       callq  4014e4 <explode_bomb>
  401032:  b8 00 00 00 00       mov    $0x0,%eax
  401037:  ba 00 00 00 00       mov    $0x0,%edx
  40103c:  0f b6 0c 03          movzbl (%rbx,%rax,1),%ecx
  401040:  83 e1 0f             and    $0xf,%ecx
  401043:  03 14 8d 00 25 40 00 add    0x402500(,%rcx,4),%edx
  40104a:  48 83 c0 01          add    $0x1,%rax
  40104e:  48 83 f8 06          cmp    $0x6,%rax
  401052:  75 e8                jne    40103c <phase_5+0x1d>
  401054:  83 fa 38             cmp    $0x38,%edx
  401057:  74 05                je     40105e <phase_5+0x3f>
  401059:  e8 86 04 00 00       callq  4014e4 <explode_bomb>
  40105e:  5b                   pop    %rbx
  40105f:  90                   nop
  401060:  c3                   retq
```

（2）汇编语言翻译：

内存地址	汇编代码	含义
40101f:	push %rbx	压栈%rbx。
401020:	mov %rdi,%rbx	%rbx=%rdi。
401023:	callq 401261 <string_length>	计算字符串长度。
401028:	cmp $0x6,%eax	比较字符串长度和的大小，若不等于 6，则爆炸。
40102b:	je 401032 <phase_5+0x13>	
40102d:	callq 4014e4 <explode_bomb>	
401032:	mov $0x0,%eax	%eax=0。
401037:	mov $0x0,%edx	%edx=0。

40103c:	movzbl (%rbx,%rax,1),%ecx	字符的低 4 位 ascii 码值对应一个数组的 index，然后数组中对应数字累加进%edx。
401040:	and $0xf,%ecx	
401043:	add 0x402500(,%rcx,4),%edx	
40104a:	add $0x1,%rax	%rax 计数，%rax++。
40104e:	cmp $0x6,%rax	若%rax 小于 6，跳转回 40103c，循环。
401052:	jne 40103c <phase_5+0x1d>	
401054:	cmp $0x38,%edx	比较累加结果与 0x38 的大小，若等于则通过，否则爆炸。
401057:	je 40105e <phase_5+0x3f>	
401059:	callq 4014e4 <explode_bomb>	
40105e:	pop %rbx	弹出%rbx。
40105f:	nop	空。
401060:	retq	返回。

（3）源码分析

对于 phase_5,概括性理解，就是输入 6 个 ascii 字符，然后这 6 个字符的低 4 位对应一个数组的 index，然后对应数字累加要等于 0x38,即 56。

通过 GDB 查看该数组的内容，如下所示：

```
(gdb) print *(int*)0x401bc0
$31 = 2
(gdb) print *(int*)(0x401bc0+4)
$32 = 10
(gdb) print *(int*)(0x401bc0+8)
$33 = 6
(gdb) print *(int*)(0x401bc0+12)
$34 = 1
(gdb) print *(int*)(0x401bc0+16)
$35 = 12
(gdb) print *(int*)(0x401bc0+2-)
A syntax error in expression, near `)'.
