

# 天津大学

## 计算机系统基础上机实验报告

实验题目 3：拆弹专家 bomb

学院名称\_\_\_\_\_未来技术学院\_\_\_\_\_

专    业\_\_\_\_\_计算机科学与技术\_\_\_\_\_

学生姓名\_\_\_\_\_石云天\_\_\_\_\_

学    号\_\_\_\_\_3020205015\_\_\_\_\_

年    级\_\_\_\_\_2020 级\_\_\_\_\_

班    级\_\_\_\_\_2 班\_\_\_\_\_

时    间\_\_\_\_\_2022 年 11 月 4 日\_\_\_\_\_

# 实验 3：拆弹专家

## Bomb

### 一、实验目的

进一步掌握程序的机器级表示一章的知识。理解程序控制、过程调用的汇编级实现，熟练掌握 汇编语言程序的阅读。

### 二、实验内容

程序 bomb 是一个电子炸弹，当该程序运行时，需要按照一定的顺序输入口令，才能阻止炸弹的引爆。当输入错误的密码时，炸弹将会引爆。此时控制台将会产生如下输出（见图 1），并结束程序

```
1 BOOM!!!  
2 The bomb has blown up.
```

图 1 错误密码输出

在炸弹程序中，你需要输入多组口令，且每一组口令都正确才能够防止引爆。

目前已知的内容只有炸弹程序的二进制可执行文件 **bomb**（目标平台为：x86-64）和 **bomb** 的 **main** 函数框架代码，见 **main.c**。其他的细节均不会以 **c** 语言的方式呈现。你的任务是：利用现有的资源以及相关的工具，猜出炸弹的全部口令，并输入至炸弹程序中，以完成最终的拆弹工作。

### 三、实验要求

（1）在 Unbuntu18.04LTS 操作系统下，按照实验指导说明书，使用 **gdb** 和 **objdump** 等工具，以反向工程方式完成 **Bomb** 拆弹。

（2）需提交：拆弹口令文本文件、电子版实验报告全文。

## 四、实验结果

（拆弹 phase1 到 phase6 的拆弹分析过程，可以包括典型的汇编码片段、拆弹思路、拆弹过程中的结果截图。可选项：隐藏关卡的破解过程分析）

首先利用 objdump 反汇编工具获得 bomb 的全部反汇编代码，并将其定向写入至文件 disbomb.txt 中，所用代码如下：

```
linux> objdump -d bomb >disbomb.txt
```

### 4.1 phase1

#### (1) 任务描述：

通过 phase1 的反汇编代码寻找对应要求输入的字符串；

#### (2) 任务求解：

在 disbomb.txt 文件中查找 phase1 对应的反汇编代码，所得结果见下图 2：

```
0000000000400ee0 <phase_1>:
400ee0: 48 83 ec 08      sub    $0x8,%rsp
400ee4: be 00 24 40 00   mov    $0x402400,%esi
400ee9: e8 4a 04 00 00   callq 401338 <strings_not_equal>
400eee: 85 c0           test   %eax,%eax
400ef0: 74 05           je     400ef7 <phase_1+0x17>
400ef2: e8 43 05 00 00   callq 40143a <explode_bomb>
400ef7: 48 83 c4 08      add    $0x8,%rsp
400efb: c3             retq
```

图 2 phase1 对应反汇编代码

在该段反汇编代码中，400ee9 行中出现<strings\_not\_equal>，据此可以推断出第一个问题是判断两个字符串是否相等，其中一个由我们输入，另一个是系统预先设定好的密码。通过分析反汇编代码可知，我们输入的字符串被存储在%rsp 中，而密码被存储在%esi 的地址 0x402400 中，因此直接查找 0x402400 处的内容即可获得密码，利用 gdb 调试工具，输入指令：

```
(gdb) x /s 0x402400
```

得到结果如下图 3 所示：

```
(gdb) x /s 0x402400
0x402400: "Border relations with Canada have never been better."
```

图 3 phase1 字符串

随后将所得的字符串”Border relations with Canada have never been better”输入到可执行程序 bomb 中，显示 phase1 拆弹成功。

#### (3) 任务结果：

结果见下图 4：

```

simpleedu@simpleedu:~/lab3$ ./bomb password.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?

```

图 4 phase1 拆弹结果

故 phase1 密码为: **Border relations with Canada have never been better.**

## 4.2 phase2

### (1) 任务描述:

通过 phase2 的反汇编代码寻找对应要求输入的数据;

### (2) 任务求解:

在 disbomb.txt 文件中查找 phase2 对应的反汇编代码, 所得结果见下图 5:

```

0000000000400efc <phase_2>:
400efc: 55                push    %rbp
400efd: 53                push    %rbx
400efe: 48 83 ec 28       sub     $0x28,%rsp
400f02: 48 89 e6          mov     %rsp,%rsi
400f05: e8 52 05 00 00    callq  40145c <read_six_numbers>
400f0a: 83 3c 24 01       cmpl    $0x1,(%rsp)
400f0e: 74 20             je      400f30 <phase_2+0x34>
400f10: e8 25 05 00 00    callq  40143a <explode_bomb>
400f15: eb 19             jmp     400f30 <phase_2+0x34>
400f17: 8b 43 fc          mov     -0x4(%rbx),%eax
400f1a: 01 c0             add     %eax,%eax
400f1c: 39 03             cmp     %eax,(%rbx)
400f1e: 74 05             je      400f25 <phase_2+0x29>
400f20: e8 15 05 00 00    callq  40143a <explode_bomb>
400f25: 48 83 c3 04       add     $0x4,%rbx
400f29: 48 39 eb          cmp     %rbp,%rbx
400f2c: 75 e9             jne     400f17 <phase_2+0x1b>
400f2e: eb 0c             jmp     400f3c <phase_2+0x40>
400f30: 48 8d 5c 24 04    lea     0x4(%rsp),%rbx
400f35: 48 8d 6c 24 18    lea     0x18(%rsp),%rbp
400f3a: eb db             jmp     400f17 <phase_2+0x1b>
400f3c: 48 83 c4 28       add     $0x28,%rsp
400f40: 5b                pop     %rbx
400f41: 5d                pop     %rbp
400f42: c3                retq

```

图 5 phase2 对应反汇编代码

在该段反汇编代码中, 400f05 行中出现<read\_six\_numbers>, 据此可以推断出第二个问题要求输入六个数字, 并与预设的密码进行比较。从 phase2 的反汇编代码可以看出, 这是一个循环程序, 具体执行过程如下:

1. 输入读取六个数字存储在寄存器%rsp 中; (400f05)
2. 将%rsp 中第一个数字与 1 进行比较, 若相等跳转到 400f30 行, 若不等则炸弹爆炸; (400f0a、400f0e、400f10) (由此可知密码的第一个数字为 1)
3. 将%rsp 中地址加 4 (取下一个数字) 存储到%rbx 中, 将%rsp 中地址加 24 存储到%rbp 中, 随后跳转到 400f17 行; (400f30、400f35、400f3a)
4. 将%rbx 中地址减 4 (取上一个数字) 存储到%eax 中, 并比较 2\*%eax

