

哈尔滨工业大学

实验报告

实验（三）

题 目 Binary Bomb

二进制炸弹

专 业 计算机

学 号 1180300308

班 级 03003

学 生 刘义

指 导 教 师 史先俊

实 验 地 点 G712

实 验 日 期 10.23

计算机科学与技术学院

目 录

第 1 章 实验基本信息	- 3 -
1.1 实验目的.....	- 3 -
1.2 实验环境与工具.....	- 3 -
1.2.1 硬件环境.....	- 3 -
1.2.2 软件环境.....	- 3 -
1.2.3 开发工具.....	- 3 -
1.3 实验预习.....	- 3 -
第 2 章 实验环境建立	- 5 -
2.1 UBUNTU 下 CODEBLOCKS 反汇编（10 分）	- 5 -
2.2 UBUNTU 下 EDB 运行环境建立（10 分）	- 5 -
第 3 章 各阶段炸弹破解与分析	- 7 -
3.1 阶段 1 的破解与分析.....	- 7 -
3.2 阶段 2 的破解与分析.....	- 8 -
3.3 阶段 3 的破解与分析.....	- 10 -
3.4 阶段 4 的破解与分析.....	- 12 -
3.5 阶段 5 的破解与分析.....	- 15 -
3.6 阶段 6 的破解与分析.....	- 17 -
3.7 阶段 7 的破解与分析(隐藏阶段).....	- 23 -
第 4 章 总结.....	- 28 -
4.1 请总结本次实验的收获.....	- 28 -
4.2 请给出对本次实验内容的建议.....	- 28 -
参考文献.....	- 29 -

第 1 章 实验基本信息

1.1 实验目的

熟练掌握计算机系统的 ISA 指令系统与寻址方式

熟练掌握 Linux 下调试器的反汇编调试跟踪分析机器语言的方法

增强对程序机器级表示、汇编语言、调试器和逆向工程等的理解

1.2 实验环境与工具

1.2.1 硬件环境

X64 CPU; 2GHz; 2G RAM; 256GHD Disk 以上

1.2.2 软件环境

Windows7 64 位以上; VirtualBox/Vmware 11 以上; Ubuntu 16.04 LTS 64 位/
优麒麟 64 位;

1.2.3 开发工具

GDB/OBJDUMP; EDB; KDD 等

1.3 实验预习

认真预习实验指导书 (PPT 或 PDF)

了解实验的目的、实验环境与软硬件工具、实验操作步骤, 复习与实验有关的理论知识。

写出 C 语言下包含字符串比较、循环、分支 (含 switch)、函数调用、递归、指针、结构、链表等的例子程序 sample.c。

生成执行程序 sample.out。

用 gcc -S 或 CodeBlocks 或 GDB 或 OBJDUMP 等, 反汇编, 比较。列出每

一部分的 C 语言对应的汇编语言。

修改编译选项-O (缺省 2)、Og、O0、O1、O2、O3、Og, -m32/m64。再次查看生成的汇编语言与原来的区别。

堆栈访问[rbp+-n]或[rsp+n]。-fno-omit-frame-pointer。

GDB 命令详解 - tui 模式 ^XA 切换 layout 改变等等

有目的地学习: 看 VS 的功能 GDB 命令用什么?