(gdb) print *(int*)(0x401bc0+20)
$36 = 16
(gdb) print *(int*)(0x401bc0+24)
$37 = 9
(gdb) print *(int*)(0x401bc0+28)
$38 = 3
(gdb) print *(int*)(0x401bc0+32)
$39 = 4
```

56 可拆分为 2+2+10+10+16+16。

ASCII表

(American Standard Code for Information Interchange 美国标准信息交换代码)

高四位 / 低四位		ASCII控制字符												ASCII打印字符																							
		0000							0001							0010				0011				0100				0101				0110				0111			
		0							1							2				3				4				5				6				7			
		十进制	字符	Ctrl	代码	转义字符	字符解释		十进制	字符	Ctrl	代码	转义字符	字符解释		十进制	字符		十进制	字符		十进制	字符		十进制	字符		十进制	字符		十进制	字符		十进制	字符		Ctrl
0000	0	0		^@	NUL	\0	空字符		16	▶	^P	DLE		数据链路转义		32			48	0		64	@		80	P		96	`		112	p				
0001	1	1	☺	^A	SOH		标题开始		17	◀	^Q	DC1		设备控制 1		33	!		49	1		65	A		81	Q		97	a		113	q				
0010	2	2	☻	^B	STX		正文开始		18	↕	^R	DC2		设备控制 2		34	"		50	2		66	B		82	R		98	b		114	r				
0011	3	3	♥	^C	ETX		正文结束		19	‼	^S	DC3		设备控制 3		35	#		51	3		67	C		83	S		99	c		115	s				
0100	4	4	♦	^D	EOT		传输结束		20	¶	^T	DC4		设备控制 4		36	$		52	4		68	D		84	T		100	d		116	t				
0101	5	5	♣	^E	ENQ		查询		21	§	^U	NAK		否定应答		37	%		53	5		69	E		85	U		101	e		117	u				
0110	6	6	♠	^F	ACK		肯定应答		22	▬	^V	SYN		同步空闲		38	&		54	6		70	F		86	V		102	f		118	v				
0111	7	7	•	^G	BEL	\a	响铃		23	↨	^W	ETB		传输块结束		39	'		55	7		71	G		87	W		103	g		119	w				
1000	8	8	◘	^H	BS	\b	退格		24	↑	^X	CAN		取消		40	(		56	8		72	H		88	X		104	h		120	x				
1001	9	9	○	^I	HT	\t	横向制表		25	↓	^Y	EM		介质结束		41	)		57	9		73	I		89	Y		105	i		121	y				
1010	A	10	◙	^J	LF	\n	换行		26	→	^Z	SUB		替代		42	*		58	:		74	J		90	Z		106	j		122	z				
1011	B	11	♂	^K	VT	\v	纵向制表		27	←	^[	ESC	\e	溢出		43	+		59	;		75	K		91	[		107	k		123	{				
1100	C	12	♀	^L	FF	\f	换页		28	∟	^\	FS		文件分隔符		44	,		60	<		76	L		92	\		108	l		124	|			
1101	D	13	♪	^M	CR	\r	回车		29	↔	^]	GS		组分隔符		45	-		61	=		77	M		93	]		109	m		125	}				
1110	E	14	♫	^N	SO		移出		30	▲	^^	RS		记录分隔符		46	.		62	>		78	N		94	^		110	n		126	~				
1111	F	15	☼	^O	SI		移入		31	▼	^-	US		单元分隔符		47	/		63	?		79	O		95	_		111	o		127	⌂		^Backspace 代码：DEL	

注：表中的ASCII字符可以用“Alt + 小键盘上的数字键”方法输入。 2013/08/08

按上表中的低四位，查询出“2+2+10+10+16+16”对应的字符，应是001155。

答案：001155

6. Phase_6

（1）源代码

Phase_6 汇编源码如下：

```
0000000000401061 <phase_6>:
401061: 41 55              push   %r13
401063: 41 54              push   %r12
401065: 55                 push   %rbp
