

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 计算机科学与技术

学 号 1160300901

班 级 1603009

学 生 孙月晴

指 导 教 师 吴锐

实 验 地 点 G712

实 验 日 期 2017-10-24

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.....	- 9 -
2.1 UBUNTU 下 CODEBLOCKS 反汇编（10 分）	- 9 -
2.2 UBUNTU 下 EDB 运行环境建立（10 分）	- 11 -
第 3 章 各阶段炸弹破解与分析.....	- 13 -
3.1 阶段 1 的破解与分析.....	- 13 -
3.2 阶段 2 的破解与分析.....	- 14 -
3.3 阶段 3 的破解与分析.....	- 15 -
3.4 阶段 4 的破解与分析.....	- 18 -
3.5 阶段 5 的破解与分析.....	- 20 -
3.6 阶段 6 的破解与分析.....	- 21 -
3.7 阶段 7 的破解与分析(隐藏阶段).....	- 26 -
第 4 章 总结.....	- 30 -
4.1 请总结本次实验的收获.....	- 30 -
4.2 请给出对本次实验内容的建议.....	- 30 -
参考文献.....	- 31 -

第 1 章 实验基本信息

1.1 实验目的

- 1.熟练掌握计算机系统的 ISA 指令系统与寻址方式
- 2.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
- 3.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

1.2.2 软件环境

Windows10 64 位; Vmware 12; Ubuntu 16.04 LTS 64 位;

1.2.3 开发工具

Visual Studio 2010 64 位; GDB/OBJDUMP; KDD 等

1.3 实验预习

Sample.c 程序原代码:

```
#include <stdio.h>
#include <conio.h>
#include <assert.h>
#include <mem.h>
#include <malloc.h>
```

```
//指针、结构体
typedef struct stu {
    int i;
    struct stu *next;
} ST;
```

```
//递归
```

```
int fib2(int n) {
    if (n <= 0) {
        return 0;
    } else if (n <= 2) {
        return 1;    //递归终止条件
    } else {
        return fib2(n - 1) + fib2(n - 2);    //递归
    }
}

//字符串比较
void sc(char *s, char *t) {
    while (*s++ == *t++ && strlen(s)); //strlen()的功能在于限定比较的次数
    if (*s > *t)
        printf("大于\n");
    else if (*s < *t)
        printf("小于\n");
    else printf("等于\n");
}

ST *InitPoly(void) {
    ST *pHead;
    if ((pHead = (ST *) malloc(sizeof(ST))) == NULL)
        exit(-1);
    pHead->next = NULL;

    return pHead;
}

void FreeList(ST *pHead) {
    ST *pTemp1, *pTemp2;
    pTemp1 = pHead->next;
    while (pTemp1 != NULL) {
        pTemp2 = pTemp1->next;
        free(pTemp1);
        pTemp1 = pTemp2;
    }

    pHead->next = NULL;
}

void DesList(ST *pHead) {
    FreeList(pHead);
    free(pHead);
}

void Input(ST *pHead) {
```

```
int expn;
int j;
ST *pTail, *pNew;
pTail = pHead;
int a[5] = {4, 3, 2, 1, 0};
for (j = 0; j < 5; j++) {

    if (a[j] != 0) {
        if ((pNew = (ST *) malloc(sizeof(ST))) == NULL)
            exit(-1);
        pNew->i = a[j];
        pTail->next = pNew;
        pTail = pNew;
    } else {
        if ((pNew = (ST *) malloc(sizeof(ST))) == NULL)
            exit(-1);
        pNew->i = a[j];
        pTail->next = pNew;
        pTail = pNew;
        break;
    }
}
pTail->next = NULL;
}

void Output(ST *phead) {
    ST *pTemp;
    int a;
    pTemp = phead->next;
    for (a = 0; a < 5; a++) {
        printf("%d ", pTemp->i);
        pTemp = pTemp->next;
    }
    free(pTemp);
    printf("\n");
}

int main() {
    char s[10] = {'a', 'b', 'c'};
    char t[10] = {'a', 'b', 'c'};
    int j;
    int a = 2;
    int *p = &a;
    ST *head;
    sc(s, t);
    for (j = 0; j <= 3; j++) {
        printf("%d ", j);
    }
}
```

```

printf("\n");
int num;
float C = 90;
num = (int) (C / 10);
switch (num) {
    case 10:
    case 9:
        printf("等级为优秀! \n");
        break;
    case 8:
        printf("等级为良好! \n");
        break;
    case 7:
    case 6:
        printf("等级为合格。 \n");
        break;
    default:
        printf("等级为不合格。 \n");
        break;
}
printf("%d\n", fib2(6));
head = InitPoly();
Input(head);
Output(head);
DesList(head);

return 0;
}

```

汇编语言：

```

0x4016c9  push    %ebp
0x4016ca  mov     %esp,%ebp
0x4016cc  and     $0xffffffff0,%esp
0x4016cf  sub     $0x50,%esp
0x4016d2  call    0x401d40 <__main>
0x4016d7  movl    $0x0,0x32(%esp)
0x4016df  movl    $0x0,0x36(%esp)
0x4016e7  movw    $0x0,0x3a(%esp)
0x4016ee  movb    $0x61,0x32(%esp)
0x4016f3  movb    $0x62,0x33(%esp)
0x4016f8  movb    $0x63,0x34(%esp)
0x4016fd  movl    $0x0,0x28(%esp)
0x401705  movl    $0x0,0x2c(%esp)

```

0x40170d	movw	\$0x0,0x30(%esp)
0x401714	movb	\$0x61,0x28(%esp)
0x401719	movb	\$0x62,0x29(%esp)
0x40171e	movb	\$0x63,0x2a(%esp)
0x401723	movl	\$0x2,0x24(%esp)
0x40172b	lea	0x24(%esp),%eax
0x40172f	mov	%eax,0x48(%esp)
0x401733	lea	0x28(%esp),%eax
0x401737	mov	%eax,0x4(%esp)
0x40173b	lea	0x32(%esp),%eax
0x40173f	mov	%eax,(%esp)
0x401742	call	0x401487 <sc>
0x401747	movl	\$0x0,0x4c(%esp)
0x40174f	jmp	0x40176a <main+161>
0x401751	mov	0x4c(%esp),%eax
0x401755	mov	%eax,0x4(%esp)
0x401759	movl	\$0x40b073,(%esp)
0x401760	call	0x408fa0 <printf>
0x401765	addl	\$0x1,0x4c(%esp)
0x40176a	cmpl	\$0x3,0x4c(%esp)
0x40176f	jle	0x401751 <main+136>
0x401771	movl	\$0xa,(%esp)
0x401778	call	0x408f98 <putchar>
0x40177d	flds	0x40b0c8
0x401783	fstps	0x44(%esp)
0x401787	flds	0x44(%esp)
0x40178b	flds	0x40b0cc
0x401791	fdivrp	%st,%st(1)
0x401793	fnstcw	0x1e(%esp)
0x401797	movzwl	0x1e(%esp),%eax
0x40179c	mov	\$0xc,%ah
0x40179e	mov	%ax,0x1c(%esp)
0x4017a3	fldcw	0x1c(%esp)
0x4017a7	fistpl	0x40(%esp)
0x4017ab	fldcw	0x1e(%esp)
0x4017af	mov	0x40(%esp),%eax
0x4017b3	sub	\$0x6,%eax
0x4017b6	cmp	\$0x4,%eax

```

0x4017b9  ja      0x4017ee <main+293>
0x4017bb  mov     0x40b0b4(,%eax,4),%eax
0x4017c2  jmp     *%eax
0x4017c4  movl    $0x40b077,(%esp)
0x4017cb  call    0x408f90 <puts>
0x4017d0  jmp     0x4017fb <main+306>
0x4017d2  movl    $0x40b084,(%esp)
0x4017d9  call    0x408f90 <puts>
0x4017de  jmp     0x4017fb <main+306>
0x4017e0  movl    $0x40b091,(%esp)
0x4017e7  call    0x408f90 <puts>
0x4017ec  jmp     0x4017fb <main+306>
0x4017ee  movl    $0x40b09e,(%esp)
0x4017f5  call    0x408f90 <puts>
0x4017fa  nop
0x4017fb  movl    $0x6,(%esp)
0x401802  call    0x401440 <fib2>
0x401807  mov     %eax,0x4(%esp)
0x40180b  movl    $0x40b0ad,(%esp)
0x401812  call    0x408fa0 <printf>
0x401817  call    0x4014ff <InitPoly>
0x40181c  mov     %eax,0x3c(%esp)
0x401820  mov     0x3c(%esp),%eax
0x401824  mov     %eax,(%esp)
0x401827  call    0x401592 <Input>
0x40182c  mov     0x3c(%esp),%eax
0x401830  mov     %eax,(%esp)
0x401833  call    0x40166f <Output>
0x401838  mov     0x3c(%esp),%eax
0x40183c  mov     %eax,(%esp)
0x40183f  call    0x401573 <DesList>
0x401844  mov     $0x0,%eax
0x401849  leave
0x40184a  ret

