

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 计算学部

学 号 1190200717

班 级 1903008

学 生 梁浩

指 导 教 师 吴锐

实 验 地 点 G709

实 验 日 期 2021/04/26

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).....	- 22 -
第 4 章 总结.....	- 27 -
4.1 请总结本次实验的收获.....	- 27 -
4.2 请给出对本次实验内容的建议.....	- 27 -
参考文献.....	- 28 -

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

处理器: Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40GHz

已安装的内存(RAM): 8.00GB(7.81GB 可用)

系统类型: 64 位操作系统, 基于 x64 的处理器

1.2.2 软件环境

Windows 10 家庭中文版; VirtualBox 6.1; Ubuntu 20.04

1.2.3 开发工具

GDB/OBJDUMP; EDB

1.3 实验预习

上实验课前, 必须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。

请写出 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)、O0、O1、O2、O3, -m32/m64。再次查看生成的汇编语

言与原来的区别。

注意 O1 之后无栈帧，EBP 做别的用途。-fno-omit-frame-pointer 加上栈指针。

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。

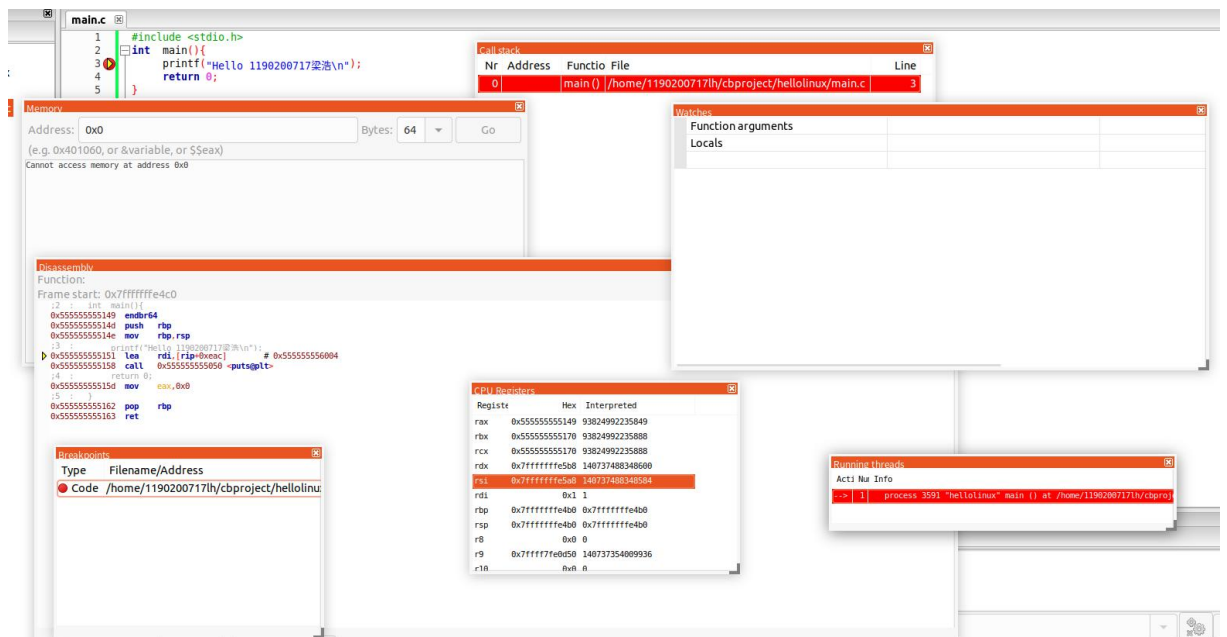


图 2-1 Ubuntu 下 CodeBlocks 反汇编截图

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

计算机系统实验报告

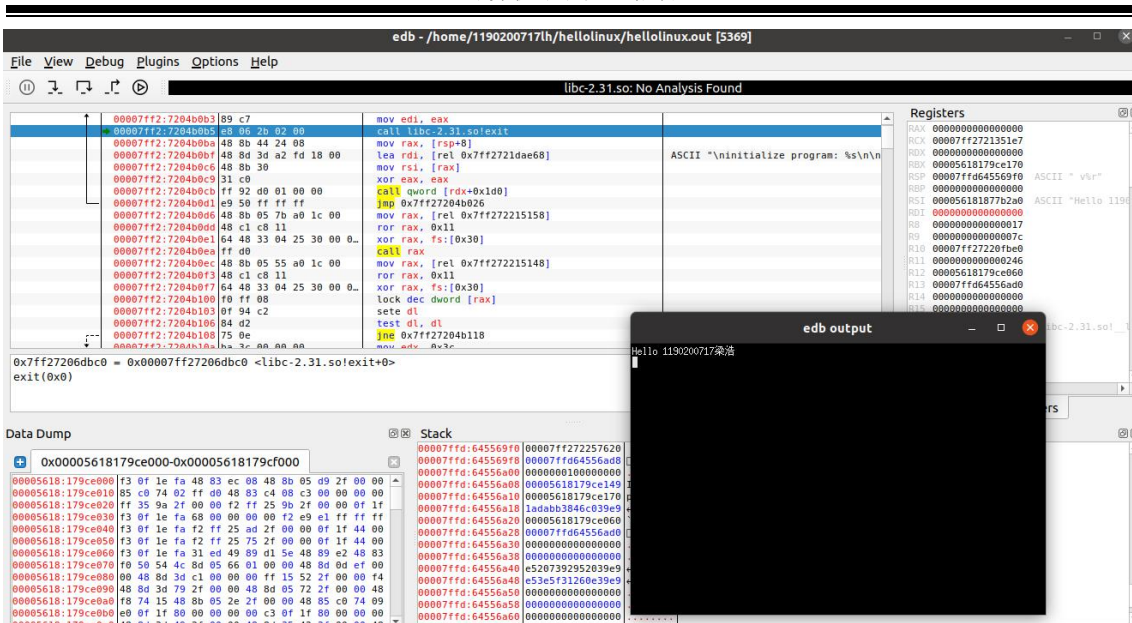


图 2-2 Ubuntu 下 EDB 截图

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：I am just a renegade hockey mom.

破解过程：

```

00000000004013f9 <phase_1>:
4013f9: 55                push    %rbp
4013fa: 48 89 e5          mov     %rsp,%rbp
4013fd: be 50 31 40 00    mov     $0x403150,%esi