401066: 53                 push   %rbx
401067: 48 83 ec 58           sub    $0x58,%rsp
40106b: 48 8d 74 24 30        lea    0x30(%rsp),%rsi
401070: e8 a5 04 00 00        callq  40151a <read_six_numbers>
401075: 4c 8d 6c 24 30        lea    0x30(%rsp),%r13
40107a: 41 bc 00 00 00 00        mov    $0x0,%r12d
401080: 4c 89 ed           mov    %r13,%rbp
401083: 41 8b 45 00           mov    0x0(%r13),%eax
401087: 83 e8 01           sub    $0x1,%eax
40108a: 83 f8 05           cmp    $0x5,%eax
40108d: 76 05              jbe    401094 <phase_6+0x33>
40108f:  e8 50 04 00 00       callq  4014e4 <explode_bomb>
401094: 41 83 c4 01           add    $0x1,%r12d
401098: 41 83 fc 06        cmp    $0x6,%r12d
40109c: 75 07              jne    4010a5 <phase_6+0x44>
40109e: be 00 00 00 00        mov    $0x0,%esi
4010a3: eb 42              jmp    4010e7 <phase_6+0x86>
4010a5: 44 89 e3           mov    %r12d,%ebx
4010a8: 48 63 c3           movslq %ebx,%rax
4010ab: 8b 44 84 30           mov    0x30(%rsp,%rax,4),%eax
4010af:  39 45 00          cmp    %eax,0x0(%rbp)
4010b2: 75 05              jne    4010b9 <phase_6+0x58>
4010b4: e8 2b 04 00 00        callq  4014e4 <explode_bomb>
4010b9: 83 c3 01           add    $0x1,%ebx
4010bc: 83 fb 05           cmp    $0x5,%ebx
```

```
512  4010bf: 7e e7           jle    4010a8 <phase_6+0x47>
513  4010c1: 49 83 c5 04        add    $0x4,%r13
514  4010c5: eb b9           jmp    401080 <phase_6+0x1f>
515  4010c7: 48 8b 52 08        mov    0x8(%rdx),%rdx
516  4010cb: 83 c0 01        add    $0x1,%eax
517  4010ce: 39 c8           cmp    %ecx,%eax
518  4010d0: 75 f5           jne    4010c7 <phase_6+0x66>
519  4010d2: eb 05           jmp    4010d9 <phase_6+0x78>
520  4010d4: ba f0 33 60 00     mov    $0x6033f0,%edx
521  4010d9: 48 89 14 74        mov    %rdx,(%rsp,%rsi,2)
522  4010dd: 48 83 c6 04        add    $0x4,%rsi
523  4010e1: 48 83 fe 18     cmp    $0x18,%rsi
524  4010e5: 74 15           je     4010fc <phase_6+0x9b>
525  4010e7: 8b 4c 34 30        mov    0x30(%rsp,%rsi,1),%ecx
526  4010eb: 83 f9 01        cmp    $0x1,%ecx
527  4010ee: 7e e4           jle    4010d4 <phase_6+0x73>
528  4010f0:  b8 01 00 00 00    mov    $0x1,%eax
529  4010f5:  ba f0 33 60 00    mov    $0x6033f0,%edx