```

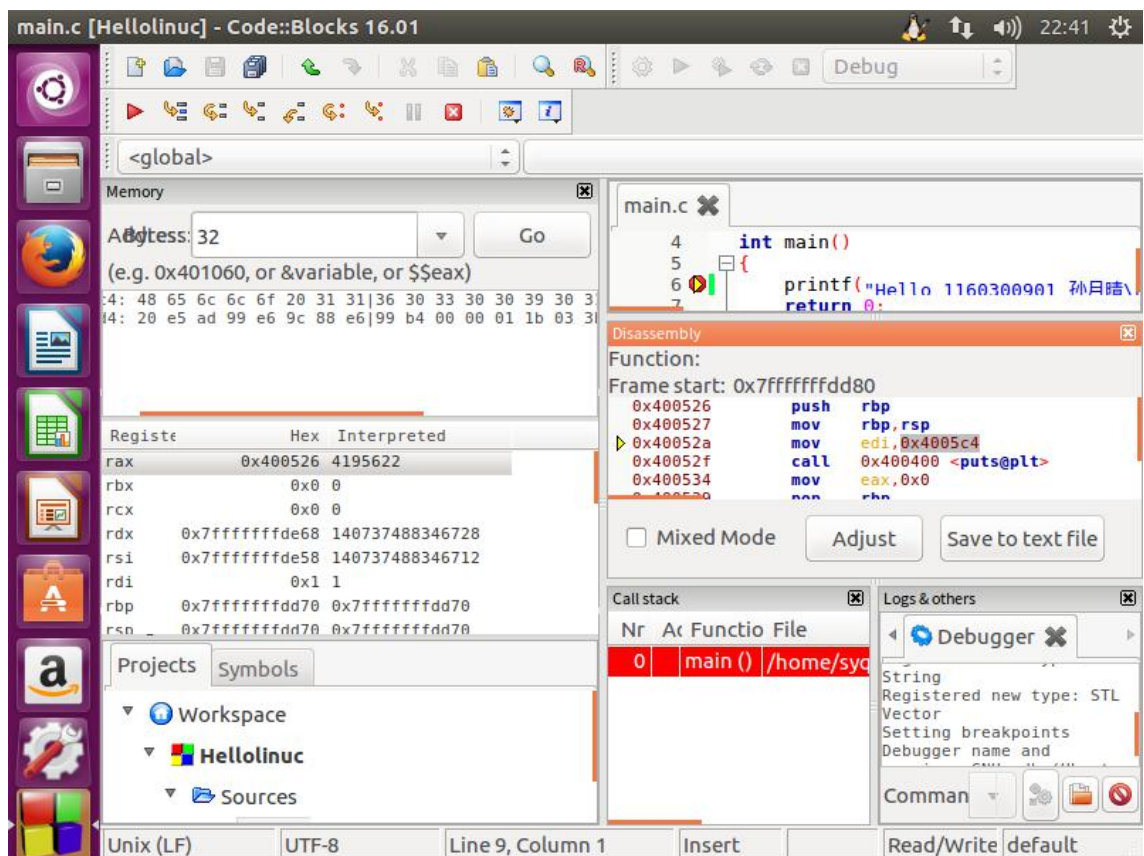

第 2 章 实验环境建立

2.1 Ubuntu 下 CodeBlocks 反汇编 (10 分)

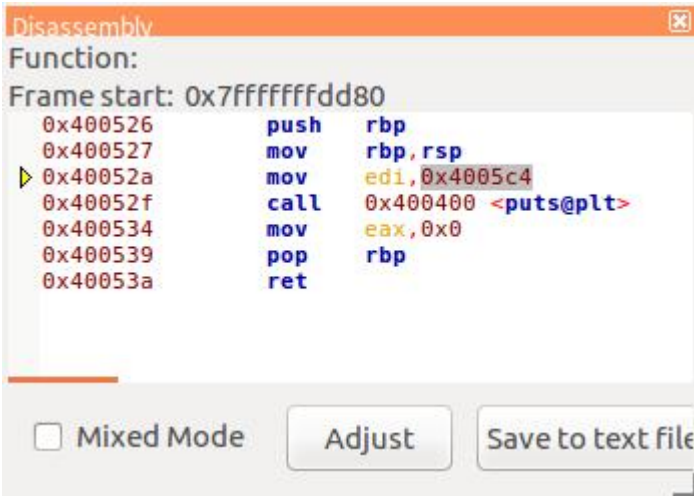
CodeBlocks 运行 hellolinux.c。反汇编查看 printf 函数的实现。

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。

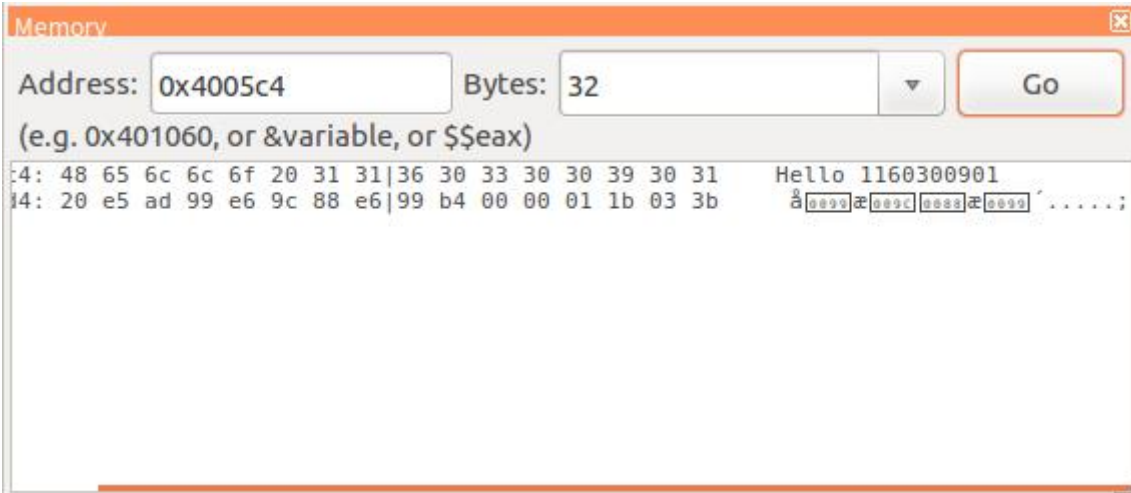
总窗口：



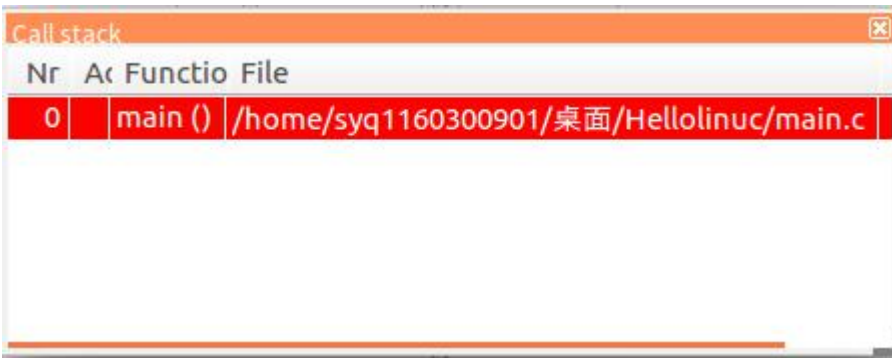
ASM:



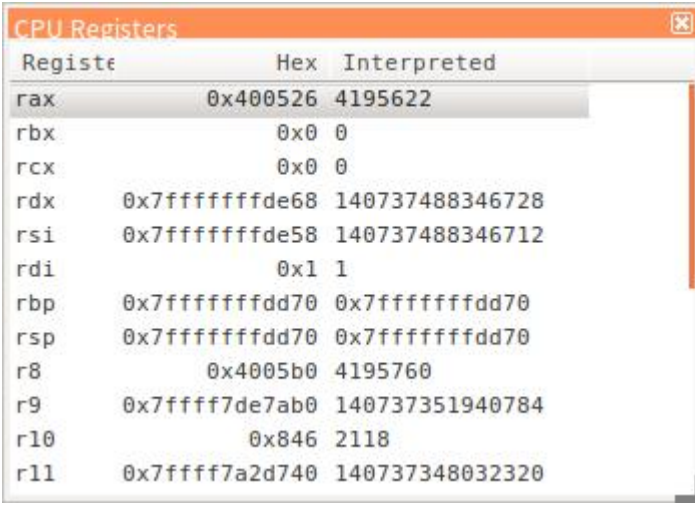
内存:



堆栈:



寄存器:



Register	Hex	Interpreted
rax	0x400526	4195622
rbx	0x0	0
rcx	0x0	0
rdx	0x7fffffffde68	140737488346728
rsi	0x7fffffffde58	140737488346712
rdi	0x1	1
rbp	0x7fffffffdd70	0x7fffffffdd70
rsp	0x7fffffffdd70	0x7fffffffdd70
r8	0x4005b0	4195760
r9	0x7ffff7de7ab0	140737351940784
r10	0x846	2118
r11	0x7ffff7a2d740	140737348032320

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

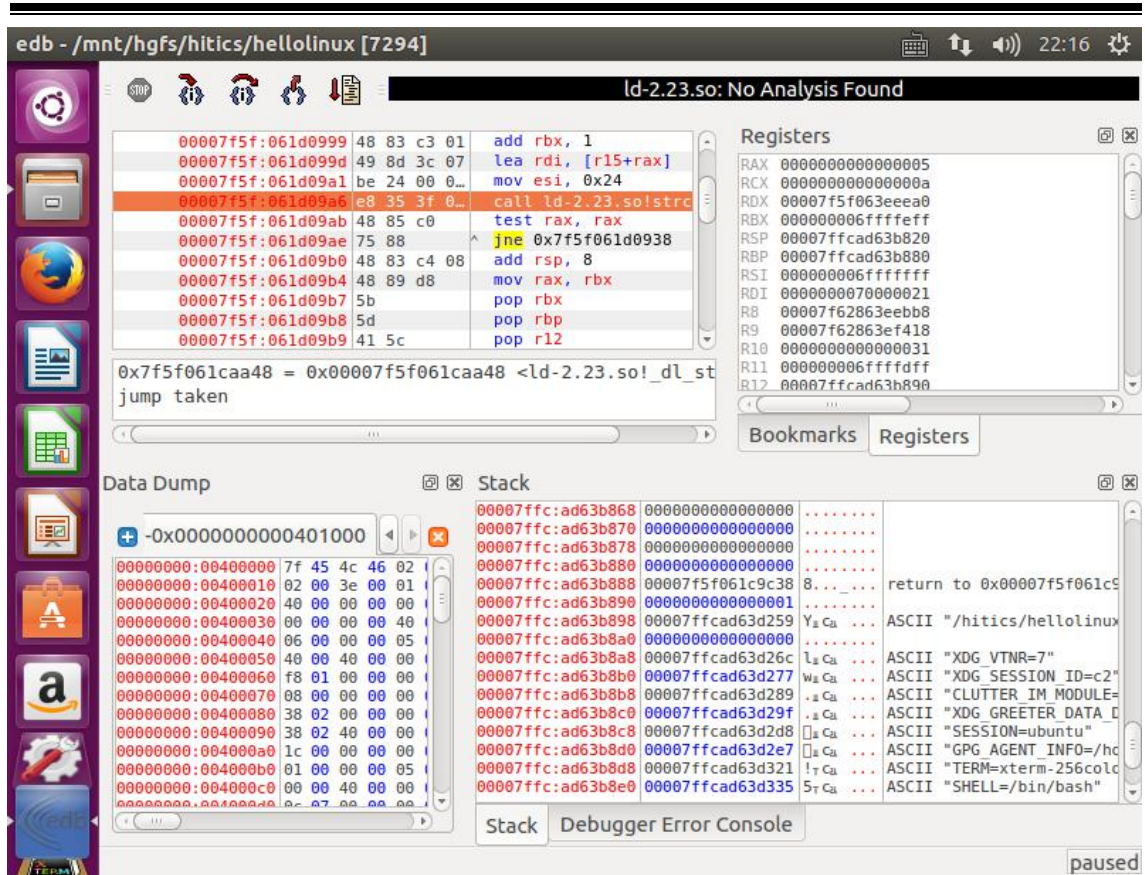


图 2-2 Ubuntu 下 EDB 截图

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：Public speaking is very easy.

```
Public speaking is very easy.
Phase 1 defused. How about the next one?
```

破解过程：

```
0000000000400e8d <phase_1>:
400e8d: 48 83 ec 08          sub    $0x8,%rsp
400e91: be 8c 23 40 00      mov    $0x40238c,%esi
400e96: e8 5e 04 00 00      callq 4012f9 <strings_not_equal>
400e9b: 85 c0               test   %eax,%eax
400e9d: 74 05              je     400ea4 <phase_1+0x17>
400e9f: e8 54 05 00 00      callq 4013f8 <explode_bomb>
400ea4: 48 83 c4 08          add    $0x8,%rsp
400ea8: c3                 retq

400de8: e8 6c 06 00 00      callq 401459 <read_line>
400ded: 48 89 c7            mov    %rax,%rdi
400df0: e8 98 00 00 00      callq 400e8d <phase_1>
400df5: e8 85 07 00 00      callq 40157f <phase_defused>
```

```
(gdb) x/s 0x40238c
0x40238c: "Public speaking is very easy."
```

```
syq1160300901@syq1160300901-virtual-machine:~/bomb265$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
```

1. 递减栈指针
2. 把地址值 0x40238c 给寄存器%esi（第二个参数）
3. 用 gdb 调试查看 0x40238c 的值，得到字符串 Public speaking is very easy. 即为答案。

3.2 阶段 2 的破解与分析

密码如下：0 1 1 2 5

0 1 1 2 3 5
That's number 2. Keep going!