和%rbx 的值，若相等跳转到 400f25 行，若不等则炸弹爆炸；(400f17、400f1a、400f1c、400f1e、400f20) (由此可知密码的第二个数字为  $1*2=2$ )

5. 将%rbx 的地址加 4 (再取下一个数字)，并与%rbp 的地址进行比较，若不等，则跳转到 400f17 行，进行下一次循环，若相等，跳转到 400f3c 行，结束循环；(400f25、400f29、400f2c、400f2e)(结合第 3 步可知，循环共进行  $24/4=6$  次)

通过以上循环过程可以看出每次循环都是将设定的前一密码值翻倍并与输入的数字进行比较，共循环 6 次，故预测密码是首项为 1，公比为 2，长度为 6 的等比数列。随后将预测的密码 1 2 4 8 16 32 输入到可执行程序 bomb 中，显示 phase2 拆弹成功。

### (3) 任务结果:

结果见下图 6:

```

simpleedu@simpleedu:~/lab3$ ./bomb password.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!
  
```

图 6 phase2 拆弹结果

故 phase2 密码为: 1 2 4 8 16 32

## 4.3 phase3

### (1) 任务描述:

通过 phase3 的反汇编代码寻找对应要求输入的数据;

### (2) 任务求解:

在 disbomb.txt 文件中查找 phase3 对应的反汇编代码，所得结果见下图 7:

0000000000400f43 <phase_3>:		
400f43:	48 83 ec 18	sub \$0x18,%rsp
400f47:	48 8d 4c 24 0c	lea 0xc(%rsp),%rcx
400f4c:	48 8d 54 24 08	lea 0x8(%rsp),%rdx
400f51:	be cf 25 40 00	mov \$0x4025cf,%esi
400f56:	b8 00 00 00 00	mov \$0x0,%eax
400f5b:	e8 90 fc ff ff	callq 400bf0 <__isoc99_sscanf@plt>
400f60:	83 f8 01	cmp \$0x1,%eax
400f63:	7f 05	jg 400f6a <phase_3+0x27>
400f65:	e8 d0 04 00 00	callq 40143a <explode_bomb>
400f6a:	83 7c 24 08 07	cmpl \$0x7,0x8(%rsp)
400f6f:	77 3c	ja 400fad <phase_3+0x6a>
400f71:	8b 44 24 08	mov 0x8(%rsp),%eax
400f75:	ff 24 c5 70 24 40 00	jmpq *0x402470(,%rax,8)
400f7c:	b8 cf 00 00 00	mov \$0xcf,%eax
400f81:	eb 3b	jmp 400fbe <phase_3+0x7b>
400f83:	b8 c3 02 00 00	mov \$0x2c3,%eax
400f88:	eb 34	jmp 400fbe <phase_3+0x7b>
400f8a:	b8 00 01 00 00	mov \$0x100,%eax
400f8f:	eb 2d	jmp 400fbe <phase_3+0x7b>
400f91:	b8 85 01 00 00	mov \$0x185,%eax
400f96:	eb 26	jmp 400fbe <phase_3+0x7b>
400f98:	b8 ce 00 00 00	mov \$0xce,%eax
400f9d:	eb 1f	jmp 400fbe <phase_3+0x7b>
400f9f:	b8 aa 02 00 00	mov \$0x2aa,%eax
400fa4:	eb 18	jmp 400fbe <phase_3+0x7b>
400fa6:	b8 47 01 00 00	mov \$0x147,%eax
400fab:	eb 11	jmp 400fbe <phase_3+0x7b>
400fad:	e8 88 04 00 00	callq 40143a <explode_bomb>
400fb2:	b8 00 00 00 00	mov \$0x0,%eax
400fb7:	eb 05	jmp 400fbe <phase_3+0x7b>
400fb9:	b8 37 01 00 00	mov \$0x137,%eax
400fbe:	3b 44 24 0c	cmp 0xc(%rsp),%eax
400fc2:	74 05	je 400fc9 <phase_3+0x86>
400fc4:	e8 71 04 00 00	callq 40143a <explode_bomb>
400fc9:	48 83 c4 18	add \$0x18,%rsp
400fcd:	c3	retq

图 7 phase3 对应反汇编代码

从 phase3 的反汇编代码中我们可以发现，整个程序的主体一个 switch 跳转语句，具体执行过程如下：

1. 给寄存器 %rsp 分配 24 个字节，通过 %rsp 给 %rcx 分配 12 个字节，通过 %rsp 给 %rdx 分配 8 个字节；(400f43、400f47、400f4c)

2. 将输入个数存储在 %eax 中，并判断输入个数是否大于 1，若输入个数大于 1，则跳转到 400f6a 行，若不大于 1 则炸弹爆炸；(400f56、400f5b、400f60、400f63、400f65)

3. 判断 %rax 中第一个数字和 7 的大小，若大于 7，则炸弹爆炸，若小于等于 7，则将这个数字移动到 %eax 中，随后跳转到地址 (\*0x402470(,%rax,8)) (需计算对应结果)；(400f6a、400f6f、400f71、400f75、400fad)

4. 利用 gdb 工具查看 0x402470 地址下对应内容，得到结果见下图 8：

```
(gdb) x/100wx 0x402470
0x402470: 0x00400f7c 0x00000000 0x00400fb9 0x00000000
0x402480: 0x00400f83 0x00000000 0x00400f8a 0x00000000
0x402490: 0x00400f91 0x00000000 0x00400f98 0x00000000
0x4024a0: 0x00400f9f 0x00000000 0x00400fa6 0x00000000
```

图 8 0x402470 地址对应内容

由于 %rax 中的值在 0-7 之间，因此根据 (\*0x402470(,%rax,8)) 计算出的地址进行跳转选择，并将对应值存入 %eax 中：

当 %rax 中值为 0 时，对应地址为 0x00400f7c，对应 0xcf=207；

当 %rax 中值为 1 时，对应地址为 0x00400fb9，对应 0x137=311；

当 %rax 中值为 2 时，对应地址为 0x00400f83，对应 0x2c3=707；

当 %rax 中值为 3 时，对应地址为 0x00400f8a，对应 0x100=256；

当 %rax 中值为 4 时，对应地址为 0x00400f91，对应 0x185=289；

当 %rax 中值为 5 时，对应地址为 0x00400f98，对应 0xce=206；

当 %rax 中值为 6 时，对应地址为 0x00400f9f，对应 0x2aa=682；

当 %rax 中值为 7 时，对应地址为 0x00400fa6，对应 0x147=327；

(400f7c-400fa6)