第 2 章 实验环境建立

2.1 Ubuntu 下 CodeBlocks 反汇编 (10 分)

CodeBlocks 运行 hellolinux.c。反汇编查看 printf 函数的实现。

要求：C、ASM、内存(显示 hello 等内容)、堆栈（call printf 前）、寄存器同时在一个窗口。

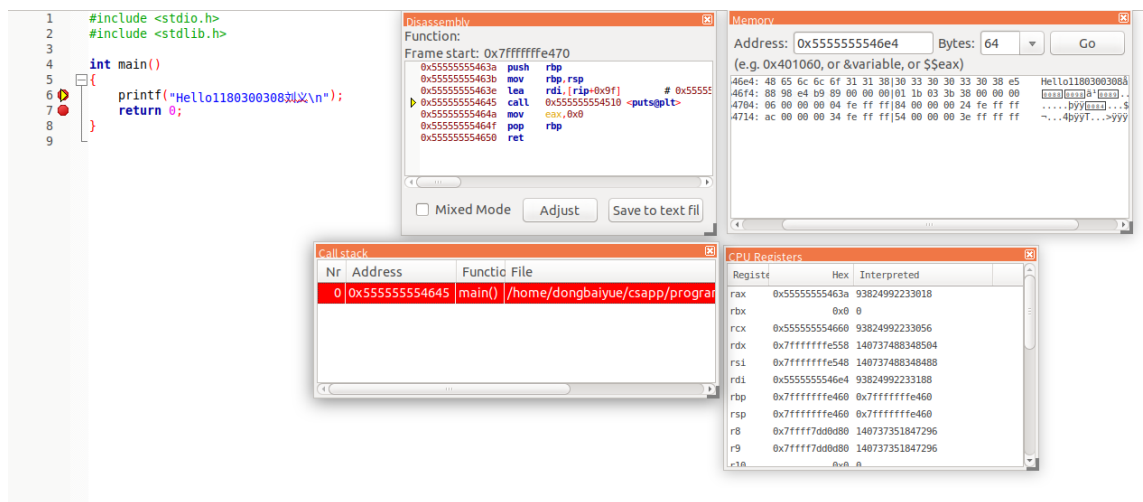


图 2-1 Ubuntu 下 CodeBlocks 反汇编截图

2.2 Ubuntu 下 EDB 运行环境建立 (10 分)

用 EDB 调试 hellolinux.c 的执行文件，截图，要求同 2.1

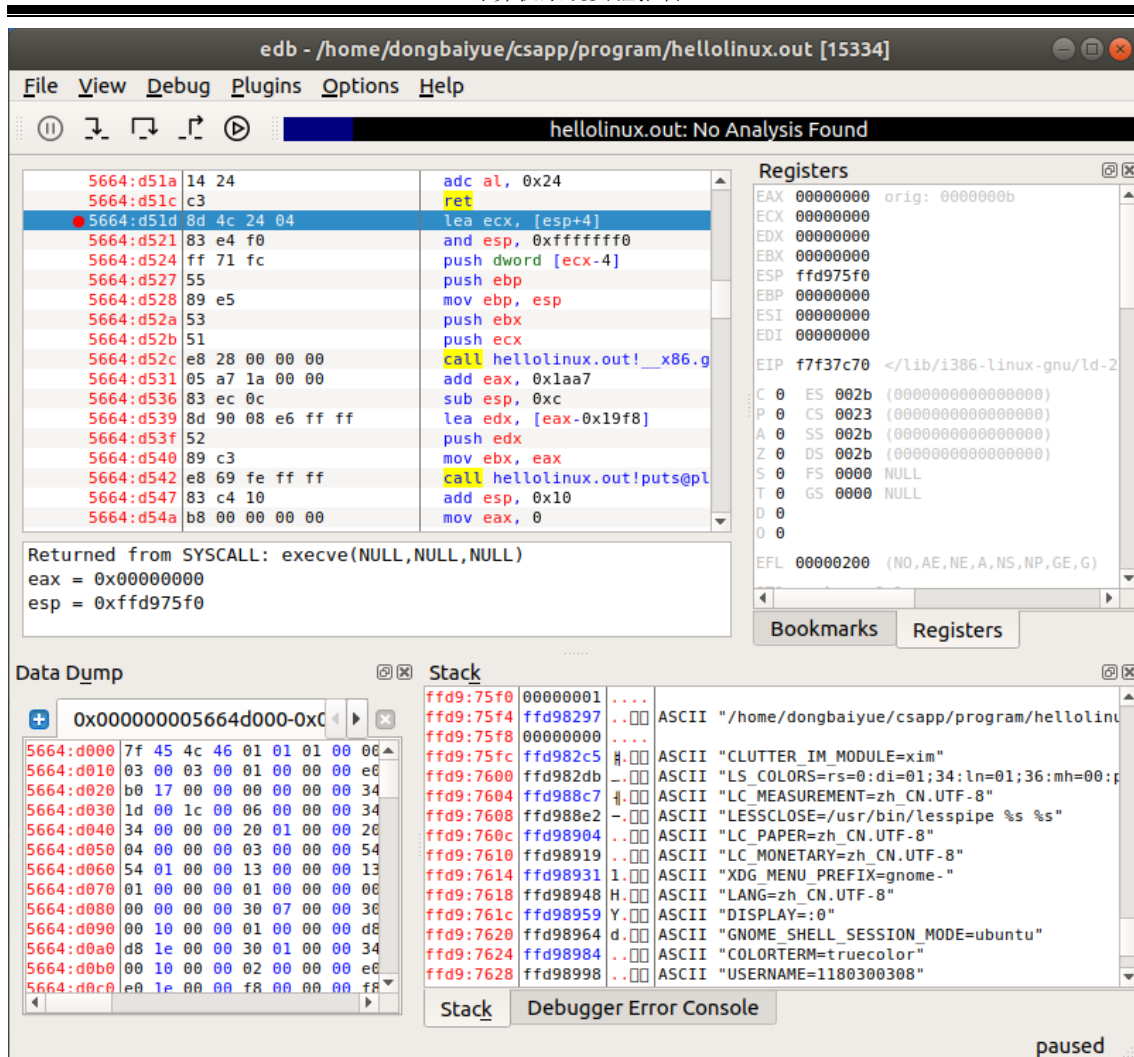


图 2-2 Ubuntu 下 EDB 截图

第 3 章 各阶段炸弹破解与分析

每阶段 15 分，密码 10 分，分析 5 分，总分不超过 80 分

3.1 阶段 1 的破解与分析

密码如下：Brownie, you are doing a heck of a job.

破解过程：

查看 main 函数第一个炸弹部分（如下图）。调用<read_line>函数，函数返回值在寄存器 rax 中，然后将 rax 的值赋给 rdi，这样 rdi 中就存储着用户输入的内容。之后进入函数 phase_1。

```
callq 40194e <read_line>
mov    %rax,%rdi
callq 4013f5 <phase_1>
callq 401a7f <phase_defused>
```

查看 phase_1 函数汇编代码如下

```
00000000004013f5 <phase_1>:
4013f5: 55                push    %rbp
4013f6: 48 89 e5          mov     %rsp,%rbp
4013f9: be 50 31 40 00    mov     $0x403150,%esi
4013fe: e8 ee 03 00 00    callq  4017f1 <strings_not_equal>
401403: 85 c0             test    %eax,%eax
401405: 75 02             jne     401409 <phase_1+0x14>
401407: 5d                pop     %rbp
401408: c3                retq
401409: e8 e2 04 00 00    callq  4018f0 <explode_bomb>
40140e: eb f7             jmp     401407 <phase_1+0x12>
```

```
mov     $0x403150,%esi
```

上面这条指令将首地址为 0x403150 的存储空间的值赋给寄存器 esi

然后 callq 4017f1 <strings_not_equal>

这条指令调用函数<strings_not_equal>判断字符串是否正确，猜测其为比较寄存器 rdi 和寄存器 esi 的值。

查看寄存器 esi 的内容，得到结果：

```
(gdb) x/s $esi
0x403150: "Brownie, you are doing a heck of a job."
```

3.2 阶段 2 的破解与分析

密码如下：1 2 4 8 16 32

破解过程：

查看函数 phase_2 汇编代码：

首先

```
; void __fastcall phase_2(char *input)
public phase_2
phase_2 proc near

    numbers= dword ptr -30h

    input = rdi                ; char *
    ; __unwind {
    push    rbp
    mov     rbp, rsp
    push    rbx
    sub     rsp, 28h
    lea     rsi, [rbp+numbers] ; numbers
    call    read_six_numbers
    cmp     [rbp+numbers], 1
    jnz     short loc_40142F
```

```
loc_40142F:
call     explode_bomb
```