破解过程：

```

400eaf: 64 48 8b 04 25 28 00  mov    %fs:0x28,%rax
400eb6: 00 00                  mov    %rax,0x18(%rsp)
400eb8: 48 89 44 24 18        mov    %eax,%eax
400ebd: 31 c0                  xor     %eax,%eax
400ebf: 48 89 e6              mov     %rsi,%rsi
400ec2: e8 53 05 00 00        callq  40141a <read_six_numbers>
400ec7: 83 3c 24 00           cmpl   $0x0,(%rsp)
400ecb: 75 07                 jne     400ed4 <phase_2+0x2b>
400ecd: 83 7c 24 04 01        cmpl   $0x1,0x4(%rsp)
400ed2: 74 05                 je      400ed9 <phase_2+0x30>
400ed4: e8 1f 05 00 00        callq  4013f8 <explode_bomb>
400ed9: 48 89 e3              mov     %rsi,%rsi
400edc: 48 8d 6c 24 10        lea     0x10(%rsp),%rbp
400ee1: 8b 43 04              mov     0x4(%rbx),%eax
400ee4: 03 03                 add     (%rbx),%eax
400ee6: 39 43 08              cmp     %eax,0x8(%rbx)
400ee9: 74 05                 je      400ef0 <phase_2+0x47>
400eeb: e8 08 05 00 00        callq  4013f8 <explode_bomb>
400ef0: 48 83 c3 04           add     $0x4,%rbx
400ef4: 48 39 eb              cmp     %rbp,%rbx
400ef7: 75 e8                 jne     400ee1 <phase_2+0x38>
400ef9: 48 8b 44 24 18        mov     0x18(%rsp),%rax
400efe: 64 48 33 04 25 28 00  xor     %fs:0x28,%rax
400f05: 00 00                  je      400f0e <phase_2+0x65>
400f07: 74 05                 callq  400b00 <__stack_chk_fail@plt>
400f09: e8 f2 fb ff ff        add     $0x28,%rsp
400f0e: 48 83 c4 28           pop     %rbx
400f12: 5b                    pop     %rbp
400f13: 5d

```

这题最关键的就是注意<read_six_numbers>这个函数，依据函数名字可以猜测这函数的密码应该是 6 个数字。紧接着通过分析

1. 把 0 与 (%rsp) 比较，如果不相等引爆，说明 (%rsp) = 0
2. 把 0x1 与 0x4(%rsp) 比较，如果相等跳转到 400ed9, 否则引爆，说明 0x4(%rsp)=1
3. 把 %rsp 的值赋给 %rbx
4. 把 0x10 (%rsp) 加载到 %rbp
5. 把 0x4 (%rbx) 的值赋给 %eax,(也就是 0x4(%rsp)，所以为 1,)
6. 把 (%rbx)+%eax 赋给 %eax,((%rbx)也就是 (%rsp) 即 0, %eax 为 1，所以

为 0+1, 此时%eax 为 1)

7. 比较%eax 的值和 0x8 (%rbx) 的值即比较 1 与 0x8(%rbx),如果相等跳转到 400ef0,否则引爆,说明 0x8 (%rbx) 为 1

8. %rbx=%rbx+0x4,%rbx 上移一位(存的是一个 int 类型的数组,4 个字节,32 位)

9. 比较%rbp 和%rbx,如果不相等也就是%rbx 与 0x10 (%rsp) 中的地址值不同,跳到第 5 步,继续循环。

可以得到第一个数是 0 第二个数是 1,当%rbp 与%rbx 指向同一位置时,循环结束

然后就是在上图画具体循环,找到逻辑关系

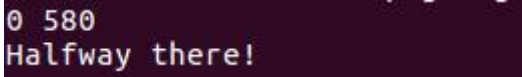
$a[0]=0$ $a[1]=1$ $a[2]=a[0]+a[1]=1$ $a[3]=a[2]+a[1]=2$ $a[4]=3$ $a[5]=5$

即答案为 0 1 1 2 3 5

3.3 阶段 3 的破解与分析

密码如下: 0 580 / 1 527 / 2 854 / 3 565 / 4 605 / 5 881 / 7 824

/ 8 888



```
0 580
Halfway there!
```

破解过程:

```

0000000000400f15 <phase_3>:
400f15: 48 83 ec 18      sub    $0x18,%rsp
400f19: 64 48 8b 04 25 28 00 mov    %fs:0x28,%rax
400f20: 00 00
400f22: 48 89 44 24 08      mov    %rax,0x8(%rsp)
400f27: 31 c0            xor    %eax,%eax
400f29: 48 8d 4c 24 04      lea    0x4(%rsp),%rcx
400f2e: 48 89 e2          mov    %rsp,%rdx
400f31: be 77 25 40 00      mov    $0x402577,%esi
400f36: e8 75 fc ff ff      callq 400bb0 <__isoc99_sscanf@plt>
400f3b: 83 f8 01          cmp    $0x1,%eax
400f3e: 7f 05            jg     400f45 <phase_3+0x30>
400f40: e8 b3 04 00 00      callq 4013f8 <explode_bomb>
400f45: 83 3c 24 07        cmpl   $0x7,(%rsp)
400f49: 77 3b            ja     400f86 <phase_3+0x71>
400f4b: 8b 04 24          mov    (%rsp),%eax
400f4e: ff 24 c5 c0 23 40 00 jmpq    *0x4023c0(,%rax,8)
400f55: b8 0f 02 00 00      mov    $0x20f,%eax
400f5a: eb 3b            jmp    400f97 <phase_3+0x82>
400f5c: b8 56 03 00 00      mov    $0x356,%eax
400f61: eb 34            jmp    400f97 <phase_3+0x82>
400f63: b8 35 02 00 00      mov    $0x235,%eax
400f68: eb 2d            jmp    400f97 <phase_3+0x82>
400f6a: b8 5d 02 00 00      mov    $0x25d,%eax
400f6f: eb 26            jmp    400f97 <phase_3+0x82>
400f71: b8 71 03 00 00      mov    $0x371,%eax
400f76: eb 1f            jmp    400f97 <phase_3+0x82>
400f78: b8 38 03 00 00      mov    $0x338,%eax

400f7d: eb 18            jmp    400f97 <phase_3+0x82>
400f7f: b8 78 03 00 00      mov    $0x378,%eax
400f84: eb 11            jmp    400f97 <phase_3+0x82>
400f86: e8 6d 04 00 00      callq 4013f8 <explode_bomb>
400f8b: b8 00 00 00 00      mov    $0x0,%eax
400f90: eb 05            jmp    400f97 <phase_3+0x82>
400f92: b8 44 02 00 00      mov    $0x244,%eax
400f97: 3b 44 24 04        cmp    0x4(%rsp),%eax
400f9b: 74 05            je     400fa2 <phase_3+0x8d>
400f9d: e8 56 04 00 00      callq 4013f8 <explode_bomb>
400fa2: 48 8b 44 24 08      mov    0x8(%rsp),%rax
400fa7: 64 48 33 04 25 28 00 xor    %fs:0x28,%rax
400fae: 00 00
400fb0: 74 05            je     400fb7 <phase_3+0xa2>
400fb2: e8 49 fb ff ff      callq 400b00 <__stack_chk_fail@plt>
400fb7: 48 83 c4 18        add    $0x18,%rsp
400fbb: c3              retq

0000000000400f15 <phase_3>:
400f15: 48 83 ec 18      sub    $0x18,%rsp //rsp-24
400f19: 64 48 8b 04 25 28 00 mov    %fs:0x28,%rax
400f20: 00 00
400f22: 48 89 44 24 08      mov    %rax,0x8(%rsp)
400f27: 31 c0            xor    %eax,%eax
400f29: 48 8d 4c 24 04      lea    0x4(%rsp),%rcx
400f2e: 48 89 e2          mov    %rsp,%rdx

400f31: be 77 25 40 00      mov    $0x402577,%esi
400f36: e8 75 fc ff ff      callq 400bb0 <__isoc99_sscanf@plt> //输入两个数

400f3b: 83 f8 01          cmp    $0x1,%eax //所输入的值的个数大于 1
400f3e: 7f 05            jg     400f45 <phase_3+0x30>