因此预测密码为 (0 207) / (1 311) / (2 707) / (3 256) / (4 389) / (5 206) / (6 682) / (7 327)，在八组中任取其一即可。以 (1 311) 为例将其输入到可执行程序 bomb 中，显示 phase3 拆弹成功。

### (3) 任务结果：

结果见下图 9：

```
simpleedu@simpleedu:~/lab3$ ./bomb password.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!
```

图 9 phase3 拆弹结果



故 phase3 密码为: (0 207) / (1 311) / (2 707) / (3 256) / (4 389) / (5 206)  
/ (6 682) / (7 327)

## 4.4 phase4

### (1) 任务描述:

通过 phase4 的反汇编代码寻找对应要求输入的数据;

### (2) 任务求解:

在 disbomb.txt 文件中查找 phase4 对应的反汇编代码, 所得结果见下图 10:

```
000000000040100c <phase_4>:
40100c: 48 83 ec 18      sub    $0x18,%rsp
401010: 48 8d 4c 24 0c    lea    0xc(%rsp),%rcx
401015: 48 8d 54 24 08    lea    0x8(%rsp),%rdx
40101a: be cf 25 40 00    mov    $0x4025cf,%esi
40101f: b8 00 00 00 00    mov    $0x0,%eax
401024: e8 c7 fb ff ff    callq 400bf0 <__isoc99_sscanf@plt>
401029: 83 f8 02         cmp    $0x2,%eax
40102c: 75 07           jne    401035 <phase_4+0x29>
40102e: 83 7c 24 08 0e    cmpl   $0xe,0x8(%rsp)
401033: 76 05           jbe    40103a <phase_4+0x2e>
401035: e8 00 04 00 00    callq 40143a <explode_bomb>
40103a: ba 0e 00 00 00    mov    $0xe,%edx
40103f: be 00 00 00 00    mov    $0x0,%esi
401044: 8b 7c 24 08      mov    0x8(%rsp),%edi
401048: e8 81 ff ff ff    callq 400fce <func4>
40104d: 85 c0           test   %eax,%eax
40104f: 75 07           jne    401058 <phase_4+0x4c>
401051: 83 7c 24 0c 00    cmpl   $0x0,0xc(%rsp)
401056: 74 05           je     40105d <phase_4+0x51>
401058: e8 dd 03 00 00    callq 40143a <explode_bomb>
40105d: 48 83 c4 18      add    $0x18,%rsp
401061: c3             retq
```

图 10 phase4 对应反汇编代码

从 phase4 的反汇编代码可以看出, 这是一个循环程序, 具体执行过程如下:

1. 给寄存器%rsp 分配 24 个字节, 通过%rsp 给%rcx 分配 12 个字节, 通过%rsp 给%rdx 分配 8 个字节; (40100c、401010、401015)

2. 将输入个数存储在%eax 中, 并判断输入个数是否等于 2, 若输入个数等于 2, 则判断第一个数字是否小于等于 14, 若不等于 2 则炸弹爆炸; (40101f、401024、401029、40102c、40102e、401035) (由此知应输入两个数字且第一个数字应小于 14)

3. 若第一个数字小于等于 14, 则跳转到第 40103a 行, 若大于 14, 则炸弹爆炸。随后将%edx 中的值设为 14, %esi 中的值设为 0, 将%edi 中的值设为输入的第二个数字, 将其带入之前定义的 func4 函数进行计算; (401033、40103a、40103f、401044、401048)

4. 判断 func4 函数的返回值是否为 0, 若不为 0, 则炸弹爆炸, 若为 0, 则判断输入的第二个数字是否为 0, 若不为 0, 则炸弹爆炸, 若为 0, 则程序结束 (拆弹成功); (40104d-40105d) (由此知输入的第二个数字经 func4 函数运算后

返回值应为 0 且输入的第二个数字应为 0)

接下来需要研究 func4 函数的反汇编代码,寻找在 0-14 中可以使其返回值等于 0 的数,即可预测出最终的密码。由于 func4 函数的反汇编代码直接研究起来较为困难,故考虑将其转化为对应的 C++ 程序,func4 函数反汇编代码见图 11:

```
0000000000400fce <func4>:
400fce: 48 83 ec 08      sub    $0x8,%rsp
400fd2: 89 d0            mov    %edx,%eax
400fd4: 29 f0            sub    %esi,%eax
400fd6: 89 c1            mov    %eax,%ecx
400fd8: c1 e9 1f         shr    $0x1f,%ecx
400fdb: 01 c8            add    %ecx,%eax
400fdd: d1 f8            sar    %eax
400fdf: 8d 0c 30         lea    (%rax,%rsi,1),%ecx
400fe2: 39 f9            cmp    %edi,%ecx
400fe4: 7e 0c            jle    400ff2 <func4+0x24>
400fe6: 8d 51 ff         lea    -0x1(%rcx),%edx
400fe9: e8 e0 ff ff ff   callq  400fce <func4>
400fee: 01 c0            add    %eax,%eax
400ff0: eb 15            jmp    401007 <func4+0x39>
400ff2: b8 00 00 00 00   mov    $0x0,%eax
400ff7: 39 f9            cmp    %edi,%ecx
400ff9: 7d 0c            jge    401007 <func4+0x39>
400ffb: 8d 71 01         lea    0x1(%rcx),%esi
400ffe: e8 cb ff ff ff   callq  400fce <func4>
401003: 8d 44 00 01         lea    0x1(%rax,%rax,1),%eax
401007: 48 83 c4 08      add    $0x8,%rsp
40100b: c3              retq
```

图 11 func4 函数对应反汇编代码

func4 函数对应 C++ 程序及两者对应关系如下:

```
int func4(int a,int b, int c){
//a in %edi,b in %esi,c in %edx,d in %eax,e in %ecx.
    int d,e;
    d = c - b;//400fd2,400fd4.
    e = d;//400fd6
    e>>= 31;//400fd8
    d = d + e;//400fdb
    d >>= 1;//400fdd
    e = d + b;//400fdf
    if(e > a){//400fe2,400fe4
        c = e - 1;//400fe6
        d = 2 * func4(a,b,c);//400fe9,400fee
    }else{
        d = 0;//400ff2
        if(e < a){//400ff7,400ff9
            b = e + 1;//400ffb
        }
    }
}
```



```

        d = 2 * func4(a,b,c) + 1; //400ffe,401003
    }
}
return d;
}

```

结合 phase4 的反汇编代码，写出其对应的 main 函数如下：

```

int main(){
    for(int i=0; i<=14; i++){
        if(func4(i,0,14) == 0)
            cout << i << " " << func4(i,0,14) << endl;
    }
    return 0;
}

```

利用 CodeBlocks 编译器运行以上程序，所得结果见下图 12：

```

C:\Users\LENOVO\Desktop\phase4\bin\Debug\phase4.exe
0 0
1 0
3 0
7 0

Process returned 0 (0x0)   execution time : 0.030 s
Press any key to continue.