原 c 代码等价于：

```
int numbers[6];
```

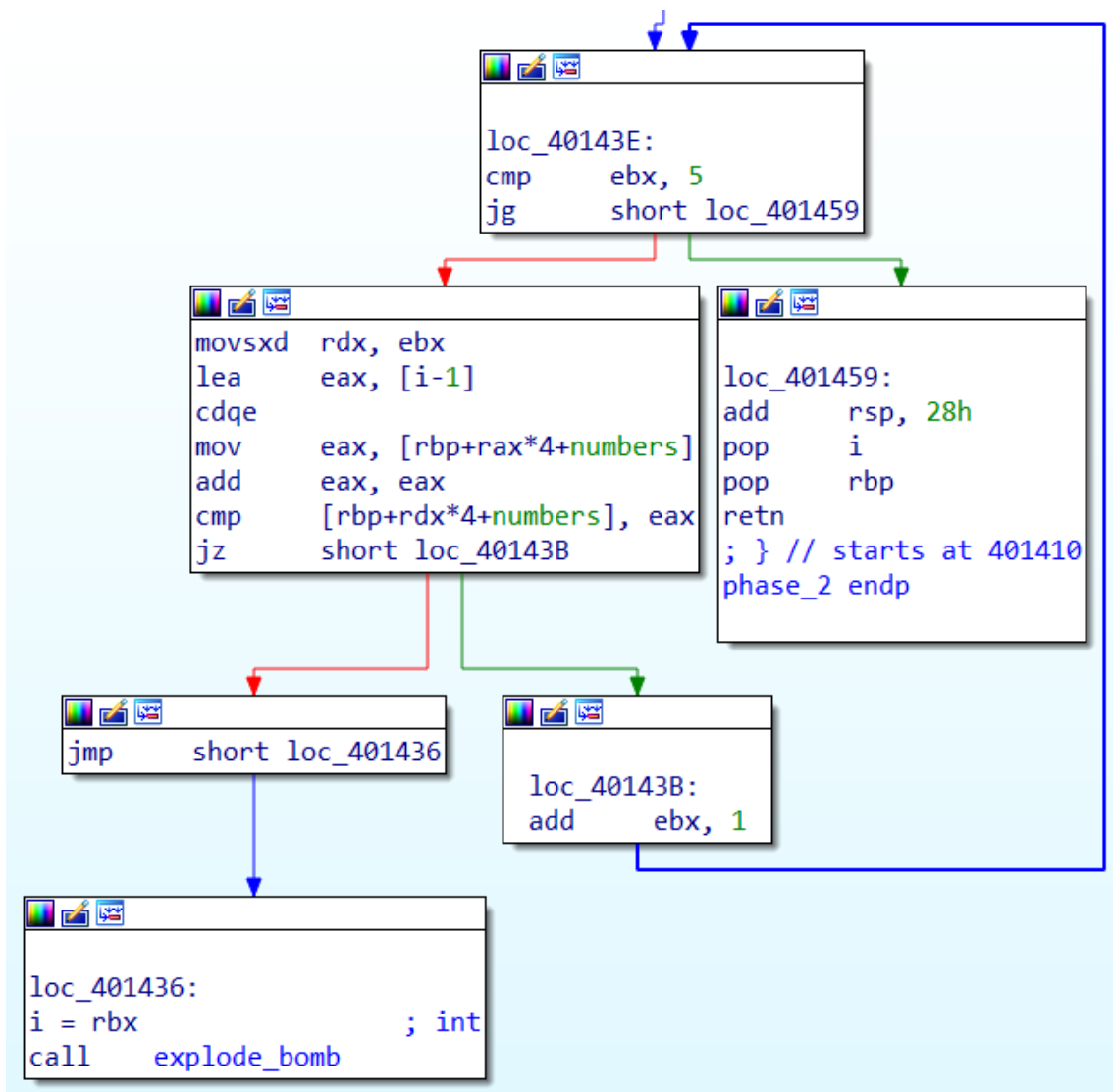
```
read_six_numbers(input, numbers);
```



```
if ( numbers[0] != 1 ) explode_bomb();
```

```
//读入 6 个数字到数组 numbers, numbers[0] = 1, 否则爆炸
```

然后



原 c 代码等价于:

```
for ( i = 1; i <= 5; ++i )
```

```
{
```

```
    if ( numbers[i] != 2 * numbers[i - 1] ) explode_bomb();
```

```
}
```

```
//每个数组元素等于前面元素值的 2 倍, 否则爆炸
```

密码为：1 2 4 8 16 32

3.3 阶段 3 的破解与分析

密码如下：0 283

破解过程：

查看函数 phase_3 汇编代码：

首先

```
; void __fastcall phase_3(char *input)
public phase_3
phase_3 proc near

    val= dword ptr -8
    index= dword ptr -4

    input = rdi                ; char *
    ; __unwind {
    push    rbp
    mov     rbp, rsp
    sub     rsp, 10h
    lea     rcx, [rbp+val]
    lea     rdx, [rbp+index]
    mov     esi, offset aDD ; "%d %d"
    mov     eax, 0
    call    ___isoc99_sscanf
    numScanned = rax           ; int
    cmp     eax, 1
    jle     short loc_401494
```

0040148D default case

```
loc_401494:
numScanned = rax           ; int
call    explode_bomb
```

原 c 代码等价于：

```
if(___isoc99_sscanf(input, "%d %d", &index, &val) <= 1 ) explode_bomb();
```

//读入两个整数，index 与 val，当读入数字少于等于 1 时爆炸

然后

是 switch 函数的跳转，原 c 代码等价于：

```
switch (index )
{
    case 0:
        v1 = 283;
        break;
    case 1:
    case 4:
        v1 = 591;
        break;
    case 2:
        v1 = 863;
        break;
    case 3:
        v1 = 250;
        break;
    case 5:
        v1 = 981;
        break;
    case 6:
        v1 = 943;
        break;
    case 7:
        v1 = 616;
```

```
break;

default:

    explode_bomb();

return;

}

if ( val != v1 )

    explode_bomb();
```