```



```

400f40: e8 b3 04 00 00      callq 4013f8 <explode_bomb>
//说明 eax (此时 sscanf 函数的返回值) 大于 1, 这个函数的返回值是输入元素的个数

400f45: 83 3c 24 07         cmpl $0x7, (%rsp)
400f49: 77 3b               ja 400f86 <phase_3+0x71>
//如果 rsp 大于 7, 则跳到<phase_3+0x71> (炸弹) 说明 rsp 小于等于 7.

400f4b: 8b 04 24            mov (%rsp), %eax //rsp 的值赋给 eax
400f4e: ff 24 c5 c0 23 40 00 jmpq *0x4023c0(, %rax, 8)
//这是典型的 switch 跳转语句, 即跳转到以地址*0x4023c0 为基址的跳转表中, 读出基值为
+125
400f55: b8 0f 02 00 00      mov $0x20f, %eax //527
400f5a: eb 3b               jmp 400f97 <phase_3+0x82>
400f5c: b8 56 03 00 00      mov $0x356, %eax //854
400f61: eb 34               jmp 400f97 <phase_3+0x82>
400f63: b8 35 02 00 00      mov $0x235, %eax //565
400f68: eb 2d               jmp 400f97 <phase_3+0x82>
400f6a: b8 5d 02 00 00      mov $0x25d, %eax //605
400f6f: eb 26               jmp 400f97 <phase_3+0x82>
400f71: b8 71 03 00 00      mov $0x371, %eax //881
400f76: eb 1f               jmp 400f97 <phase_3+0x82>
400f78: b8 38 03 00 00      mov $0x338, %eax //824
400f7d: eb 18               jmp 400f97 <phase_3+0x82>
400f7f: b8 78 03 00 00      mov $0x378, %eax //888
400f84: eb 11               jmp 400f97 <phase_3+0x82>
400f86: e8 6d 04 00 00      callq 4013f8 <explode_bomb>
400f8b: b8 00 00 00 00      mov $0x0, %eax
400f90: eb 05               jmp 400f97 <phase_3+0x82>
400f92: b8 44 02 00 00      mov $0x244, %eax //580
//*0x4023c0 这个值, 就是上面这个地址, eax=580
400f97: 3b 44 24 04         cmp 0x4(%rsp), %eax
400f9b: 74 05               je 400fa2 <phase_3+0x8d>
400f9d: e8 56 04 00 00      callq 4013f8 <explode_bomb>
400fa2: 48 8b 44 24 08      mov 0x8(%rsp), %rax
400fa7: 64 48 33 04 25 28 00 xor %fs:0x28, %rax
400fae: 00 00
400fb0: 74 05               je 400fb7 <phase_3+0xa2>
400fb2: e8 49 fb ff ff      callq 400b00 <__stack_chk_fail@plt>
400fb7: 48 83 c4 18         add $0x18, %rsp
400fbb: c3                 retq

```

首先, 题目中给出最清晰的线索是那个地址, 我们看一下它的值:

```

(gdb) x/s 0x402577
0x402577: "%d %d"

```

是%d, %d, 这要输入两个整数值, 考虑栈帧结构, rsp+4 的值是参数 1, 存在 rcx 处, rsp 是参数 2, 存在 rdx 处, 要输入的就是这两个数。

```

(gdb) x/s *0x4023c0
0x400f92 <phase_3+125>: "\270D\002"

```

跳转到以地址*0x4023c0 为基址的跳转表中, 读出基值为+125

从后面的代码逻辑可以看出，要根据第一个参数的具体值跳到不同的地址，来执行命令。而且第一个参数要小于等于 7. 可以分别假设第一个参数等于 0 到 7，然后可以看出根据这个分支选择可以确定 eax 的大小，然后第二个参数等 eax 的值即可。

可以得出本阶段共 8 个答案，即

0 580 / 1 527 / 2 854 / 3 565 / 4 605 / 5 881 / 7 824 / 8 888

3.4 阶段 4 的破解与分析

密码如下：

3 10
So you got that one. Try this one.

破解过程：

```
0000000000400fbc <func4>:
400fbc: 53                push    %rbx
400fbd: 89 d0             mov     %edx,%eax    //%edx 为第三个参数
400fbf: 29 f0             sub     %esi,%eax    //%esi 为第二个参数
400fc1: 89 c3             mov     %eax,%ebx
400fc3: c1 eb 1f          shr     $0x1f,%ebx   //%ebx 逻辑右移 0x1f
400fc6: 01 d8             add     %ebx,%eax
400fc8: d1 f8             sar     %eax          // eax =
((edx-esi)>>5+(edx-esi))>>1 = (edx-esi)/2
400fca: 8d 1c 30          lea     (%rax,%rsi,1),%ebx //ebx =
0.5*(edx-esi)+rsi
400fcd: 39 fb             cmp     %edi,%ebx    //第一个输入数字与%ebx 比
较
400fcf: 7e 0c             jle     400fdd <func4+0x21> //小于或等于
400fd1: 8d 53 ff          lea     -0x1(%rbx),%edx //如果输入第一个参
数小于 ebx, 递归调用 edx = rbx -1
400fd4: e8 e3 ff ff ff    callq   400fbc <func4>
400fd9: 01 d8             add     %ebx,%eax
400fdb: eb 10             jmp     400fed <func4+0x31>
400fdd: 89 d8             mov     %ebx,%eax
400fdf: 39 fb             cmp     %edi,%ebx
400fe1: 7d 0a             jge     400fed <func4+0x31>
400fe3: 8d 73 01          lea     0x1(%rbx),%esi //esi = rbx + 1
400fe6: e8 d1 ff ff ff    callq   400fbc <func4>
400feb: 01 d8             add     %ebx,%eax
400fed: 5b               pop     %rbx
400fee: c3               retq

0000000000400fef <phase_4>:
400fef: 48 83 ec 18       sub     $0x18,%rsp
```

```

400ff3: 64 48 8b 04 25 28 00 mov    %fs:0x28,%rax
400ffa: 00 00
400ffc: 48 89 44 24 08      mov    %rax,0x8(%rsp) //第二个数
401001: 31 c0              xor    %eax,%eax
401003: 48 8d 4c 24 04      lea    0x4(%rsp),%rcx //第一个数
401008: 48 89 e2          mov    %rsp,%rdx
40100b: be 77 25 40 00      mov    $0x402577,%esi
                        // $0x402577 内为%d %d, 即输入两个整数。
401010: e8 9b fb ff ff      callq  400bb0 <__isoc99_sscanf@plt>
401015: 83 f8 02          cmp    $0x2,%eax
401018: 75 06            jne    401020 <phase_4+0x31>
//以上三行要求之前输入的为两个数据, 否则引爆。
40101a: 83 3c 24 0e      cmpl   $0xe, (%rsp)      //第一个参数<=14
40101e: 76 05            jbe    401025 <phase_4+0x36>
401020: e8 d3 03 00 00      callq  4013f8 <explode_bomb>
//接下去的代码分别是为调用<fun4>函数做准备, 传参数
401025: ba 0e 00 00 00      mov    $0xe,%edx        //%edx=0xe
40102a: be 00 00 00 00      mov    $0x0,%esi        //%esi=0x0
40102f: 8b 3c 24          mov    (%rsp),%edi       //%edi=0xe
401032: e8 85 ff ff ff      callq  400fbc <func4>

401037: 83 f8 0a          cmp    $0xa,%eax
40103a: 75 07            jne    401043 <phase_4+0x54> //返回
值%eax=0xa, 不相等就爆炸

40103c: 83 7c 24 04 0a      cmpl   $0xa,0x4(%rsp)
401041: 74 05            je     401048 <phase_4+0x59> //
0x4(%rsp)=0xa, 不相等就爆炸
401043: e8 b0 03 00 00      callq  4013f8 <explode_bomb>

401048: 48 8b 44 24 08      mov    0x8(%rsp),%rax
40104d: 64 48 33 04 25 28 00 xor    %fs:0x28,%rax
401054: 00 00
401056: 74 05            je     40105d <phase_4+0x6e>
401058: e8 a3 fa ff ff      callq  400b00 <__stack_chk_fail@plt>
40105d: 48 83 c4 18        add    $0x18,%rsp
401061: c3                retq

```

Bomb 4 的开头代码和 Bomb3 相类似。首先和 Bomb 3 类似读取 \$0x402577 内存地址中存储的字符串可以得到输入格式为 %d %d。

根据 `cmp $0x2,%eax` 代码及下面一行可以指导需要输入 2 个数字。

```
(gdb) x/s 0x402577
0x402577: "%d %d"
```