```

图 12 phase4 对应 C++ 程序运行结果

因此预测密码为 (0 0) / (1 0) / (3 0) / (7 0)，在四组中任取其一即可。以 (0 0) 为例将其输入到可执行程序 bomb 中，显示 phase4 拆弹成功。

### (3) 任务结果：

结果见下图 13：

```

simpleedu@simpleedu:~/lab3$ ./bomb password.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.

```

图 13 phase4 拆弹结果

故 phase4 密码为：(0 0) / (1 0) / (3 0) / (7 0)

## 4.5 phase5

### (1) 任务描述:

通过 phase5 的反汇编代码寻找对应要求输入的字符串;

### (2) 任务求解:

在 disbomb.txt 文件中查找 phase5 对应的反汇编代码, 所得结果见下图 14:

```
0000000000401062 <phase_5>:
401062: 53                push    %rbx
401063: 48 83 ec 20       sub     $0x20,%rsp
401067: 48 89 fb          mov     %rdi,%rbx
40106a: 64 48 8b 04 25 28 00 mov     %fs:0x28,%rax
401071: 00 00
401073: 48 89 44 24 18    mov     %rax,0x18(%rsp)
401078: 31 c0             xor     %eax,%eax
40107a: e8 9c 02 00 00    callq  40131b <string_length>
40107f: 83 f8 06          cmp     $0x6,%eax
401082: 74 4e             je      4010d2 <phase_5+0x70>
401084: e8 b1 03 00 00    callq  40143a <explode_bomb>
401089: eb 47             jmp     4010d2 <phase_5+0x70>
40108b: 0f b6 0c 03       movzbl (%rbx,%rax,1),%ecx
40108f: 88 0c 24          mov     %cl,(%rsp)
401092: 48 8b 14 24       mov     (%rsp),%rdx
401096: 83 e2 0f          and     $0xf,%edx
401099: 0f b6 92 b0 24 40 00 movzbl 0x4024b0(%rdx),%edx
4010a0: 88 54 04 10       mov     %dl,0x10(%rsp,%rax,1)
4010a4: 48 83 c0 01       add     $0x1,%rax
4010a8: 48 83 f8 06       cmp     $0x6,%rax
4010ac: 75 dd             jne     40108b <phase_5+0x29>
4010ae: c6 44 24 16 00    movb    $0x0,0x16(%rsp)
4010b3: be 5e 24 40 00    mov     $0x40245e,%esi
4010b8: 48 8d 7c 24 10    lea     0x10(%rsp),%rdi
4010bd: e8 76 02 00 00    callq  401338 <strings_not_equal>
4010c2: 85 c0             test    %eax,%eax
4010c4: 74 13             je      4010d9 <phase_5+0x77>
4010c6: e8 6f 03 00 00    callq  40143a <explode_bomb>
4010cb: 0f 1f 44 00 00    nopl    0x0(%rax,%rax,1)
4010d0: eb 07             jmp     4010d9 <phase_5+0x77>
4010d2: b8 00 00 00 00    mov     $0x0,%eax
4010d7: eb b2             jmp     40108b <phase_5+0x29>
4010d9: 48 8b 44 24 18    mov     0x18(%rsp),%rax
4010de: 64 48 33 04 25 28 00 xor     %fs:0x28,%rax
4010e5: 00 00
4010e7: 74 05             je      4010ee <phase_5+0x8c>
4010e9: e8 42 fa ff ff    callq  400b30 <__stack_chk_fail@plt>
4010ee: 48 83 c4 20       add     $0x20,%rsp
4010f2: 5b                pop     %rbx
4010f3: c3                retq
```

图 14 phase5 对应反汇编代码

从 phase5 的反汇编代码可以看出, 该程序主体包含一个循环程序, 具体执行过程如下:

1. 首先处理各种函数的栈帧, 然后将一个字符串储存在寄存器%rdi 中, 移动到%rax 中, 再移动到%rsp 中, 随后调用<string\_length>函数计算字符串长度并将数据储存在%eax 中。判断%eax 中的值是否等于 6, 若等于 6, 则跳转到 4010d2 行, 若不等于 6, 则炸弹爆炸; (401062-401084) (由此可知密码为长度等于 6 的

字符串)

2. 将%eax 的值设置为 0, 随后跳转到 40108b 行, 进入循环流程, 循环过程中: 首先按照循环变量取%rbx 中的一个字符并将其移动到%ecx 中, 随后取%ecx 的后 8 位, 将其移动到%rsp 中, 再将其移动到%rdx 中, 将%edx 中的字符与 0xf 进行按位与运算留下最后四位作为索引。接下来利用留下的最后 4 位与 0x4024b0 进行寻址运算, 将找出的相应字符储存在%edx 中, 最后将%edx 的末 8 位放入%rsp 的后面部分即利用数组存储找到的字符, 利用%rax 记录循环次数, 当其值不等于 6 时, 每次循环增加 1, 其值等于 6 时, 循环结束。利用 gdb 工具查看 0x4024b0 地址下的内容, 见图 15:

```
(gdb) x /s 0x4024b0
0x4024b0 <array.3449>: "maduiersnfotvbylSo you think you can stop the bomb with
ctrl-c, do you?"
```

图 15 0x4024b0 地址对应内容

从上图可知, 该字符串为“maduiersnfotvbyl”, 同时“So you think you can stop the bomb with ctrl-c, do you?”提供了一种新的拆弹方法, 即按 **ctrl+c** 可以直接完成拆弹。(4010d2,4010d7,40108b-4010ac)

3. 由于输入的字符串未知, 故需要进一步研究循环结束之后的反汇编代码, 首先将%rsp 的后 16 个字节设置为 0, 清空字符串后面部分以便进行比较, 随后将地址 0x40245e 对应的内容转移到%esi。利用 gdb 工具查看 0x40245e 地址下的内容, 见图 16:

```
(gdb) x /s 0x40245e
0x40245e: "flyers"
```

图 16 0x40245e 地址对应内容

从上图可知, 输入字符串为“flyers”, 随后将%rsp 的 10 字节偏移的地址转移到%rdi, 这便是上面循环过程求出的字符串;(4010ae-4010b8)

4. 接下来判断两个字符串是否相等, 若不相等则炸弹爆炸, 相等则跳转到 4010d9 行, 至此主体部分基本结束;(4010bd-4010c6, 4010d9) (**phase5** 的目的就是判断输入字符串(每个字符后四位)经过“maduiersnfotvbyl”索引是否能得到“flyers”, 若能得到即拆弹成功, 反之炸弹爆炸)