530  4010fa:  eb cb          jmp    4010c7 <phase_6+0x66>
531  4010fc:  48 8b 1c 24       mov    (%rsp),%rbx
532  401100: 48 8d 44 24 08     lea    0x8(%rsp),%rax
533  401105: 48 8d 74 24 30     lea    0x30(%rsp),%rsi
534  40110a: 48 89 d9        mov    %rbx,%rcx
535  40110d: 48 8b 10        mov    (%rax),%rdx
536  401110: 48 89 51 08        mov    %rdx,0x8(%rcx)
537  401114: 48 83 c0 08        add    $0x8,%rax
538  401118: 48 39 f0        cmp    %rsi,%rax
539  40111b: 74 05           je     401122 <phase_6+0xc1>
```

```
40111d: 48 89 d1          mov    %rdx,%rcx
401120: eb eb             jmp    40110d <phase_6+0xac>
401122: 48 c7 42 08 00 00 00  movq   $0x0,0x8(%rdx)
401129: 00
40112a: bd 05 00 00 00       mov    $0x5,%ebp
40112f:  48 8b 43 08         mov    0x8(%rbx),%rax
401133: 8b 00             mov    (%rax),%eax
401135: 39 03             cmp    %eax,(%rbx)
401137: 7e 05             jle    40113e <phase_6+0xdd>
401139: e8 a6 03 00 00       callq  4014e4 <explode_bomb>
40113e: 48 8b 5b 08          mov    0x8(%rbx),%rbx
401142: 83 ed 01          sub    $0x1,%ebp
401145: 75 e8             jne    40112f <phase_6+0xce>
401147: 48 83 c4 58          add    $0x58,%rsp
40114b: 5b              pop    %rbx
40114c: 5d              pop    %rbp
40114d: 41 5c              pop    %r12
40114f:  41 5d             pop    %r13
401151: c3              retq
```

（2）源码分析

对 phase_6（）函数反汇编。可以通过函数看出，我们输入的 6 位数字，且都小于 6。接下去是一个 while 循环，r12d 和 ebx 是一个循环计数器。该循环用于判断这 6 个数是否存在等于 0 的。可以推测得到每个数字都不为 0。接下来查看 0x6033F0 位置处的内容，按照我们输入的数字的标号在指定位置排序。之后将 node 之间建立链表连接指针。最后判断该链表是否递减，如果递增，则不爆，否则爆炸。

Node	数据
1	25d

2	65
3	15D
4	19C
5	155
6	3C7

答案：2 5 3 4 1 6

7. Secret_Phase

（1）源代码

secret_Phase 汇编源码如下：

```
0000000000401190 <secret_phase>:
  401190: 53                    push   %rbx
  401191: e8 c6 03 00 00        callq  40155c <read_line>
  401196: ba 0a 00 00 00        mov    $0xa,%edx
  40119b: be 00 00 00 00        mov    $0x0,%esi
  4011a0: 48 89 c7              mov    %rax,%rdi
  4011a3: e8 d8 f9 ff ff        callq  400b80 <strtol@plt>
  4011a8: 48 89 c3              mov    %rax,%rbx
  4011ab: 8d 40 ff              lea    -0x1(%rax),%eax
  4011ae: 3d e8 03 00 00        cmp    $0x3e8,%eax
  4011b3: 76 05                 jbe    4011ba <secret_phase+0x2a>