继续看 `phase_4` 函数, 最后有

```

cmp    $0xa,%eax
jne    401043 <phase_4+0x54> //返回值%eax=0xa=10, 不相等就爆炸

```

可以看出只要第二个参数为 10 即可满足，<fun4>的递归过程很复杂，但知道第一个数输入的范围，例举也不失为一种简单粗暴的方法。

答案：3 10

3.5 阶段 5 的破解与分析

密码如下：aaaaal

```
aaaaal
Good work! On to the next...
```

破解过程：

```
0000000000401062 <phase_5>:
401062: 53                push    %rbx
401063: 48 89 fb          mov     %rdi,%rbx
401066: e8 70 02 00 00    callq  4012db <string_length> //调用string_length函数
40106b: 83 f8 06          cmp     $0x6,%eax
40106e: 74 05             je      401075 <phase_5+0x13> //需要输入一个长度为6的字符串
401070: e8 83 03 00 00    callq  4013f8 <explode_bomb>
401075: 48 89 d8          mov     %rbx,%rax
401078: 48 8d 7b 06       lea     0x6(%rbx),%rdi
40107c: b9 00 00 00 00    mov     $0x0,%ecx
401081: 0f b6 10          movzbl (%rax),%edx
401084: 83 e2 0f          and     $0xf,%edx //保留低四位
401087: 03 0c 95 00 24 40 00 add     0x402400(,%rdx,4),%ecx //%ecx+=*(0x402400+%rdx*4)
40108e: 48 83 c0 01       add     $0x1,%rax //循环变量%rax+=1
401092: 48 39 f8          cmp     %rdi,%rax //比较循环变量%rax与%rdi，不等则再循环
401095: 75 ea            jne     401081 <phase_5+0x1f>
401097: 83 f9 3d          cmp     $0x3d,%ecx //比较%ecx是否等于61，不等则引爆
40109a: 74 05             je      4010a1 <phase_5+0x3f>
40109c: e8 57 03 00 00    callq  4013f8 <explode_bomb>
4010a1: 5b              pop     %rbx
4010a2: c3              retq
```

程序流程

1. 调用 `string_length` 的字符串长度函数，测得的长度若不为 6 则引爆；
2. 做一个 6 次的循环，循环变量 `%rax` 从 1 到 6，令 `%edx<-(%rax)`；
3. 只保留 `%edx` 的低四位，并以此作为地址搜索变量，取出 `*(0x402400+%rdx*4)` 中的数，加到 `%ecx` 上，如此循环 6 次后跳出；
4. 比较总和 `%ecx` 的值是否等于 `0x3d` 即 61，相等则返回进入下一关，不等则引爆。

首先我们得知输入的字符串长度为 6，然后通过 `p/x *(0x402400+%rdx*4)` 的指令得到：

```
(gdb) p/x *0x402400
$1 = 0x2
(gdb) p/x *0x402404
$2 = 0xa
(gdb) p/x *0x402408
$3 = 0x6
(gdb) p/x *0x40240c
$4 = 0x1
(gdb) p/x *0x402410
$5 = 0xc
(gdb) p/x *0x402414
$6 = 0x10
(gdb) p/x *0x402418
$7 = 0x9
(gdb) p/x *0x40241c
$8 = 0x3
(gdb) p/x *0x402420
$9 = 0x4
(gdb) p/x *0x402424
$10 = 0x7
(gdb) p/x *0x402428
$11 = 0xe
(gdb) p/x *0x40242c
$12 = 0x5
(gdb) p/x *0x402430
$13 = 0xb
```

上述值分别对应%rdx=0~12。因为字符在机器中是以 ASCII 码的形式存储，因此我们要寻找的一个 6 位字符串要满足其每个字符的 ASCII 码位数对应的值相加为 0x3d 即 61,所以我们不妨取\$2=0xa,\$13=0xb, rdx 分别对应为 1 和 12,又 rdx 是输入字符 ASCII 的低四位，所以答案为 aaaaal。

3.6 阶段 6 的破解与分析

密码如下：4 5 3 1 6 2

```
Good work! On to the next...
4 5 3 1 6 2
Congratulations! You've defused the bomb!
```

破解过程：

```
00000000004010a3 <phase_6>:
4010a3: 41 56          push    %r14
4010a5: 41 55          push    %r13
4010a7: 41 54          push    %r12
4010a9: 55            push    %rbp
4010aa: 53            push    %rbx
```

```

4010ab: 48 83 ec 60      sub    $0x60,%rsp
4010af: 64 48 8b 04 25 28 00 mov    %fs:0x28,%rax
4010b6: 00 00
4010b8: 48 89 44 24 58      mov    %rax,0x58(%rsp)
4010bd: 31 c0             xor    %eax,%eax
4010bf: 48 89 e6           mov    %rsp,%rsi
4010c2: e8 53 03 00 00      callq  40141a <read_six_numbers>
4010c7: 49 89 e4           mov    %rsp,%r12
4010ca: 49 89 e5           mov    %rsp,%r13
//循环 1, 判断 6 个数字是否大于 6 (且不能为 0), 大于 6 则引爆; 判断 6 个数字是否有相
等, 相等则引爆
4010cd: 41 be 00 00 00 00      mov    $0x0,%r14d
4010d3: 4c 89 ed           mov    %r13,%rbp
4010d6: 41 8b 45 00          mov    0x0(%r13),%eax
4010da: 83 e8 01            sub    $0x1,%eax
4010dd: 83 f8 05            cmp    $0x5,%eax
4010e0: 76 05              jbe    4010e7 <phase_6+0x44>  eax<=5
4010e2: e8 11 03 00 00      callq  4013f8 <explode_bomb>
4010e7: 41 83 c6 01          add    $0x1,%r14d
4010eb: 41 83 fe 06          cmp    $0x6,%r14d
4010ef: 74 21              je     401112 <phase_6+0x6f>
4010f1: 44 89 f3            mov    %r14d,%ebx
4010f4: 48 63 c3            movslq %ebx,%rax
4010f7: 8b 04 84            mov    (%rsp,%rax,4),%eax
4010fa: 39 45 00            cmp    %eax,0x0(%rbp)
4010fd: 75 05              jne    401104 <phase_6+0x61>
eax!=(rbp)
4010ff: e8 f4 02 00 00      callq  4013f8 <explode_bomb>
401104: 83 c3 01            add    $0x1,%ebx
401107: 83 fb 05            cmp    $0x5,%ebx
40110a: 7e e8              jle    4010f4 <phase_6+0x51>
40110c: 49 83 c5 04          add    $0x4,%r13
401110: eb c1              jmp     4010d3 <phase_6+0x30>
//循环 2, 注意这里用 7 减去各数值
401112: 48 8d 4c 24 18      lea    0x18(%rsp),%rcx
401117: ba 07 00 00 00      mov    $0x7,%edx
40111c: 89 d0              mov    %edx,%eax
40111e: 41 2b 04 24          sub    (%r12),%eax
401122: 41 89 04 24          mov    %eax,(%r12)
401126: 49 83 c4 04          add    $0x4,%r12
40112a: 4c 39 e1            cmp    %r12,%rcx
40112d: 75 ed              jne    40111c <phase_6+0x79>
//循环 3 开始, 给相应的节点赋地址、赋值
40112f: be 00 00 00 00      mov    $0x0,%esi
401134: eb 1a              jmp     401150 <phase_6+0xad>
401136: 48 8b 52 08          mov    0x8(%rdx),%rdx
40113a: 83 c0 01            add    $0x1,%eax
40113d: 39 c8              cmp    %ecx,%eax
40113f: 75 f5              jne    401136 <phase_6+0x93>
401141: 48 89 54 74 20      mov    %rdx,0x20(%rsp,%rsi,2)
401146: 48 83 c6 04          add    $0x4,%rsi
40114a: 48 83 fe 18          cmp    $0x18,%rsi
40114e: 74 14              je     401164 <phase_6+0xc1>
401150: 8b 0c 34            mov    (%rsp,%rsi,1),%ecx
401153: b8 01 00 00 00      mov    $0x1,%eax
401158: ba f0 32 60 00      mov    $0x6032f0,%edx