接下来便是根据索引建立对应关系: 在“maduiersnfotvbyl”中, f、l、y、e、r、s 对应数组下标分别为 9、15、14、5、6、7, 对应的二进制数分别为 1001、1111、1110、0101、0110、0111, 我们需要在每个数字前加 4 位二进制数, 将其变成 8 位, 从而根据 ASCII 码找到对应的字符, 故本题答案不唯一。

由于 ASCII 码表有数字大小限制(即所组成的 8 位二进制数要在 00100000-01111110 之间), 答案组合如下:

第一位可取: )、9、I、Y、i、y 共 6 种;

第二位可取: /、?、O、\_、o 共 5 种;(01111111 超出范围)

第三位可取: .、>、N、^、n、~共 6 种;

第四位可取：%、5、E、U、e、u 共 6 种；

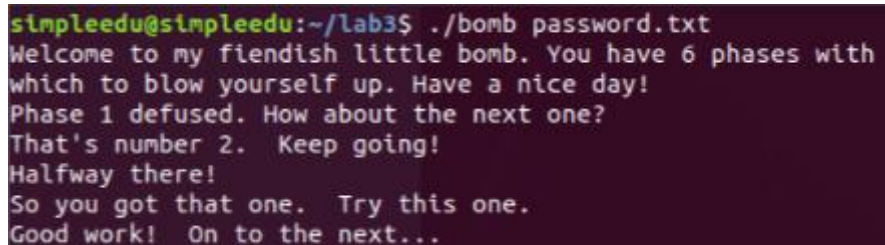
第五位可取：&、6、F、V、f、v 共 6 种；

第六位可取：'、7、G、W、g、w 共 6 种；

故共有  $5 \times 6^5 = 38880$  种组合方式，即 38880 组密码。以 yonuvw 为例将其输入到可执行程序 bomb 中，显示 phase5 拆弹成功。

### (3) 任务结果：

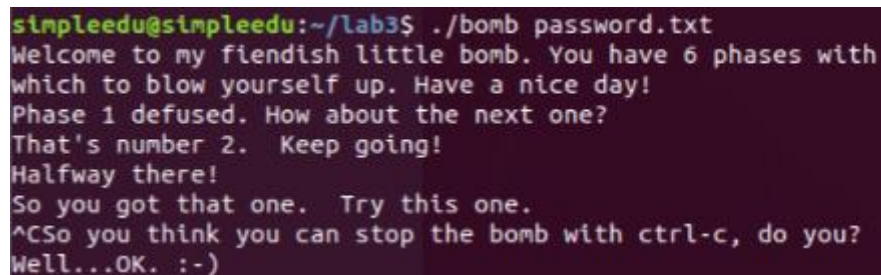
结果见下图 17：



```
simpleedu@simpleedu:~/lab3$ ./bomb password.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...
```

图 17 phase5 拆弹结果

由于 0x4024b0 地址处对应内容提示直接按 **ctrl+c** 也可以成功拆弹，此处进行尝试，结果见下图 18：



```
simpleedu@simpleedu:~/lab3$ ./bomb password.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.
^CSo you think you can stop the bomb with ctrl-c, do you?
Well...OK. :-)
```

图 18 phase5 拆弹结果 (ctrl+c)

故 phase5 密码为：yonuvw（答案共有 38880 种）

## 4.6 phase6

### (1) 任务描述：

通过 phase6 的反汇编代码寻找对应要求输入的字符串；

### (2) 任务求解：

在 disbomb.txt 文件中查找 phase6 对应的反汇编代码，所得结果见下图 19、20、21：



```

00000000004010f4 <phase_6>:
4010f4: 41 56          push    %r14
4010f6: 41 55          push    %r13
4010f8: 41 54          push    %r12
4010fa: 55            push    %rbp
4010fb: 53            push    %rbx
4010fc: 48 83 ec 50    sub     $0x50,%rsp
401100: 49 89 e5       mov     %rsp,%r13
401103: 48 89 e6       mov     %rsp,%rsi
401106: e8 51 03 00 00 callq   40145c <read_six_numbers>
40110b: 49 89 e6       mov     %rsp,%r14
40110e: 41 bc 00 00 00 00 mov     $0x0,%r12d
401114: 4c 89 ed       mov     %r13,%rbp
401117: 41 8b 45 00    mov     0x0(%r13),%eax
40111b: 83 e8 01      sub     $0x1,%eax
40111e: 83 f8 05      cmp     $0x5,%eax
401121: 76 05         jbe     401128 <phase_6+0x34>
401123: e8 12 03 00 00 callq   40143a <explode_bomb>
401128: 41 83 c4 01    add     $0x1,%r12d
40112c: 41 83 fc 06    cmp     $0x6,%r12d
401130: 74 21         je      401153 <phase_6+0x5f>
401132: 44 89 e3       mov     %r12d,%ebx
401135: 48 63 c3      movslq  %ebx,%rax
401138: 8b 04 84       mov     (%rsp,%rax,4),%eax
40113b: 39 45 00      cmp     %eax,0x0(%rbp)
40113e: 75 05         jne     401145 <phase_6+0x51>
401140: e8 f5 02 00 00 callq   40143a <explode_bomb>
401145: 83 c3 01      add     $0x1,%ebx
401148: 83 fb 05      cmp     $0x5,%ebx
40114b: 7e e8         jle     401135 <phase_6+0x41>
40114d: 49 83 c5 04    add     $0x4,%r13
401151: eb c1         jmp     401114 <phase_6+0x20>
401153: 48 8d 74 24 18 lea     0x18(%rsp),%rsi

```

图 19 phase6 对应反汇编代码片段 1

```

401158: 4c 89 f0       mov     %r14,%rax
40115b: b9 07 00 00 00 mov     $0x7,%ecx
401160: 89 ca         mov     %ecx,%edx
401162: 2b 10         sub     (%rax),%edx
401164: 89 10         mov     %edx,(%rax)
401166: 48 83 c0 04    add     $0x4,%rax
40116a: 48 39 f0       cmp     %rsi,%rax
40116d: 75 f1         jne     401160 <phase_6+0x6c>
40116f: be 00 00 00 00 mov     $0x0,%esi
401174: eb 21         jmp     401197 <phase_6+0xa3>
401176: 48 8b 52 08    mov     0x8(%rdx),%rdx
40117a: 83 c0 01      add     $0x1,%eax
40117d: 39 c8         cmp     %ecx,%eax
40117f: 75 f5         jne     401176 <phase_6+0x82>
401181: eb 05         jmp     401188 <phase_6+0x94>
401183: ba d0 32 60 00 mov     $0x6032d0,%edx
401188: 48 89 54 74 20 mov     %rdx,0x20(%rsp,%rsi,2)
40118d: 48 83 c6 04    add     $0x4,%rsi
401191: 48 83 fe 18    cmp     $0x18,%rsi
401195: 74 14         je      4011ab <phase_6+0xb7>
401197: 8b 0c 34       mov     (%rsp,%rsi,1),%ecx
40119a: 83 f9 01      cmp     $0x1,%ecx
40119d: 7e e4         jle     401183 <phase_6+0x8f>
40119f: b8 01 00 00 00 mov     $0x1,%eax
4011a4: ba d0 32 60 00 mov     $0x6032d0,%edx
4011a9: eb cb         jmp     401176 <phase_6+0x82>
4011ab: 48 8b 5c 24 20 mov     0x20(%rsp),%rbx
4011b0: 48 8d 44 24 28 lea     0x28(%rsp),%rax
4011b5: 48 8d 74 24 50 lea     0x50(%rsp),%rsi
4011ba: 48 89 d9       mov     %rbx,%rcx
4011bd: 48 8b 10       mov     (%rax),%rdx
4011c0: 48 89 51 08    mov     %rdx,0x8(%rcx)
4011c4: 48 83 c0 08    add     $0x8,%rax
4011c8: 48 39 f0       cmp     %rsi,%rax