可以看出有若干可能的密码，0 283 是其中一个。

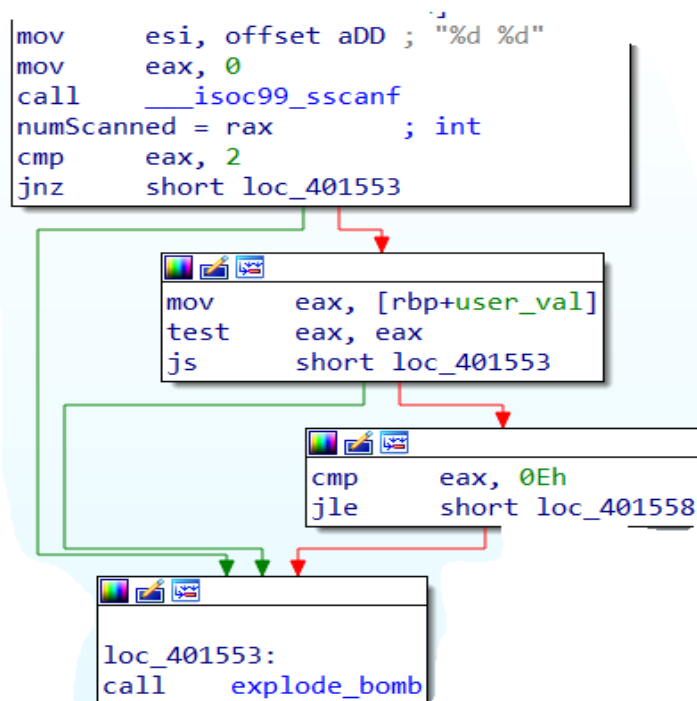
3.4 阶段 4 的破解与分析

密码如下：8 35

破解过程：

查看函数 phase_4 汇编代码：

首先



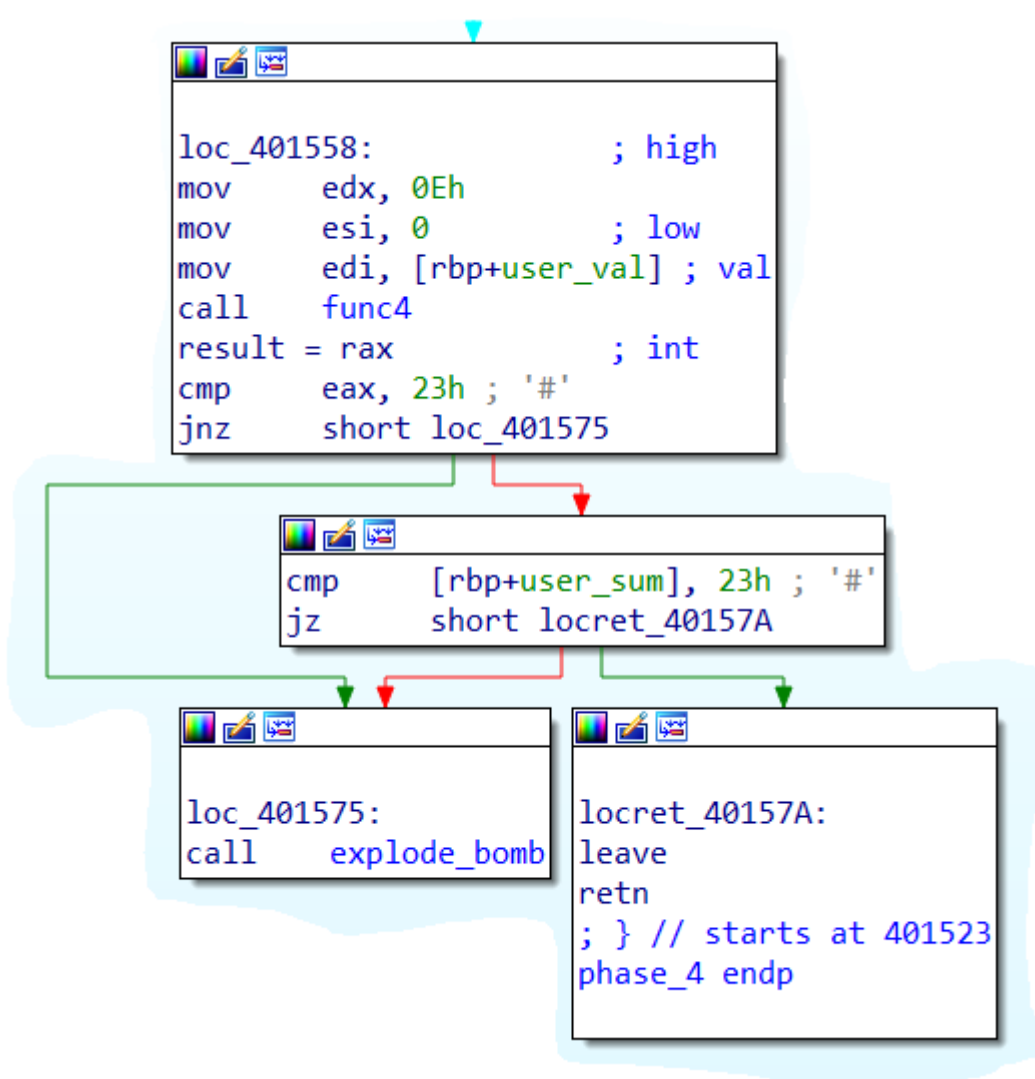
原 c 代码等价于：

```
if (__isoc99_sscanf(input, "%d %d", &user_val, &user_sum) != 2 || user_val < 0 ||
    user_val > 14 )
```

```
{
    explode_bomb();
}
```

//输入两个数字，user_val 与 user_sum，在 0~14 之间，否则爆炸

然后



原 c 代码等价于：

```
if ( func4(user_val, 0, 14) != 35 || user_sum != 35 )
    explode_bomb();
```

//应使 func4(user_val, 0, 14) = 35 & user_sum = 35，否则爆炸

再看 func4 的汇编代码：

```
; int __fastcall func4(int val, int low, int high)
public func4
func4 proc near
val = rdi          ; int
low = rsi          ; int
high = rdx         ; int
; __unwind {
push    rbp
mov     rbp, rsp
push    rbx
sub     rsp, 8
mov     eax, edx
sub     eax, esi
mov     ebx, eax
shr     ebx, 1Fh
add     ebx, eax
sar     ebx, 1
add     ebx, esi
mid = rbx          ; int
cmp     ebx, edi
jg      short loc_40150B
```

```
jnl     short loc_401517
```

```
loc_40150B:          ; high
val = rdi          ; int
low = rsi          ; int
high = rdx         ; int
mid = rbx          ; int
lea     edx, [mid-1]
call    func4
add     ebx, eax
jmp     short loc_401502
```

```
loc_401517:          ; low
val = rdi          ; int
low = rsi          ; int
high = rdx         ; int
mid = rbx          ; int
lea     esi, [mid+1]
call    func4
add     ebx, eax
jmp     short loc_401502
; } // starts at 4014E4
func4 endp
```

原 c 代码等价于：

```
int __fastcall func4(int val, int low, int high)
```

```
{
```

```
    int v3; // ebx
```

```
v3 = low + (high - low) / 2;
if ( v3 > val )
{
    v3 += func4(val, low, v3 - 1);
}
else if ( v3 < val )
{
    v3 += func4(val, v3 + 1, high);
}
return v3;
}
```