401402: e8 01 04 00 00    callq  401808 <strings_not_equal>
401407: 85 c0            test    %eax,%eax
401409: 75 02            jne     40140d <phase_1+0x14>
40140b: 5d              pop     %rbp
40140c: c3              retq
40140d: e8 f2 04 00 00    callq  401904 <explode_bomb>
401412: eb f7            jmp     40140b <phase_1+0x12>
  
```

Handwritten annotations:

- je: ZF=1时 跳转 (if ZF=1, jump)
- jne: ZF=0时 跳转 (if ZF=0, jump)
- 0x403150 这个地址里 密码存在 (password is at this address)
- <strings_not_equal> 相等, 返回0; 不相等, 返回1 (if equal, return 0; if not equal, return 1)

<string_not_equals>函数用来比较输入的字符串和 0x403150 下存储的字符串是否相等，如果相等返回 0，不相等返回 1，返回值存在 %eax 中，如果 %eax 不为 0，跳转到 40140d 爆炸。

先查看存储在 0x403150 下的字符串：

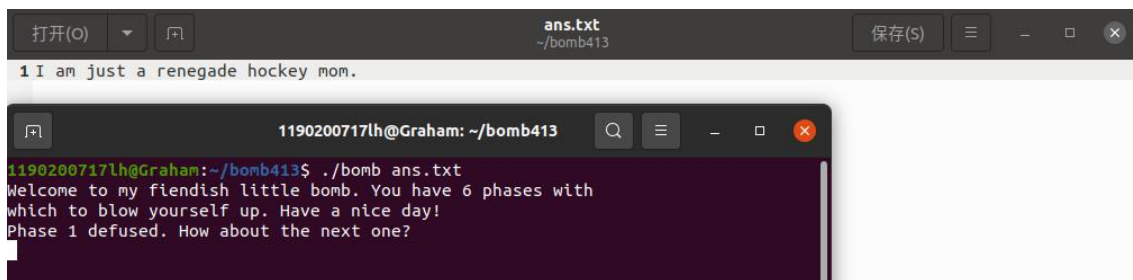
```

(gdb) b *0x4013fd
Breakpoint 1 at 0x4013fd: file phases.c, line 22.
(gdb) r
Starting program: /home/1190200717lh/bomb413/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
what

Breakpoint 1, phase_1 (input=0x405780 <input_strings> "what") at phases.c:22
22  phases.c: 没有那个文件或目录.
(gdb) ni
0x0000000000401402 22 in phases.c
(gdb) x/s 0x403150
0x403150: "I am just a renegade hockey mom."
(gdb)
  
```

得到密码:I am just a renegade hockey mom. 将密码存到 ans.txt 文件中。

再次运行程序，



3.2 阶段 2 的破解与分析

密码如下：0 1 3 6 10 15(要满足第一个数不小于 0)

前一项和后一项之间要满足： $a[n] = a[n-1] + n (n \geq 1, a[0] \geq 0)$

破解过程：

000000000401414 <phase_2>:

401414:	55	push	%rbp
401415:	48 89 e5	mov	%rsp,%rbp

o/o rsp, %o rbp 等价

401418:	53	push	%rbx
401419:	48 83 ec 28	sub	\$0x28,%rsp
40141d:	48 8d 75 d0	lea	-0x30(%rbp),%rsi
401421:	e8 00 05 00 00	callq	401926 <read_six_numbers>
401426:	83 7d d0 00	cmpl	\$0x0,-0x30(%rbp), 第几个数? a[0]
40142a:	78 07	js	401433 <phase_2+0x1f> 炸 < a[0]-0 < 0, 炸
40142c:	bb 01 00 00 00	mov	\$0x1,%ebx 0xebx:1
401431:	eb 0f	jmp	401442 <phase_2+0x2e>
401433:	e8 cc 04 00 00	callq	401904 <explode_bomb>
401438:	eb f2	jmp	40142c <phase_2+0x18>
40143a:	e8 c5 04 00 00	callq	401904 <explode_bomb>
40143f:	83 c3 01	add	\$0x1,%ebx 2
401442:	83 fb 05	cmp	\$0x5,%ebx
401445:	7f 17	jl	40145e <phase_2+0x4a>
401447:	48 63 c3	movslq	%ebx,%rax
40144a:	8d 53 ff	lea	-0x1(%rbx),%edx
40144d:	48 63 d2	movslq	%edx,%rdx
401450:	89 d9	mov	%ebx,%ecx
401452:	03 4c 95 d0	add	-0x30(%rbp,%rdx,4),%ecx
401456:	39 4c 85 d0	cmpl	%ecx,-0x30(%rbp,%rax,4) a[1]+2 a[2]
40145a:	74 e3	je	40143f <phase_2+0x2b>
40145c:	eb dc	jmp	40143a <phase_2+0x26> 炸
40145e:	48 83 c4 28	add	\$0x28,%rsp
401462:	5b	pop	%rbx
401463:	5d	pop	%rbp
401464:	c3	retq	

48 4个字节 - 一个 int long?

结果加偏移 (js)

如果大于

如果 2f = 1, 2f = 0

条件 炸

条件 %ebx > 5

左侧计算过程：

$$\begin{aligned} & \text{for } bp + 4 \text{ } \%rbp + 4 - 0 = 30 \\ & 1 + a(0) = a(1) \quad \%rbp = 1 \\ & 2 + a(1) = a(2) \quad \%rbp = 2 \\ & 3 + a(2) = a(3) \quad \%rbp = 3 \\ & 4 + a(3) = a(4) \quad \%rbp = 4 \\ & 5 + a(4) = a(5) \quad \%rbp = 5 \\ & a(0) = 0 \\ & \quad 1 \\ & \quad 3 \quad 15 \\ & \quad 6 \\ & \quad 10 \end{aligned}$$

0000000000401926 <read_six_numbers>:

```

401926: 55          push    %rbp
401927: 48 89 e5    mov     %rsp,%rbp
40192a: 48 89 f2    mov     %rsi,%rdx
40192d: 48 8d 4e 04 lea     0x4(%rsi),%rcx
401931: 48 8d 46 14 lea     0x14(%rsi),%rax
401935: 50          push    %rax
401936: 48 8d 46 10 lea     0x10(%rsi),%rax
40193a: 50          push    %rax
40193b: 4c 8d 4e 0c lea     0xc(%rsi),%r9
40193f: 4c 8d 46 08 lea     0x8(%rsi),%r8
401943: be 23 33 40 00 mov     $0x403323,%esi
401948: b8 00 00 00 00 mov     $0x0,%eax
40194d: e8 be f7 ff ff callq   401110 <__isoc99_sscanf@plt> 返回读到数字个数
401952: 48 83 c4 10 add     $0x10,%rsp
401956: 83 f8 05    cmp     $0x5,%eax
401959: 7e 02      %eax ≤ 5    jle     40195d <read_six_numbers+0x37> 炸
40195b: c9          leaveq
40195c: c3          retq
40195d: e8 a2 ff ff ff callq   401904 <explode_bomb>

```

Handwritten notes in the image:

- For the `lea` instructions, the offsets `0x4`, `0x14`, `0x10`, `0xc`, and `0x8` are circled in blue.
- A blue box on the right contains a table mapping these offsets to registers:

Offset	Register
0x0	%rdx
0x4	%rcx
0x8	%r8
0xc	%r9
0x10	%rax
0x14	%rax ?
- Red handwritten notes include:
 - `%eax ≤ 5` with a downward arrow pointing to the `jle` instruction.
 - `如果读到的少于 6 个就炸` (If fewer than 6 are read, it explodes).
 - `返回读到数字个数` (Return the number of digits read).