```

图 20 phase6 对应反汇编代码片段 2

4011cb:	74 05	je	4011d2 <phase_6+0xde>
4011cd:	48 89 d1	mov	%rdx,%rcx
4011d0:	eb eb	jmp	4011bd <phase_6+0xc9>
4011d2:	48 c7 42 08 00 00 00	movq	\$0x0,0x8(%rdx)
4011d9:	00		
4011da:	bd 05 00 00 00	mov	\$0x5,%ebp
4011df:	48 8b 43 08	mov	0x8(%rbx),%rax
4011e3:	8b 00	mov	(%rax),%eax
4011e5:	39 03	cmp	%eax,(%rbx)
4011e7:	7d 05	jge	4011ee <phase_6+0xfa>
4011e9:	e8 4c 02 00 00	callq	40143a <explode_bomb>
4011ee:	48 8b 5b 08	mov	0x8(%rbx),%rbx
4011f2:	83 ed 01	sub	\$0x1,%ebp
4011f5:	75 e8	jne	4011df <phase_6+0xeb>
4011f7:	48 83 c4 50	add	\$0x50,%rsp
4011fb:	5b	pop	%rbx
4011fc:	5d	pop	%rbp
4011fd:	41 5c	pop	%r12
4011ff:	41 5d	pop	%r13
401201:	41 5e	pop	%r14
401203:	c3	retq	

图 21 phase6 对应反汇编代码片段 3

由<read-six-numbers>可以推断出本关应输入六个数字，该反汇编代码具体执行过程如下：

1. 首先处理各种函数的栈帧，然后将%rsp 中储存的地址移动到%r13 和%rsi 中，随后输入六个数字并将其存储在%rsp 中，将%rsp 的地址移动到%r14，并将%r12d 中的值置为 0，将%r13 中储存的地址移动到%rbp，并将%r13 中的值移动到%eax 中，随后%eax 中的值减 1，将%eax 中的值和 5 比较，若%eax 中的值小于等于 5，则跳转到 401128 行，反之炸弹爆炸;(4010f4-401123)（由此可知输入的 6 个数均小于等于 6）

2.接下来是两个循环的嵌套：对于外层循环，循环体从 401114 行开始到 401151 行结束，循环结束后跳转到 401153 行，循环以%r12d 中的值为循环变量，在循环中，%r13 用于存放每次遍历的数据，真正体现循环目的的部分为 401132-40114b 行，其将%r12d 中的值转移到%ebx 中。随后进入内层循环部分，循环以%ebx 中的值为循环变量，首先将%ebx 中的值转移到%rax 中，随后计算%rsp+4\*%rax 的值并将其转移到%eax 中，接着比较%eax 和%rbp 中的值，若相等则炸弹爆炸，反之跳转到 401145 行，接下来%ebx 中的值自增 1，比较%ebx 中的值和 5 的大小，若小于等于 5，则跳转到 401135 行，继续循环，反之程序继续，即双层循环嵌套部分结束；（401114-40114b）

%r12d 中的值从 0 循环到 5，%ebx 中的值从%r12d+1 循环到 5，为便于理解将其转化为 C++程序，代码如下：

```
int numbers[6]={}; //输入值
for (int a = 0;a < 6;a++){ //a in %r12d
    if(numbers[a] > 6)
        explode_bomb(); //炸弹爆炸
    for(int b = a+1;b < 6;b++) //b in %ebx
```



```

        if(numbers[b] == numbers[a])
            explode_bomb();
    }

```

3. 双层循环嵌套部分结束后跳转到 401153 行，将 %rsp 中的 24 字节偏移转移到 %rsi 中，%r14 中的值转移到 %rax 中，将 %ecx 与 %edx 中的值都置为 7，随后将 %edx 中的值减去 %rax 中的值，将 %edx 中的值转移到 %rax 中，%rax 的地址增加 4，即取下一位数，接下来比较 %rsi 和 %rax 中的值，若不相等，则跳转到 401160 行继续循环，反之程序继续。%rsi 中存储的是六个数中最后一个数的结束地址，该循环的目的是用 7 分别减去每一个数，并将结果存储在 %rax 中；(401153-40116d)

4. 接下来将 %esi 中的值置为 0，跳转到 401197 行，将 %rsi+%rsp 的值转移到 %ecx 中，比较 %ecx 中的值和 1 的大小，若其小于等于 1，则跳转到 401183 行，反之继续。将 %eax 中的值置为 1，将 0x6032d0 地址对应的内容转移到 %edx 中，随后跳转到 401176 行，利用 gdb 工具查看 0x6032d0 地址下的内容，见图 22：

(gdb) x /24wx 0x6032d0				
0x6032d0	<node1>:	0x0000014c	0x00000001	0x006032e0
0x6032e0	<node2>:	0x000000a8	0x00000002	0x006032f0
0x6032f0	<node3>:	0x00000039c	0x00000003	0x00603300
0x603300	<node4>:	0x0000002b3	0x00000004	0x00603310
0x603310	<node5>:	0x0000001dd	0x00000005	0x00603320
0x603320	<node6>:	0x0000001bb	0x00000006	0x00000000

图 22 0x6032d0 地址对应内容

由图中<node1>可知：这个存储结构应该是链表，第一列存储对应值，第三列存储指针指向的下一结点的索引；(40116f-4011a9)

由后续反汇编代码可以推测出其为排序算法，为便于理解将其转化为 C++ 程序，代码如下：

```

for(int c = 0; c <= 5; c++){//c in %rsi
    for(int i = 0; numbers[c] > i; i++){
        node = node -> next;
    }
    numbers[c] = node;
}