编写相应递归程序，得出结果：func4(8, 0, 14) = 35

密码为：8 35

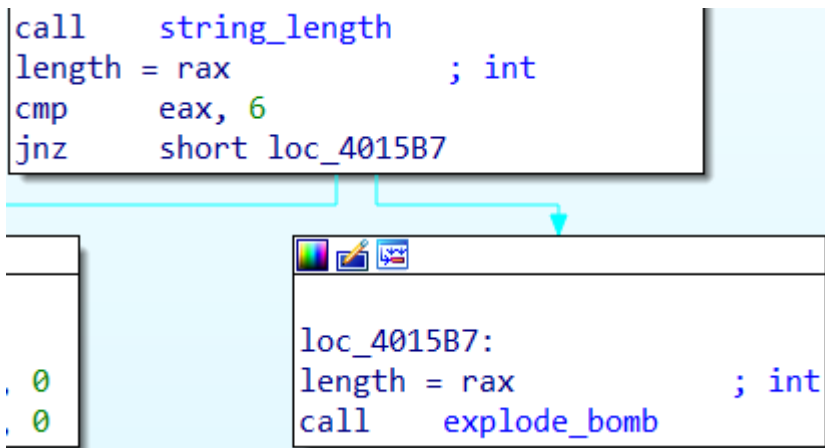
3.5 阶段 5 的破解与分析

密码如下：121212

破解过程：

查看函数 phase_5 汇编代码：

首先

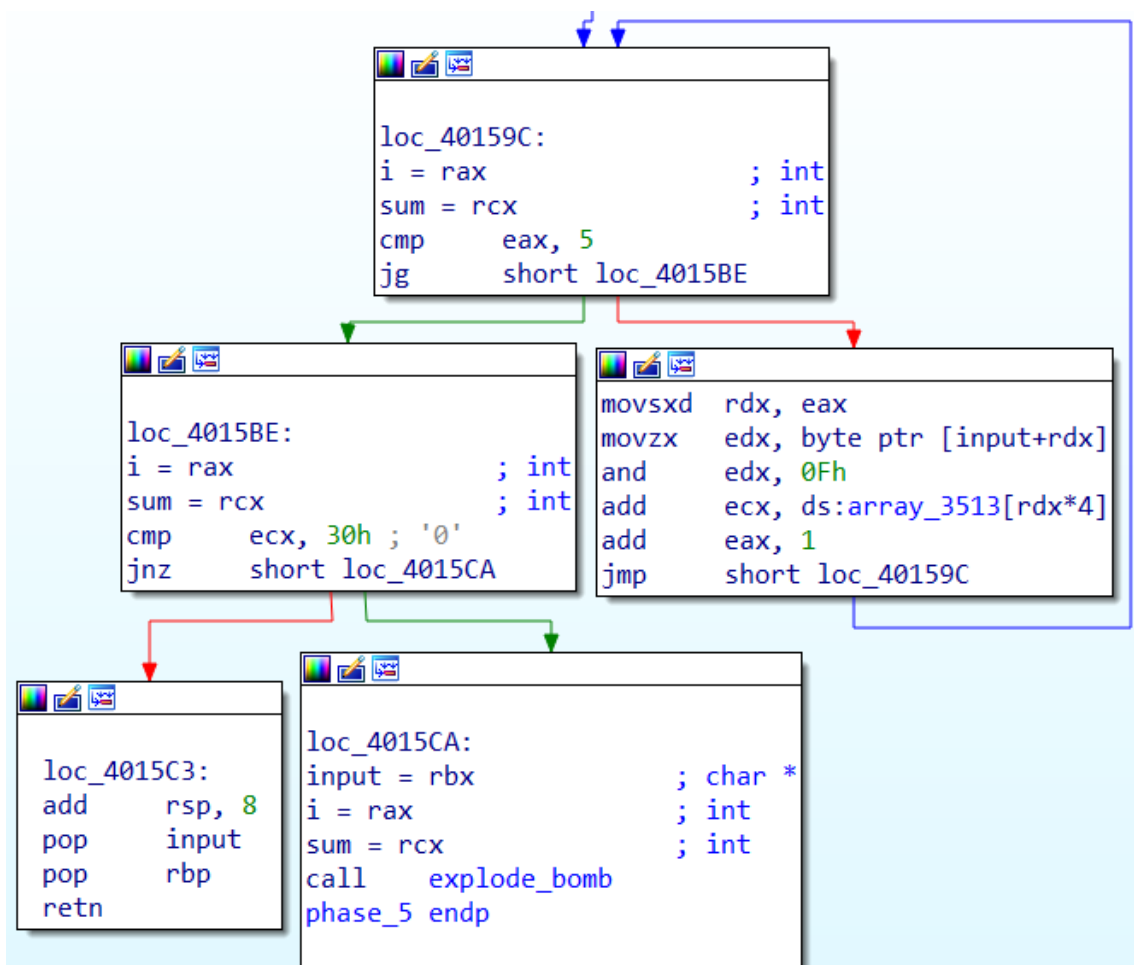


原 c 代码应等价于:

```
if ( string_length(input) != 6 )  explode_bomb();
```

//输入字符串长度为 6，否则爆炸

然后



原 c 代码应等价于:

```
v1 = 0;
for ( i = 0; i <= 5; ++i )
    v1 += array_3513[input[i] & 0xF];
if ( v1 != 48 ) explode_bomb();
```

//显然, 要使 v1 的值为 48, 6 个数组元素 array_3513[] 的值相加为: 48

查看数组 array_3513[] 的值(4 字节一个数组元素):

```
02 00 00 00 0A 00 00 00 06 00 00 00 01 00 00 00
0C 00 00 00 10 00 00 00 09 00 00 00 03 00 00 00
04 00 00 00 07 00 00 00 0E 00 00 00 05 00 00 00
0B 00 00 00 08 00 00 00 0F 00 00 00 0D 00 00 00
```

一种可能答案为: $(array_3513[1] + array_3513[2]) * 3 = 48$

密码: 121212

3.6 阶段 6 的破解与分析

密码如下: 2 5 3 6 4 1

破解过程:

分析函数 phase_6 汇编代码:

首先

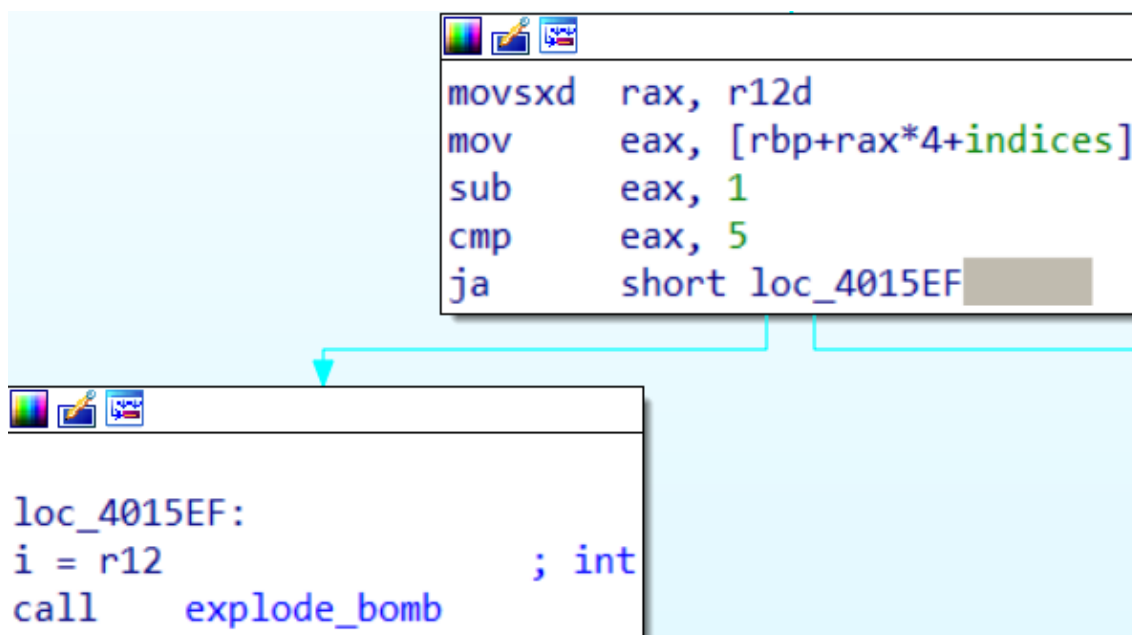
```
; void __fastcall phase_6(char *input)
public phase_6
phase_6 proc near

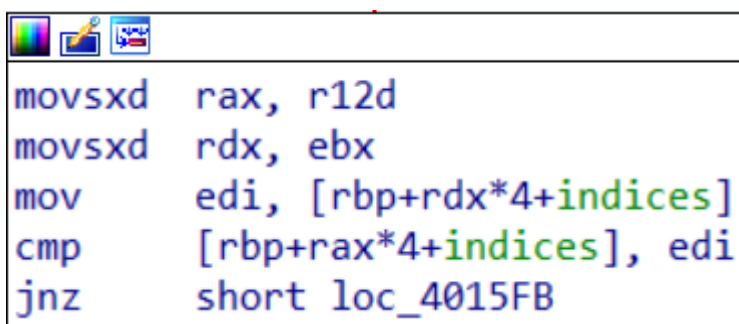
pointers= qword ptr -70h
indices= dword ptr -40h

input = rdi                ; char *
; __unwind {
push    rbp
mov     rbp, rsp
push    r13
push    r12
push    rbx
sub     rsp, 58h
lea     rsi, [rbp+indices] ; numbers
call    read_six_numbers
mov     r12d, 0
jmp     short loc_401618
```

函数读入 6 个数字；

然后





```
movsxd    rax, r12d
movsxd    rdx, ebx
mov       edi, [rbp+rdx*4+indices]
cmp       [rbp+rax*4+indices], edi
jnz       short loc_4015FB
```

原 c 代码应等价于：

```
for ( i = 0; i <= 5; ++i )
{
    if ( (indices[i] - 1) > 5 )
        explode_bomb();
    for ( j = i + 1; j <= 5; ++j )
    {
        if ( indices[i] == indices[j] )
            explode_bomb();
    }
}
```

6 个数字在 1~6 之间，且不能重复；

接着

```

40163E loc_40163E:                                ; CODE XREF: phase_6+7B↓j
40163E p = rdx                                    ; listNode *
40163E i = rsi                                    ; int
40163E j = rax                                    ; int
40163E      mov     p, [p+8]
401642      add     eax, 1
401645
401645 loc_401645:                                ; CODE XREF: phase_6+94↓j
401645      movsxd  rcx, esi
401648      cmp     [rbp+rcx*4+indices], eax
40164C      jg      short loc_40163E
40164E      mov     [rbp+rcx*8+pointers], p
401653      add     esi, 1
401656
401656 loc_401656:                                ; CODE XREF: phase_6+6B↑j
401656      cmp     esi, 5
401659      jg      short loc_401667
40165B      mov     eax, 1
401660      mov     edx, offset node1
401665      jmp     short loc_401645
401667 ; -----
401667
401667 loc_401667:                                ; CODE XREF: phase_6+88↑j
401667      mov     rbx, [rbp+pointers]
40166B p = rbx                                    ; listNode *
40166B      mov     rcx, p
40166E      mov     eax, 1
401673      jmp     short loc_401687
401675

```

原 c 代码应等价于:

```

for ( k = 0; k <= 5; ++k )
{
    v5 = 1;
    v4 = &node1;
    while ( indices[k] > v5 )
    {
        v4 = v4->next;
        ++v5;
    }
    pointers[k] = v4;
}

```

```

}

```

```

v6 = pointers[0];

```

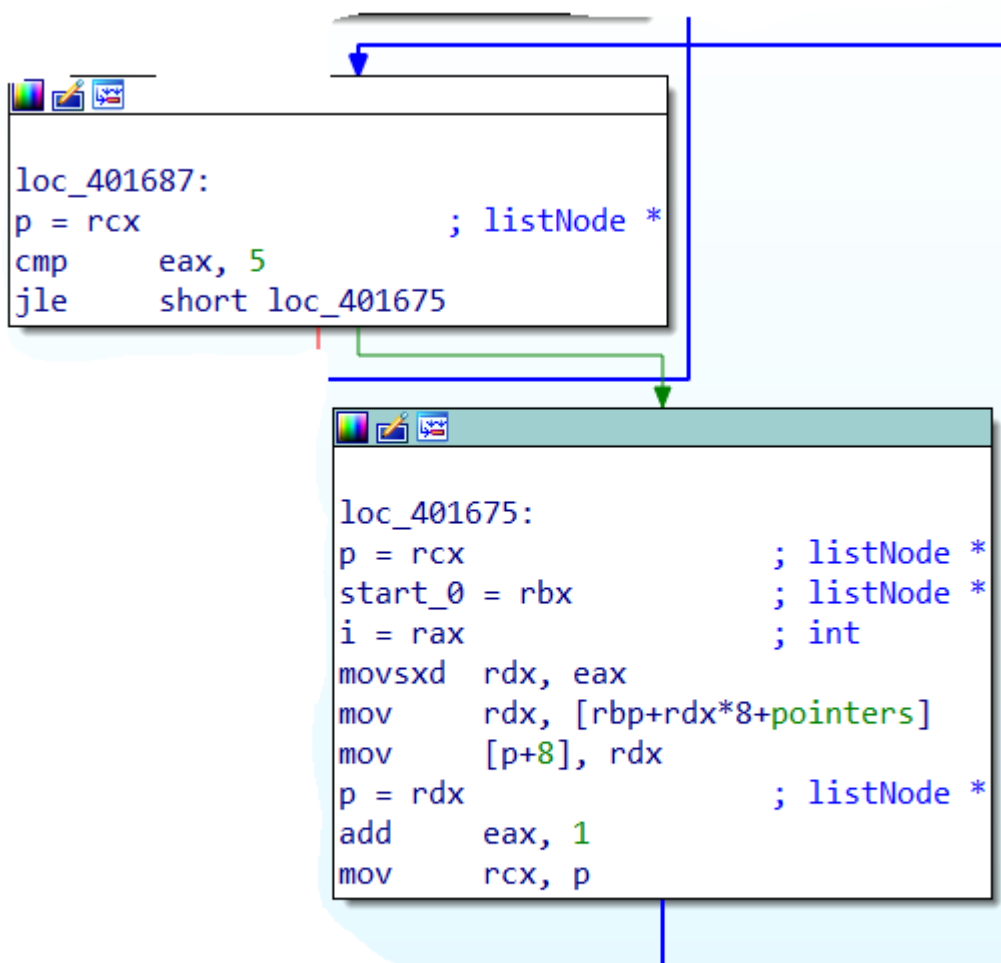
```

v7 = pointers[0];