  4011b5: e8 2a 03 00 00        callq  4014e4 <explode_bomb>
  4011ba: 89 de                 mov    %ebx,%esi
  4011bc: bf 10 32 60 00        mov    $0x603210,%edi
  4011c1: e8 8c ff ff ff        callq  401152 <fun7>
  4011c6: 83 f8 07              cmp    $0x7,%eax
  4011c9: 74 05                 je     4011d0 <secret_phase+0x40>
  4011cb: e8 14 03 00 00        callq  4014e4 <explode_bomb>
  4011d0: bf 90 24 40 00        mov    $0x402490,%edi
  4011d5: e8 e6 f8 ff ff        callq  400ac0 <puts@plt>
  4011da: e8 a3 04 00 00        callq  401682 <phase_defused>
  4011df: 5b                    pop    %rbx
  4011e0: c3                    retq
  4011e1: 66 2e 0f 1f 84 00 00  nopw   %cs:0x0(%rax,%rax,1)
  4011e8: 00 00 00
  4011eb: 0f 1f 44 00 00        nopl   0x0(%rax,%rax,1)
```

```
0000000000401152 <fun7>:
401152: 48 83 ec 08          sub    $0x8,%rsp
401156: 48 85 ff             test   %rdi,%rdi
401159: 74 2b                je     401186 <fun7+0x34>
40115b: 8b 17                mov    (%rdi),%edx
40115d: 39 f2                cmp    %esi,%edx
40115f: 7e 0d                jle    40116e <fun7+0x1c>
401161: 48 8b 7f 08          mov    0x8(%rdi),%rdi
401165: e8 e8 ff ff ff       callq  401152 <fun7>
40116a: 01 c0                add    %eax,%eax
40116c: eb 1d                jmp    40118b <fun7+0x39>
40116e: b8 00 00 00 00       mov    $0x0,%eax
401173: 39 f2                cmp    %esi,%edx
401175: 74 14                je     40118b <fun7+0x39>
401177: 48 8b 7f 10          mov    0x10(%rdi),%rdi
40117b: e8 d2 ff ff ff       callq  401152 <fun7>
401180: 8d 44 00 01          lea    0x1(%rax,%rax,1),%eax
401184: eb 05                jmp    40118b <fun7+0x39>
401186: b8 ff ff ff ff       mov    $0xffffffff,%eax
40118b: 48 83 c4 08          add    $0x8,%rsp
40118f: c3                   retq
```

```
0000000000401682 <phase_defused>:
401682: 48 83 ec 68          sub    $0x68,%rsp
401686: bf 01 00 00 00       mov    $0x1,%edi
40168b: e8 90 fd ff ff       callq  401420 <send_msg>
401690: 83 3d 05 22 20 00 06 cmpl   $0x6,0x202205(%rip)        # 60389c <num_input_strings>
401697: 75 6d                jne    401706 <phase_defused+0x84>
401699: 4c 8d 44 24 10       lea    0x10(%rsp),%r8
40169e: 48 8d 4c 24 08       lea    0x8(%rsp),%rcx