```

对于存储在<node>中的值：

```

node1=0x14c=332;
node2=0xa8=168;
node3=0x39c=927;
node4=0x2b3=691;
node5=0x1dd=477;

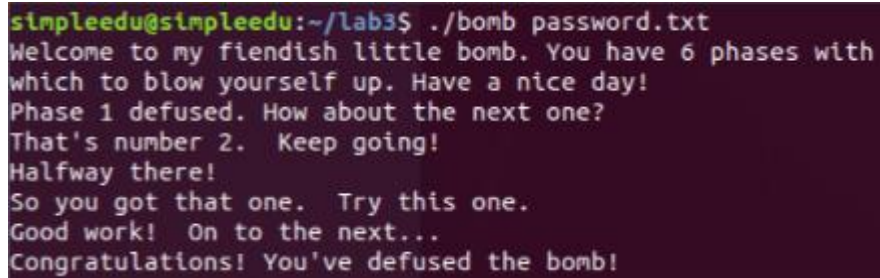
```

node6=0x1bb=443;

按其存储的值对 node 进行降序排序，得到序列如下：3 4 5 6 1 2，同时在之前的代码分析中发现需要用 7 减去每一个数字，得到最终序列：4 3 2 1 6 5。将其输入到可执行程序 bomb 中，显示 phase6 拆弹成功。

### (3) 任务结果：

结果见下图 23：



```
simpleedu@simpleedu:~/lab3$ ./bomb password.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...
Congratulations! You've defused the bomb!
```

图 23 phase6 拆弹结果

故 phase6 密码为：4 3 2 1 6 5

## 4.7 隐藏关卡

### (1) 任务描述：

通过 phase\_defused 的反汇编代码寻找对应要求输入的数据；

### (2) 任务求解：

在 disbomb.txt 文件中 phase6 的反汇编代码后还有 fun7 函数和 secret\_phase 函数的反汇编代码，因此隐藏关卡一定就在其中，并且在拆弹过程中一定会触发 secret\_phase 函数，由于正常输入密码也可以成功拆除所有炸弹，所以在 disbomb.txt 文件中直接搜索，确定其在第几个问题中出现，搜索结果见下图 24：

```

00000000004015c4 <phase_defused>:
4015c4: 48 83 ec 78      sub    $0x78,%rsp
4015c8: 64 48 8b 04 25 28 00 mov    %fs:0x28,%rax
4015cf: 00 00
4015d1: 48 89 44 24 68      mov    %rax,0x68(%rsp)
4015d6: 31 c0            xor    %eax,%eax
4015d8: 83 3d 81 21 20 00 06 cmpl   $0x6,0x202181(%rip)    # 603760
<num_input_strings>
4015df: 75 5e            jne    40163f <phase_defused+0x7b>
4015e1: 4c 8d 44 24 10      lea    0x10(%rsp),%r8
4015e6: 48 8d 4c 24 0c      lea    0xc(%rsp),%rcx
4015eb: 48 8d 54 24 08      lea    0x8(%rsp),%rdx
4015f0: be 19 26 40 00      mov    $0x402619,%esi
4015f5: bf 70 38 60 00      mov    $0x603870,%edi
4015fa: e8 f1 f5 ff ff      callq  400bf0 <__isoc99_sscanf@plt>
4015ff: 83 f8 03          cmp     $0x3,%eax
401602: 75 31            jne    401635 <phase_defused+0x71>
401604: be 22 26 40 00      mov    $0x402622,%esi
401609: 48 8d 7c 24 10      lea    0x10(%rsp),%rdi
40160e: e8 25 fd ff ff      callq  401338 <strings_not_equal>
401613: 85 c0            test   %eax,%eax
401615: 75 1e            jne    401635 <phase_defused+0x71>
401617: bf f8 24 40 00      mov    $0x4024f8,%edi
40161c: e8 ef f4 ff ff      callq  400b10 <puts@plt>
401621: bf 20 25 40 00      mov    $0x402520,%edi
401626: e8 e5 f4 ff ff      callq  400b10 <puts@plt>
40162b: b8 00 00 00 00      mov    $0x0,%eax
401630: e8 0d fc ff ff      callq  401242 <secret_phase>
401635: bf 58 25 40 00      mov    $0x402558,%edi
40163a: e8 d1 f4 ff ff      callq  400b10 <puts@plt>
40163f: 48 8b 44 24 68      mov    0x68(%rsp),%rax
401644: 64 48 33 04 25 28 00 xor    %fs:0x28,%rax
40164b: 00 00
40164d: 74 05            je     401654 <phase_defused+0x90>
40164f: e8 dc f4 ff ff      callq  400b30 <__stack_chk_fail@plt>
401654: 48 83 c4 78      add    $0x78,%rsp
401658: c3              retq
401659: 90              nop
40165a: 90              nop
40165b: 90              nop

```

图 24 搜索结果

从图中 4015fa-401602 行可知，此处读入了一个数据，随后判断用于记录输入数量的%eax 中的值是否为 3，若不等于 3，则直接跳转到 401635 行，反之继续执行。随后在 401604-40160e 行，将 0x402622 地址对应的内容转移到%esi 中，并进行字符串比较操作，利用 gbd 工具查看 0x402622 地址下的内容，见图 25：

```

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

```

图 25 0x402622 地址对应内容

该内容即为第三个数据应输入的“DrEvil”，由于只有 phase3 和 phase4 的密码为两个数据组成，故将“DrEvil”分别加在两个密码后进行测试以确定开启方式，最后发现在 phase4 的密码后添加“DrEvil”可以开启隐藏关卡，见下图 26：

```

simpleedu@simpleedu:~/lab3$ ./bomb password.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...

```

图 26 成功开启隐藏关卡

为成功通过隐藏关卡，接下来需要分析 secret\_phase 函数的反汇编代码，见下图 27：

```

000000000401242 <secret_phase>:
401242: 53                push    %rbx
401243: e8 56 02 00 00    callq  40149e <read_line>
401248: ba 0a 00 00 00    mov     $0xa,%edx
40124d: be 00 00 00 00    mov     $0x0,%esi
401252: 48 89 c7          mov     %rax,%rdi
401255: e8 76 f9 ff ff    callq  400bd0 <strtol@plt>
40125a: 48 89 c3          mov     %rax,%rbx
40125d: 8d 40 ff          lea     -0x1(%rax),%eax
401260: 3d e8 03 00 00    cmp     $0x3e8,%eax
401265: 76 05            jbe     40126c <secret_phase+0x2a>
401267: e8 ce 01 00 00    callq  40143a <explode_bomb>
40126c: 89 de -- -- --    mov     %ebx,%esi
401248: ba 0a 00 00 00    mov     $0xa,%edx
40124d: be 00 00 00 00    mov     $0x0,%esi
401252: 48 89 c7          mov     %rax,%rdi
401255: e8 76 f9 ff ff    callq  400bd0 <strtol@plt>
40125a: 48 89 c3          mov     %rax,%rbx
40125d: 8d 40 ff          lea     -0x1(%rax),%eax
401260: 3d e8 03 00 00    cmp     $0x3e8,%eax
401265: 76 05            jbe     40126c <secret_phase+0x2a>
401267: e8 ce 01 00 00    callq  40143a <explode_bomb>
40126c: 89 de            mov     %ebx,%esi
40126e: bf f0 30 60 00    mov     $0x6030f0,%edi
401273: e8 8c ff ff ff    callq  401204 <fun7>
401278: 83 f8 02          cmp     $0x2,%eax
40127b: 74 05            je      401282 <secret_phase+0x40>
40127d: e8 b8 01 00 00    callq  40143a <explode_bomb>
401282: bf 38 24 40 00    mov     $0x402438,%edi
401287: e8 84 f8 ff ff    callq  400b10 <puts@plt>
40128c: e8 33 03 00 00    callq  4015c4 <phase_defused>
401291: 5b                pop     %rbx
401292: c3                retq
401293: 90                nop
401294: 90                nop
401295: 90                nop