```

使得 `pointers[k] = &node[indices[k]];`

然后



原 c 代码应等价于:

```

for ( l = 1; l <= 5; ++l )

```

```

{

```

```

    v9 = pointers[l];

```

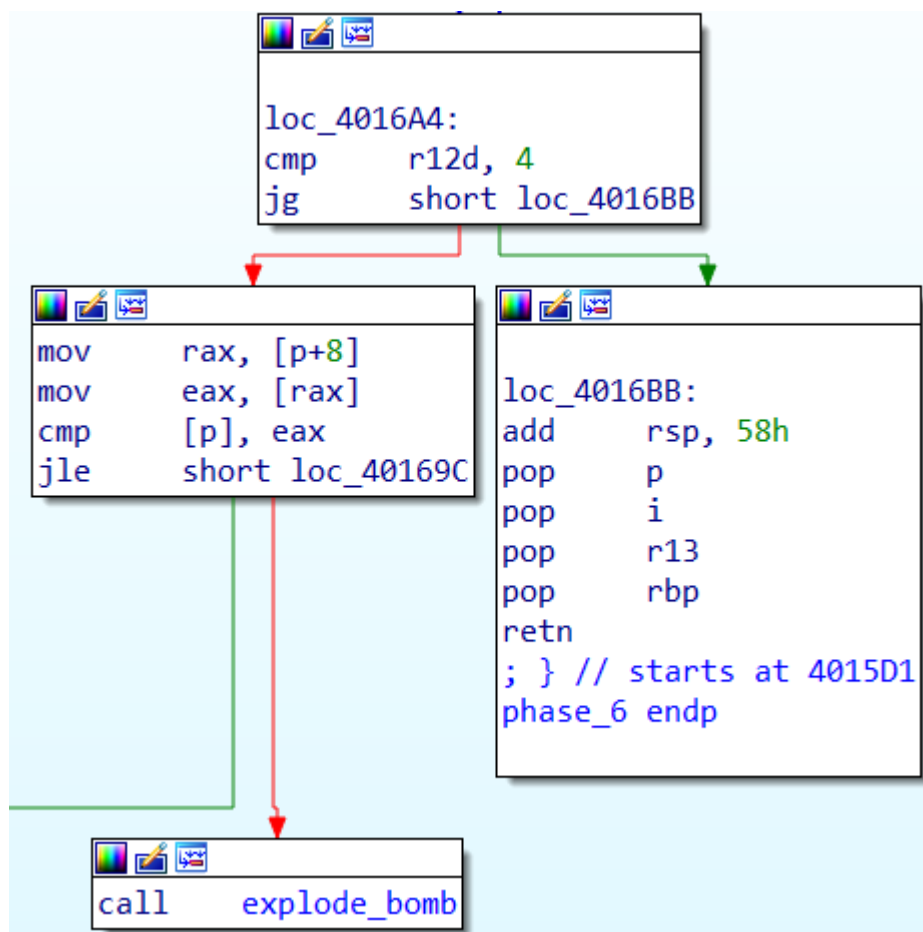
```
v7->next = v9;
```

```
v7 = v9;
```

```
}
```

按照输入的 6 个数字的值将相应的结点连接起来形成链表。

最后



原 c 代码应等价于：

```

for ( m = 0; m <= 4; ++m )
{
    if ( v6->value > v6->next->value )
        explode_bomb();
    v6 = v6->next;
}
  
```

链表结点的值应是单调不减的，否则爆炸；

查看结点：

```
node1      listNode <377h, 1, offset node2>
                                           ; DATA XREF: phase_6+8F↑o
           public node2
; listNode node2
node2      listNode <1D2h, 2, offset node3>
                                           ; DATA XREF: .data:node1↑o
           public node3
; listNode node3
node3      listNode <309h, 3, offset node4>
                                           ; DATA XREF: .data:node2↑o
           public node4
; listNode node4
node4      listNode <360h, 4, offset node5>
                                           ; DATA XREF: .data:node3↑o
           public node5
; listNode node5
node5      listNode <1E6h, 5, offset node6>
                                           ; DATA XREF: .data:node4↑o
           public node6
; listNode node6
node6      listNode <31Fh, 6, 0> ; DATA XREF: .data:node5↑o
           public bomb_id
```