4016a3: 48 8d 54 24 0c       lea    0xc(%rsp),%rdx
4016a8: be d7 27 40 00       mov    $0x4027d7,%esi
4016ad: bf b0 39 60 00       mov    $0x6039b0,%edi
4016b2: b8 00 00 00 00       mov    $0x0,%eax
4016b7: e8 e4 f4 ff ff       callq  400ba0 <__isoc99_sscanf@plt>
4016bc: 83 f8 03             cmp    $0x3,%eax
4016bf: 75 31                jne    4016f2 <phase_defused+0x70>
4016c1: be e0 27 40 00       mov    $0x4027e0,%esi
4016c6: 48 8d 7c 24 10       lea    0x10(%rsp),%rdi
4016cb: e8 ae fb ff ff       callq  40127e <strings_not_equal>
4016d0: 85 c0                test   %eax,%eax
4016d2: 75 1e                jne    4016f2 <phase_defused+0x70>
4016d4: bf 38 26 40 00       mov    $0x402638,%edi
4016d9: e8 e2 f3 ff ff       callq  400ac0 <puts@plt>
4016de: bf 60 26 40 00       mov    $0x402660,%edi
4016e3: e8 d8 f3 ff ff       callq  400ac0 <puts@plt>
4016e8: b8 00 00 00 00       mov    $0x0,%eax
4016ed: e8 9e fa ff ff       callq  401190 <secret_phase>
4016f2: bf 98 26 40 00       mov    $0x402698,%edi
4016f7: e8 c4 f3 ff ff       callq  400ac0 <puts@plt>
4016fc: bf c8 26 40 00       mov    $0x4026c8,%edi
401701: e8 ba f3 ff ff       callq  400ac0 <puts@plt>
401706: 48 83 c4 68          add    $0x68,%rsp
40170a: c3                   retq
40170b: 0f 1f 44 00 00       nopl   0x0(%rax,%rax,1)
```

phase_defused

可以看到第 7 行将函数 num_input_strings 的返回值与 6 进行比较，如果不等于 6 则的直接跳过中间代码到达最后的结束部分。

从函数名我们可以推测这个函数的作用的是检测读取的字符串的数量，当读取了 6 个字符串时，就不会跳过中间的代码。第 9 到 14 行又是熟悉的 sscanf 调用过程，我们已经知道 esi 指向的是格式化字符串的首地址，查看它的内容：

```
(gdb) x /s 0x402619
0x402619:       "%d %d %s"
```

读取两个整数和一个字符串。有所不同的是在 12 行之后又有一行给 edi 赋上了一个地址值，我们之前所有阶段中 edi 的值都是来自于我们 read_line 的地址，想到 sscanf 参数中确实存在一个输入，我们可以推测这个 edi 中存放的是我们读取位置的首地址。

那么就可以在运行时查看这个地址的内容，看是从哪里进行读取的：

```
(gdb) x /s 0x603870
0x603870 <input_strings+240>:   "0 0"
```

第 15、16 行对成功输入的数据个数进行了一个判断，如果不为 3 个则跳过调用 secret_phase 的代码。

第 17-19 行是对 strings_not_equal 的调用，我们已经知道它的两个参数分别是 esi 与 edi，esi 被赋上了一个地址值，edi 被赋上了 esp+0x10，我们可以推测出

edi 的地址就是指向我们读入的第三个字符串的，那么需要比较的对象是什么呢？我们在运行时查看内存的内容：

```
(gdb) x /s 0x402622
0x402622:       "DrEvil"
```

这就是需要的第三个参数。

可以看到如果第三个参数与上面这个字符串相同的话就会调用两次 puts 输出提示信息，然后进入 secret_phase 阶段。

secret_phase：

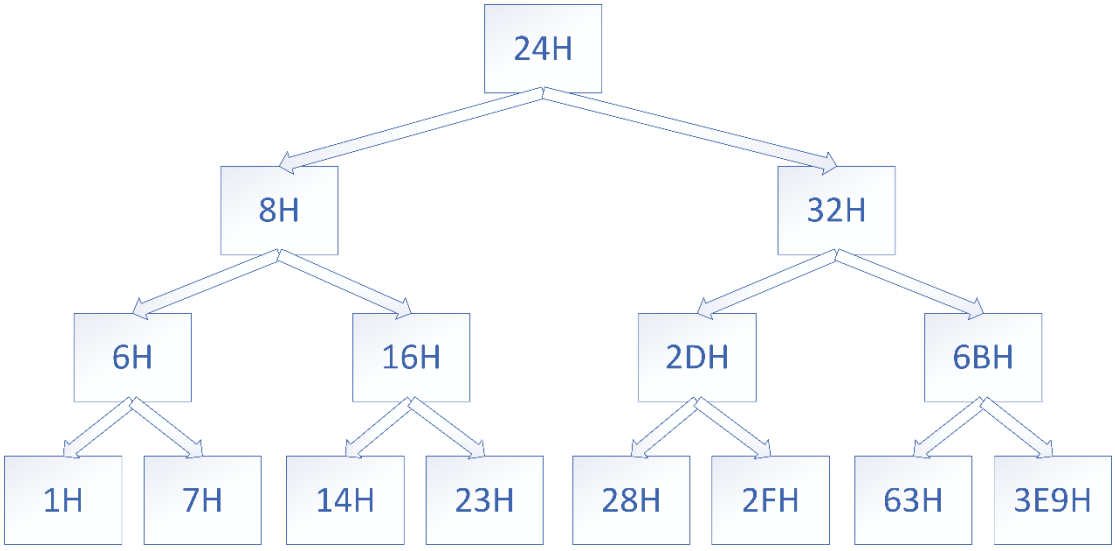