```

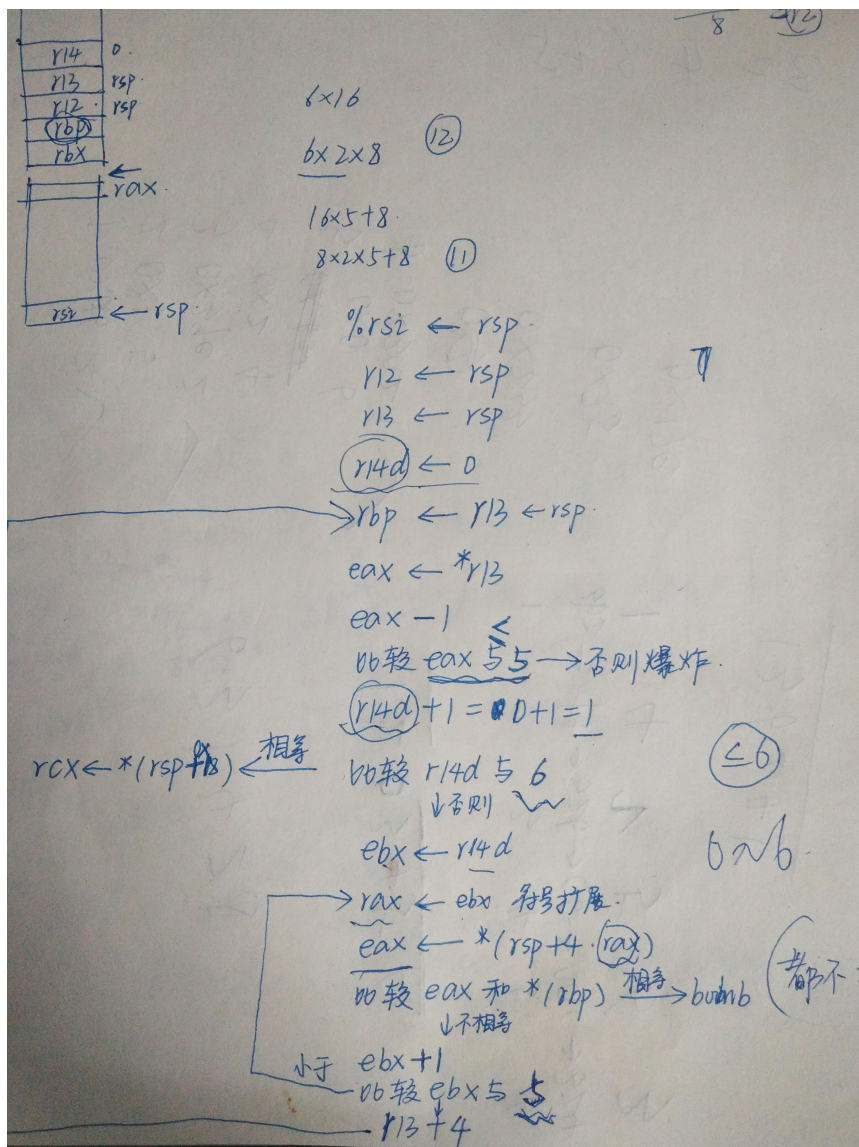
```

40115d: 83 f9 01          cmp     $0x1,%ecx
401160: 7f d4            jg      401136 <phase_6+0x93>
401162: eb dd            jmp     401141 <phase_6+0x9e>
//循环 3 结束
//把节点依次串联起来
401164: 48 8b 5c 24 20    mov     0x20(%rsp),%rbx
401169: 48 8d 44 24 20    lea     0x20(%rsp),%rax
40116e: 48 8d 74 24 48    lea     0x48(%rsp),%rsi
401173: 48 89 d9          mov     %rbx,%rcx
401176: 48 8b 50 08       mov     0x8(%rax),%rdx
40117a: 48 89 51 08       mov     %rdx,0x8(%rcx)
40117e: 48 83 c0 08       add     $0x8,%rax
401182: 48 89 d1          mov     %rdx,%rcx
401185: 48 39 c6          cmp     %rax,%rsi
401188: 75 ec            jne     401176 <phase_6+0xd3>
40118a: 48 c7 42 08 00 00 00 movq    $0x0,0x8(%rdx)
401191: 00
//循环 4 开始
401192: bd 05 00 00 00    mov     $0x5,%ebp
401197: 48 8b 43 08       mov     0x8(%rbx),%rax
40119b: 8b 00            mov     (%rax),%eax
40119d: 39 03            cmp     %eax,(%rbx)
40119f: 7d 05            jge     4011a6 <phase_6+0x103>
4011a1: e8 52 02 00 00    callq   4013f8 <explode_bomb>
4011a6: 48 8b 5b 08       mov     0x8(%rbx),%rbx
4011aa: 83 ed 01          sub     $0x1,%ebp
4011ad: 75 e8            jne     401197 <phase_6+0xf4>
//循环 4 结束
4011af: 48 8b 44 24 58    mov     0x58(%rsp),%rax
4011b4: 64 48 33 04 25 28 00 xor     %fs:0x28,%rax
4011bb: 00 00
4011bd: 74 05            je      4011c4 <phase_6+0x121>
4011bf: e8 3c f9 ff ff    callq   400b00 <__stack_chk_fail@plt>
4011c4: 48 83 c4 60       add     $0x60,%rsp
4011c8: 5b              pop     %rbx
4011c9: 5d              pop     %rbp
4011ca: 41 5c            pop     %r12
4011cc: 41 5d            pop     %r13
4011ce: 41 5e            pop     %r14
4011d0: c3              retq

```

程序流程:

1. 调用函数 `read_six_numbers`，读入 6 个数字；
2. （循环 1）判断 6 个数字是否大于 6（且不能为 0），大于 6 则引爆；
3. （循环 1）判断 6 个数字是否有相等，相等则引爆；
4. （循环 2）根据输入的参数找到索引的节点，注意这里用 7 减去各数值；
5. （循环 3）给相应的节点赋地址、赋值；
6. 把节点依次串联起来；
7. （循环 4）判断前一节点值是否大于等于下一节点，不是则引爆；
8. 炸弹破解成功，撤销栈帧，返回。




```

(gdb) p/x *0x6032f0
$1 = 0xd6
(gdb) p/x *0x6032f8
$2 = 0x603300
(gdb) p/x *0x603300
$3 = 0x39e
(gdb) p/x *0x603308
$4 = 0x603310
(gdb) p/x *0x603310
$5 = 0x3a2
(gdb) p/x *0x603318
$6 = 0x603320
(gdb) p/x *0x603320
$7 = 0x233
(gdb) p/x *0x603328
$8 = 0x603330
(gdb) p/x *0x603330
$9 = 0x75
(gdb) p/x *0x603338
$10 = 0x603340
(gdb) p/x *0x603340
$11 = 0x17e

```

可得各节点的数据依次是 0xd6、0x39e、0x3a2、0x233、0x75、0x17e。

进一步, 程序要求节点的数据从前到后按照从大到小的顺序排列, 因此排列成: 0x233<3>、0x39e<2>、0x17e<4>、0x75<6>、0x3a2<1>、0xd6<5>, 由循环 2 可知, 应该用 7 减去每个数, 所以最后答案为 4 5 3 1 6 2

3.7 阶段 7 的破解与分析(隐藏阶段)

密码如下: 20

破解过程:

在 phase_defused 函数中, 找到隐藏关的入口, 如图

```

4015bf:    be ca 25 40 00      mov     $0x4025ca,%esi
4015c4:    48 8d 7c 24 10      lea     0x10(%rsp),%rdi
4015c9:    e8 2b fd ff ff      callq  4012f9 <strings_not_equal>