按照从小到大的顺序，结点依次应是：2 5 3 6 4 1

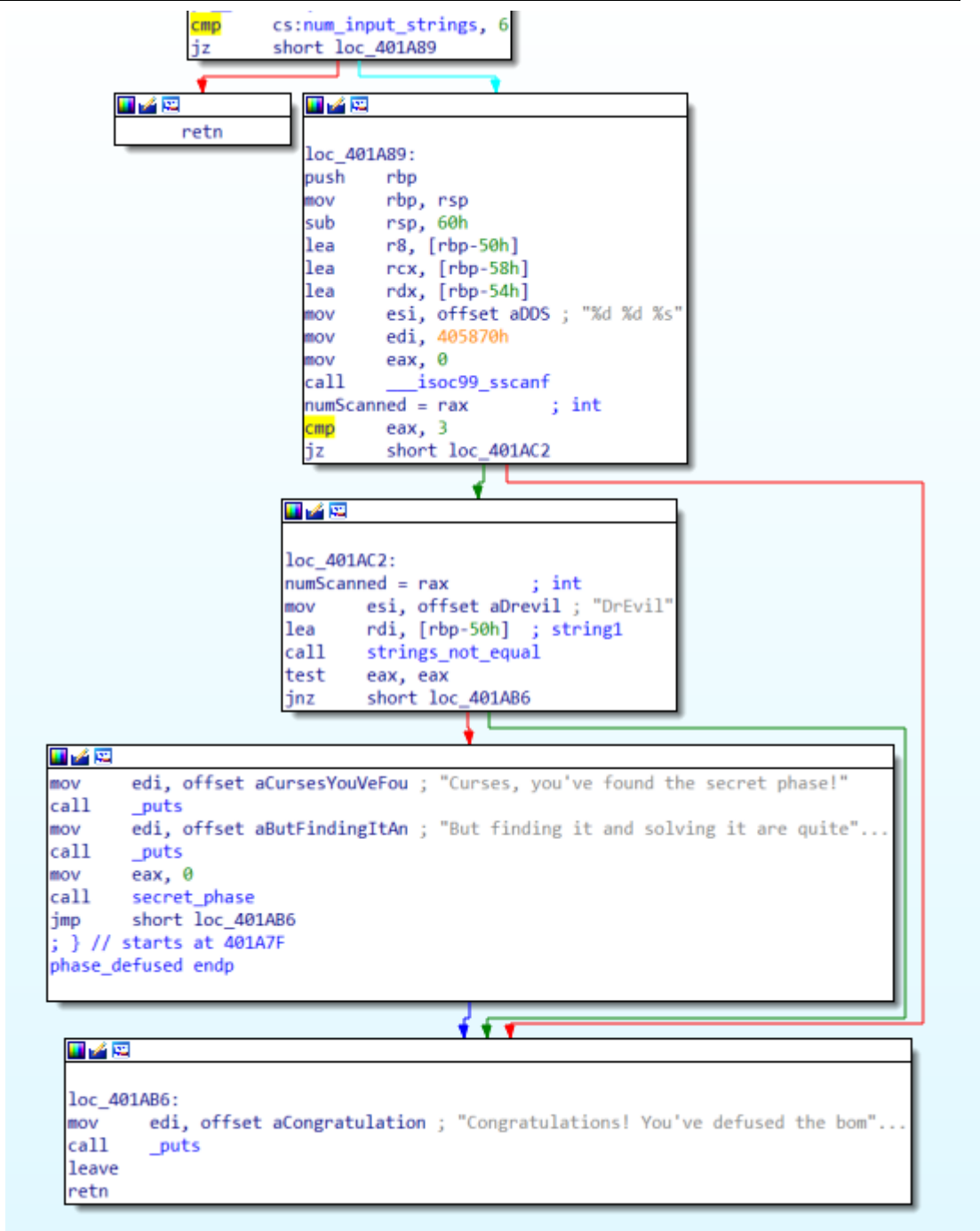
输入字符串：2 5 3 6 4 1

3.7 阶段 7 的破解与分析(隐藏阶段)

密码如下：第四个字符串：“8 35 DrEvil”，第七个字符串为：7

破解过程：

首先查看函数 phase_defused 汇编代码：



原 c 代码应等价于：

```
if ( num_input_strings == 6 )
```

```
{
```

```
    if ( __isoc99_sscanf("%d %d %s") == 3 && !strings_not_equal(passphrase,
"DrEvil") )
```



```

{
    puts("Curses, you've found the secret phase!");
    puts("But finding it and solving it are quite different...");
    secret_phase();
}

puts("Congratulations! You've defused the bomb!");
}

```

在破解 6 个密码后，读取的字符串符合要求 ("%d %d DrEvil") 才会进入秘密关卡。请看下图相应的汇编：

```

push    rbp
mov     rbp, rsp
sub     rsp, 60h
lea     r8, [rbp-50h]
lea     rcx, [rbp-58h]
lea     rdx, [rbp-54h]
mov     esi, offset aDDS ; "%d %d %s"
mov     edi, 405870h
mov     eax, 0
call    __isoc99_sscanf
x                               ; int
cmp     eax, 3
jz      short loc_401AC2

```

可以看到，__isoc99_sscanf@plt 接受了一个格式化字符串 "%d %d %s" 和一个输入字符串，输入字符串的地址是 0x405870，gdb 调试运行到此处可打印其值。

```

(gdb) x/s 0x405870
0x405870 <input_strings+240>:  "8 35"

```

"8 35" 是第四个字符串的值，说明程序读取第四个字符串作为输入字符串。

然后

函数 secret_phase 反汇编后代码大致如下：

```
secret_phase()
```

```

{
    char *v;

    int val;

    v = read_line();                //读入字符串

    val = strtol(v, NULL, 10);      //将字符串转换成 10 进制

    if ( (unsigned int)(val - 1) > 0x3E8 ) //1~0x3E9(1001)之间

        explode_bomb();

    if ( fun7(&n1, val) != 4 )        //应让 fun7(&n1, val) 返回 4

        explode_bomb();

    puts("Wow! You've defused the secret stage!");

    phase_defused();
}

```

再看函数 fun7:

```

fun7(treeNode *node, int val)

{
    if ( !node ) return -1;

    //node 为空, 返回-1

    if ( node->value > val ) return 2 * fun7(node->left, val);

    // node 的值大于 val, 返回 2 * fun7(node->left, val)

    if ( node->value == val ) return 0;

    // node 的值等于 val, 返回 0

    else return 2 * fun7(node->right, val) + 1;

    // node 的值小于 val, 返回 2 * fun7(node->right, val) + 1

}

```

要使函数返回 4, 唯一的递归过程是: $2 * (2 * (2 * 0 + 1))$

结合二叉树的结构可知：

第一次调用：n1->value(24h) > val，n1->left 为 n21；

第二次调用：n21->value(8) > val，n21->left 为 n31；

第三次调用：n31->value(6) < val，n31->right 为 n42；

第四次调用：n42->value(7) = val，返回 0。

val = 7

密码为：7

第 4 章 总结

4.1 请总结本次实验的收获

1. 训练了一定的反汇编能力
2. 对字符串，循环、分支语句以及数组、链表等数据结构的汇编实现加深了理解

4.2 请给出对本次实验内容的建议

注：本章为酌情加分项。

参考文献

为完成本次实验你翻阅的书籍与网站等

- [1] 林来兴. 空间控制技术[M]. 北京: 中国宇航出版社, 1992: 25-42.
- [2] 辛希孟. 信息技术与信息服务国际研讨会论文集: A 集[C]. 北京: 中国科学出版社, 1999.
- [3] 赵耀东. 新时代的工业工程师[M/OL]. 台北: 天下文化出版社, 1998 [1998-09-26]. <http://www.ie.nthu.edu.tw/info/ie.newie.htm> (Big5) .
- [4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨: 哈尔滨工业大学, 1992: 8-13.
- [5] KANAMORI H. Shaking Without Quaking[J]. Science, 1998, 279 (5359): 2063-2064.
- [6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science , 1998 , 281 : 331-332[1998-09-23]. <http://www.sciencemag.org/cgi/collection/anatmorp>.