可以看到第 3 行调用了 read_line 函数，接着把 read_line 的返回值赋给了 rdi，并调用了 strtol 函数，这个标准库函数的作用是把一个字符串转换成对应的长整型数值。返回值还是存放在 rax 中，第 8 行将 rax 复制给了 rbx，第 9 行将 rax 减 1 赋给 eax，第十行与 0x3e8 进行比较，如果这个值小于等于 0x3e8 就跳过引爆代码。看到这里我们可以知道我们需要再加入一行数据，它应该是一个小于等于 1001 的数值。

接下来将 ebx 赋给了 esi，也就是我们一开始输入的 rax 值。第 14 行将一个地址值赋给了 edi，15 行调用了 fun7 函数。我们还是先往下了解一下我们需要得到的结果。函数返回后令返回值 eax 与 0x7 做了一个比较，如果相等则跳过引爆代码。

fun7：

第 3、4 两行先对我们输入的这个数作一个判断，如果等于 0 直接跳到第 19 行，返回-1，这显然不是我们想要的结果。第 5 行将 rdi 的值读入到了 edx 中，第 6 行则将这个数与我们读入的数进行比较，如果这个数小于等于我们读入的数就跳至第 12 行，第 12 行将 eax 置 0，再进行一次相同的比较，如果相等则跳至第 20 行返回。

这是一个二叉树的结构，每个节点第 1 个 8 字节存放数据，第 2 个 8 字节存放左子树地址，第 3 个 8 字节存放右子树位置。并且命令也有规律，nab，a 代表层数，b 代表从左至右第 b 个节点。二叉树如下图所示：



fun7()输入的值对二叉树进行操作，并计算返回值。根据递归反写出 C 语言。

```
void fun7(Node* node,int value)
{//node in %rdi,value in %rsi,return_value in %eax
 //require %eax to be 2(Very important)
    int t=node->val;
    if(t>value)
    {
        node=node->left;
        fun7(node,value);
```

```
        return_value=2*return_value;
        return;
    }
    else
    {
        if(value==t)
        {
            return;
        }
        node=node->right;
        fun7(node,value);
        return_value=0x1+2*return_value;
        return;
    }
}
```

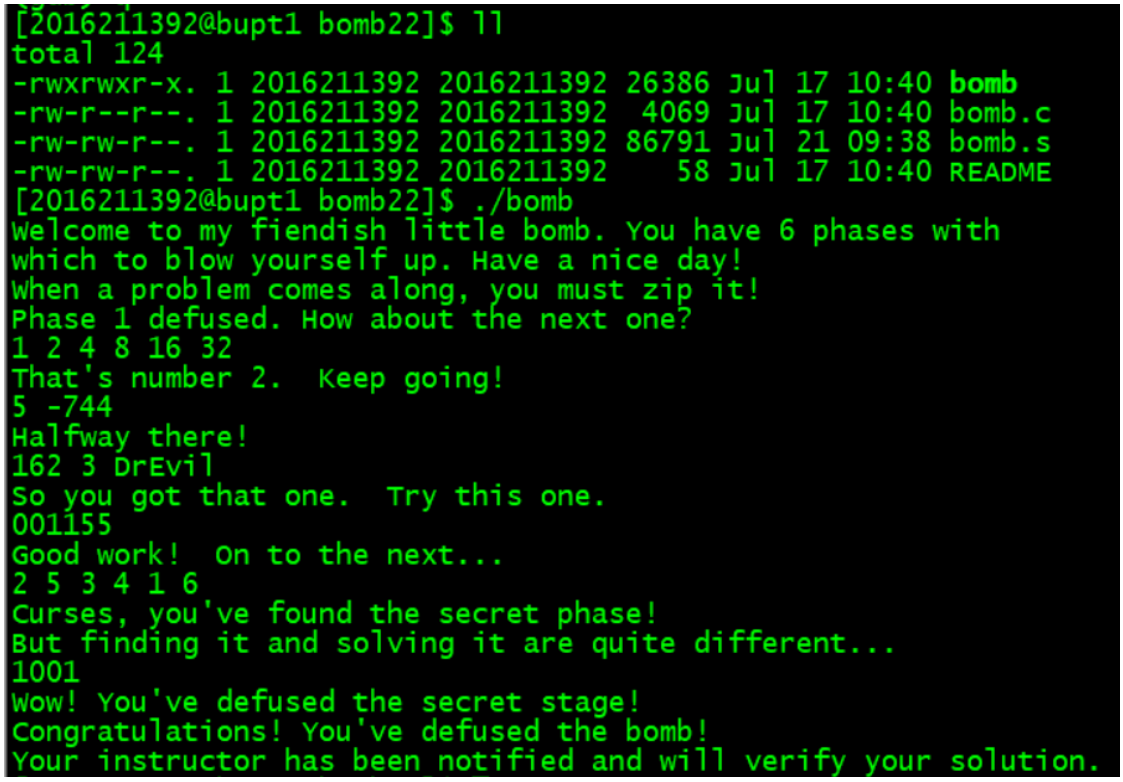
有返回值要等于 0x7，故通过计算得输入 1001.

答案：1001

七、实验结果

全部关卡全部通过，实验过程中共爆炸 6 次，全部出现在附加关卡中。由于没有任何提示信息，我下意识以为输入完“DrEvil”之后就紧接着附加关卡，直接就输入了附加关卡的答案，并且没意识到这个问题，一直以为是计算错误，最后才发现附加关卡是在 6 个关卡结束之后才显示，很遗憾在此处产生这么多次爆炸。

实验全部炸弹拆除成果如下图所示：



八、收获与体验

本次实验过程中主要卡在了隐藏关卡的查找上，错误爆炸都是在尝试进入隐藏关卡时产生。由于没有任何提示信息，我下意识以为输入完“DrEvil”之后就紧接着附加关卡，直接就输入了附加关卡的答案，并且没意识到这个问题，一直以为是计算错误，最后才发现附加关卡是在 6 个关卡结束之后才显示，很遗憾在此处产生这么多次爆炸，希望老师谅解。

整个实验包括秘密部分用时 8 个小时，引爆了 6 次炸弹（全是在尝试进入 Secret_Phase 时产生）。

一开始拿到题目的时候会比较蒙，需要先去学习工具的使用与一些编译的基础知道（符号表、定址表等等）花费了一些时间。前几个阶段过于关注函数的具体实现而没有根据常识去推测一些明显函数的作用花费了一些时间。

前 4 个阶段都算比较简单，考查了一些常用结构在汇编中的出现形式。第 5、6 阶段、秘密关卡分别考察了堆、链表、二叉树等数据结构在内存中的结构与汇编级的使用，受益良多。

整个破解思想基本为先翻译解释源码，找到能避开爆炸的输入，然后符合要求的输入。

从破解过程来看，需要较高的水平的汇编代码阅读能力，还要不怕麻烦的对许多地址进行递归的查询，很多时候对于 “查询哪个地址可以看到需要的东西”的问题只有在大致的范围上不断寻找和计算才能准确找到能完美表现变化的地址。所以还是需要很大的耐心的。由于不需要大量的指令，只要懂得 gdb 的基本使用和查看内存的指令并且耐心调试就可以。

这个实验需要细致的分析与大胆的猜测与实验验证，还需要小心操作，最重要的是耐心，面对非常晦涩的汇编代码如何一步步地弄清代码的作用很需要毅力。当然通过这次实验，我对汇编语言有了更加透彻的了解，掌握了 GDB 调试方法。也更加热爱阅读汇编源码的过程。同时，也感谢老师设置这么有意思的实验，不仅提高了我的汇编能力，还增进了我对汇编语言的兴趣。

九、参考文献

深入理解计算机系统（原书第三版）