```

图 27 secret\_phase 函数反汇编代码

从图中 40126e-401278 行可知，此处将地址 0x6030f0 对应的内容转移到%edi 中，并将其传入函数 fun7 中进行计算，由于其将返回值与 2 进行比较以决定后续是否跳转，故可推测返回值为 2，利用 gbd 工具查看 0x6030f0 地址下的内容，见图 28：



```
(gdb) x /48wx 0x6030f0
0x6030f0 <n1>: 0x00000024      0x00000000      0x00603110      0x00000000
0x603100 <n1+16>:      0x00603130      0x00000000      0x00000000      0x00000000
0x603110 <n21>: 0x00000008      0x00000000      0x00603190      0x00000000
0x603120 <n21+16>:      0x00603150      0x00000000      0x00000000      0x00000000
0x603130 <n22>: 0x00000032      0x00000000      0x00603170      0x00000000
0x603140 <n22+16>:      0x006031b0      0x00000000      0x00000000      0x00000000
0x603150 <n32>: 0x00000016      0x00000000      0x00603270      0x00000000
0x603160 <n32+16>:      0x00603230      0x00000000      0x00000000      0x00000000
0x603170 <n33>: 0x0000002d      0x00000000      0x006031d0      0x00000000
0x603180 <n33+16>:      0x00603290      0x00000000      0x00000000      0x00000000
0x603190 <n31>: 0x00000006      0x00000000      0x006031f0      0x00000000
0x6031a0 <n31+16>:      0x00603250      0x00000000      0x00000000      0x00000000
```

图 28 0x6030f0 地址对应内容

接下来研究 fun7 函数的反汇编代码，见下图 29:

```
000000000401204 <fun7>:
401204: 48 83 ec 08      sub    $0x8,%rsp
401208: 48 85 ff         test   %rdi,%rdi
40120b: 74 2b           je     401238 <fun7+0x34>
40120d: 8b 17           mov    (%rdi),%edx
40120f: 39 f2           cmp    %esi,%edx
401211: 7e 0d           jle    401220 <fun7+0x1c>
401213: 48 8b 7f 08      mov    0x8(%rdi),%rdi
401217: e8 e8 ff ff ff   callq  401204 <fun7>
40121c: 01 c0           add    %eax,%eax
40121e: eb 1d           jmp     40123d <fun7+0x39>
401220: b8 00 00 00 00   mov    $0x0,%eax
401225: 39 f2           cmp    %esi,%edx
401227: 74 14           je     40123d <fun7+0x39>
401229: 48 8b 7f 10      mov    0x10(%rdi),%rdi
40122d: e8 d2 ff ff ff   callq  401204 <fun7>
401232: 8d 44 00 01      lea    0x1(%rax,%rax,1),%eax
401236: eb 05           jmp     40123d <fun7+0x39>
401238: b8 ff ff ff ff   mov    $0xffffffff,%eax
40123d: 48 83 c4 08      add    $0x8,%rsp
401241: c3             retq
```

图 29 fun7 函数反汇编代码

最后发现这是一个树状链表的递归（二叉搜索树），只需要保持输出一直为 2 即可，遍历后得到最终结果为 20/22。以 22 为例，将其输入到可执行程序 bomb 中，显示隐藏关卡已通过。

### (3) 任务结果:

结果见下图 30:

```
simpleedu@simpleedu:~/lab3$ ./bomb password.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...
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
```

图 30 隐藏关卡通关结果

故隐藏关卡密码为：20/22

## 五、实验总结及心得体会

（拆弹操作总结，实验中遇到的问题及解决方法等）

通过本次拆弹实验，对程序的机器级表示和处理方法有了更加深刻的理解，对于使用汇编语言实现选择语句、循环语句等的方法掌握的更加深入透彻。在实际拆弹的过程中，我学会使用了功能强大的 `gdb` 调试工具以及通过 `objdump` 工具查看可执行程序的反汇编代码并通过 `>` 指令定向写入文本。

在解决 `phase1` 和 `phase2` 的过程中，由于对 `gdb` 工具使用方法不够熟悉，犯了许多错误，后通过回顾老师上课录制的视频讲解，操作逐渐熟练起来，随后通过查找资料了解了指令中各个字符的具体含义，可以根据自己的需要写出相应的指令。在解决 `phase4` 的过程中，由于 `func4` 函数使用了递归，直接研究汇编代码，很难得到最终结果，于是考虑利用 C++ 代码转化的方法使其便于理解与计算。在转化的过程中一定要时刻注意对应变量存储在哪个寄存器中，在变量存储位置发生变化时，可以通过注释的方法不断提醒自己，减少错误的出现。

在解决 `phase5` 的过程中，我最开始没有理解保留输入字符最后 4 位二进制编码的作用，经过不断重复研究反汇编代码，发现其起到索引的作用，用以在给定的字符串（字典）中找到对应的字母，最终得到“flyers”的目标字符串。同时在最后利用 ASCII 码确定每位字符具体对应情况时，我起初忽略了 ASCII 码具有上下限的问题，随后注意到这一情况，并对 ASCII 码转换操作的理解更加深刻。在解决 `phase6` 的过程中，这里的链表排序，利用每一结点存储值的大小进行比较，并返回对应序号的顺序。此外最后还需要对每一个序号分别用 7 去减，才能得到正确结果，过程较为复杂，需要静下心来仔细研究分析。

对于最终的隐藏关卡，通过 `secret_phase` 函数的触发位置进行查找，可以发现用于开启隐藏关卡的密码“DrEvil”，并通过密码数据个数确定其只能添加在 `phase3` 或 `phase4` 的后面，通过分别测试，确定将其添加在 `phase4` 后可以开启隐藏关卡，随后通过研究 `secret_phase` 函数和 `fun7` 函数的反汇编代码，找到通过隐藏关卡的正确密码，这种抽丝剥茧、逐步获得真相的过程带给了我无限的乐趣。