```

```

(gdb) x/s 0x4025ca
0x4025ca:    "DrEvil"

```

即在第四关的答案后面加字符 “DrEvil” 才能进入隐藏关。

```

000000000040120f <secret_phase>:
40120f: 53                push    %rbx
401210: e8 44 02 00 00    callq  401459 <read_line>
401215: ba 0a 00 00 00    mov     $0xa,%edx
40121a: be 00 00 00 00    mov     $0x0,%esi
40121f: 48 89 c7          mov     %rax,%rdi
401222: e8 69 f9 ff ff    callq  400b90 <strtol@plt>
401227: 48 89 c3          mov     %rax,%rbx
40122a: 8d 40 ff          lea     -0x1(%rax),%eax
40122d: 3d e8 03 00 00    cmp     $0x3e8,%eax
401232: 76 05            jbe     401239 <secret_phase+0x2a>
401234: e8 bf 01 00 00    callq  4013f8 <explode_bomb>
401239: 89 de            mov     %ebx,%esi
40123b: bf 10 31 60 00    mov     $0x603110,%edi
401240: e8 8c ff ff ff    callq  4011d1 <fun7>
401245: 83 f8 02          cmp     $0x2,%eax
401248: 74 05            je      40124f <secret_phase+0x40>
40124a: e8 a9 01 00 00    callq  4013f8 <explode_bomb>
40124f: bf 40 24 40 00    mov     $0x402440,%edi
401254: e8 87 f8 ff ff    callq  400ae0 <puts@plt>
401259: e8 21 03 00 00    callq  40157f <phase_defused>
40125e: 5b                pop     %rbx
40125f: c3                retq

```

这一段代码一开始就调用 `read_line`，然后会把内容用 `strtol` 转成十进制整数，然后和 `0x3e8`（也就是 1000）进行比较，如果小于等于的话就执行 `fun7`，然后返回值需要等于 2，于是问题就变成，给定一个值，让 `fun7` 的输出为 2。我们就先来看看 `fun7` 具体做了什么。

```

00000000004011d1 <fun7>:
4011d1: 48 83 ec 08       sub     $0x8,%rsp
4011d5: 48 85 ff          test    %rdi,%rdi
4011d8: 74 2b            je      401205 <fun7+0x34>
4011da: 8b 17            mov     (%rdi),%edx
4011dc: 39 f2            cmp     %esi,%edx
4011de: 7e 0d            jle     4011ed <fun7+0x1c>
4011e0: 48 8b 7f 08       mov     0x8(%rdi),%rdi
4011e4: e8 e8 ff ff ff    callq  4011d1 <fun7>
4011e9: 01 c0            add     %eax,%eax
4011eb: eb 1d            jmp     40120a <fun7+0x39>
4011ed: b8 00 00 00 00    mov     $0x0,%eax
4011f2: 39 f2            cmp     %esi,%edx
4011f4: 74 14            je      40120a <fun7+0x39>
4011f6: 48 8b 7f 10       mov     0x10(%rdi),%rdi
4011fa: e8 d2 ff ff ff    callq  4011d1 <fun7>
4011ff: 8d 44 00 01       lea     0x1(%rax,%rax,1),%eax
401203: eb 05            jmp     40120a <fun7+0x39>
401205: b8 ff ff ff ff    mov     $0xffffffff,%eax
40120a: 48 83 c4 08       add     $0x8,%rsp
40120e: c3                retq

```

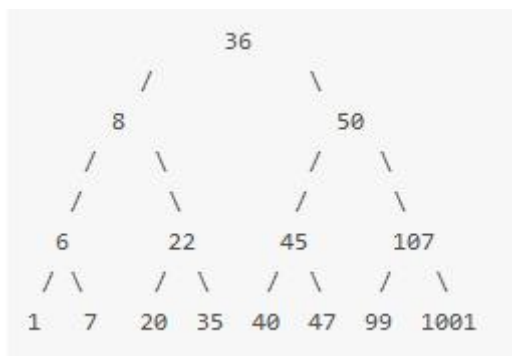
一眼就能看出这是一个递归函数了，然后我们观察一下传进来作为第一个参数的地址 `0x603110`


```

syq1160300901@syq1160300901-virtual-machine: ~/bomb265
(gdb) x/120 0x603110
0x603110 <n1>: 36      0      6304048 0
0x603120 <n1+16>:      6304080 0      0      0
0x603130 <n21>: 8      0      6304176 0
0x603140 <n21+16>:      6304112 0      0      0
0x603150 <n22>: 50     0      6304144 0
0x603160 <n22+16>:      6304208 0      0      0
0x603170 <n32>: 22     0      6304400 0
0x603180 <n32+16>:      6304336 0      0      0
0x603190 <n33>: 45     0      6304240 0
0x6031a0 <n33+16>:      6304432 0      0      0
0x6031b0 <n31>: 6      0      6304272 0
0x6031c0 <n31+16>:      6304368 0      0      0
0x6031d0 <n34>: 107    0      6304304 0
0x6031e0 <n34+16>:      6304464 0      0      0
0x6031f0 <n45>: 40     0      0      0
0x603200 <n45+16>:      0      0      0      0
0x603210 <n41>: 1      0      0      0
0x603220 <n41+16>:      0      0      0      0
0x603230 <n47>: 99     0      0      0
0x603240 <n47+16>:      0      0      0      0
0x603250 <n44>: 35     0      0      0
0x603260 <n44+16>:      0      0      0      0
0x603270 <n42>: 7      0      0      0
---Type <return> to continue, or q <return> to quit---
0x603280 <n42+16>:      0      0      0      0
0x603290 <n43>: 20     0      0      0
0x6032a0 <n43+16>:      0      0      0      0
0x6032b0 <n46>: 47     0      0      0
0x6032c0 <n46+16>:      0      0      0      0

```

可以看出是一棵树，有不同的层级。画出来的话大概是



递归实际上的逻辑类似于下面代码：

```

struct treeNode
{
    int data;

```

```
struct treeNode* leftChild;
struct treeNode* rightChild;
};
int fun7(struct treeNode* p, int v)
{
    if (p == NULL)
        return -1;
    else if (v < p->data)
        return 2 * fun7(p->leftChild, v);
    else if (v == p->data)
        return 0;
    else
        return 2 * fun7(p->rightChild, v) + 1;
}
```

为了要凑成 2，我们需要的值是 20（根据递归规律来找到合适的数字即可）

```
syq1160300901@syq1160300901-virtual-machine:~/bomb265$ ./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...
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
```

第 4 章 总结

4.1 请总结本次实验的收获

1. 熟练掌握了计算机系统的 ISA 指令系统与寻址方式，并且熟练掌握 Linux 下调试器的反汇编调试跟踪分析机器语言的方法
2. 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
3. 学习了利用 GDB 调试工具来调试程序。整个二进制炸弹的程序分别用到了直接地址寻址、循环语句、switch 语句、递归函数、字符串 ASCII 码转换、链表、函数调用等知识，其中第六关的链表结构最为复杂，难度较大。破解整个炸弹的过程使我认识和熟悉了汇编语言的应用，是一次很好的学习提高过程。
4. 通过这次『拆弹』的历练，我对数据在内存中以及汇编的表示方法有了更加深刻的认识，做得过程可能有时候会摸不着头脑，这个时候一定要冷静，相信自己，找到正确的方法。

4.2 请给出对本次实验内容的建议

1. 有些指令描述的不是很清晰

■ 安装# install dependencies

```
▪ sudo apt-get install  cmake  build-essential  libboost-dev  \
▪  libqt5xmlpatterns5-dev  qtbase5-dev  qt5-default  \
▪  libqt5svg5-dev  libgraphviz-dev  libcapstone-dev
, ...
```

这里，sudo apt-get install 是每个指令的前提，具有迷惑性。

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>.