(1) 先从<phase_2>分析出 6 个数之间的关系

从 401426—>40142a 可以看出输入的第一个数要不小于 0，否则直接爆炸。从 401445 和 40145e 可以看出循环终止的条件是 $\%ebx > 5$ 。从 401447—>40145c 可以看出如果满足 $a[n] = a[n-1] + n (n \geq 1, a[0] \geq 0)$ 炸弹就不会爆炸。

(2) 再从<read_six_numbers>中分析出输入的格式

一开始是给 6 个数分配空间，从 401943 可以看出输入的格式保存在 0x403323 中，`%eax` 是用来记录读到的数字个数，如果个数少于 6 个，直接爆炸，但是如果多于 6 个也不会引起爆炸。

(3) 利用 gdb 查看输入格式

```

(gdb) r
Starting program: /home/1190200717lh/bomb413/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am just a renegade hockey mom.
Phase 1 defused. How about the next one?
sad

Breakpoint 1, 0x0000000000401943 in read_six_numbers (
    input=0x4057d0 <input_strings+80> "sad",
    numbers=numbers@entry=0x7fffffffde10) at support.c:71
71      support.c: 没有那个文件或目录.
(gdb) ni
0x0000000000401948      71      in support.c
(gdb) x/s 0x403323
0x403323: "%d %d %d %d %d %d"

```

(4) 用 0 1 3 6 10 15 进行结果验证

```

ans.txt
~/bomb413

1 I am just a renegade hockey mom.
2 0 1 3 6 10 15

1190200717lh@Graham: ~/bomb413
1190200717lh@Graham:~/bomb413$ ./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!

```

3.3 阶段 3 的破解与分析

密码如下: {[0,180],[1,874],[2,364],[3,289],[4,83],[5,702],[6,639],[7,718]}

破解过程:

```

000000000401465 <phase_3>:
401465: 55                push    %rbp
401466: 48 89 e5          mov     %rsp,%rbp
401469: 48 83 ec 10       sub     $0x10,%rsp
40146d: 48 8d 4d f8       lea     -0x8(%rbp),%rcx
401471: 48 8d 55 fc       lea     -0x4(%rbp),%rdx
401475: be 2f 33 40 00    mov     $0x40332f,%esi
40147a: b8 00 00 00 00    mov     $0x0,%eax
40147f: e8 8c fc ff ff   callq  401110 <__isoc99_sscanf@plt>
401484: 83 f8 01         cmp     $0x1,%eax
401487: 7e 11            jle     40149a <phase_3+0x35> 炸
401489: 8b 45 fc         mov     -0x4(%rbp),%eax
40148c: 83 f8 07         cmp     $0x7,%eax
40148f: 77 46            jbe     4014d7 <phase_3+0x72> 炸
401491: 89 c0            mov     %eax,%eax
401493: ff 24 c5 a0 31 40 jmpq    4014031a0(%rax,8) 取 0x4031a0 + 8 * %rax
40149a: e8 65 04 00 00    callq  401904 <explode_bomb> 炸

O(0): 0 -> 401401 180
      1 -> 4014e3 874
      2 -> 4014ad 0x16c (364)

3 -> 4014b4 0x171 (389) 5 -> 4014c2 0x2be (702)
4 -> 4014bb 0x5d (83) 6 -> 4014c9 0x27f (639)
      > -> 4014d0 0x2ce (718)

40149f: eb e8            jmp     401489 <phase_3+0x24>
4014a1: b8 b4 00 00 00    mov     $0xb4,%eax
4014a6: 39 45 f8         cmp     %eax,-0x8(%rbp)
4014a9: 75 3f            jne     4014ea <phase_3+0x85> 炸 %eax != -0x8 (%rbp)
4014ab: c9              leaveq  %eax,%rbp
4014ac: c3              retq
4014ad: b8 6c 01 00 00    mov     $0x6c,%eax
4014b2: eb f2            jmp     4014a6 <phase_3+0x41> 验证 a(1) = 0x16c
4014b4: b8 21 01 00 00    mov     $0x21,%eax
4014b9: eb eb            jmp     4014a6 <phase_3+0x41> 验证 a(1) = 0x12
4014bb: b8 53 00 00 00    mov     $0x53,%eax
4014c0: eb e4            jmp     4014a6 <phase_3+0x41>
4014c2: b8 be 02 00 00    mov     $0x2be,%eax
4014c7: eb dd            jmp     4014a6 <phase_3+0x41>
4014c9: b8 7f 02 00 00    mov     $0x7f,%eax
4014ce: eb d6            jmp     4014a6 <phase_3+0x41>
4014d0: b8 ce 02 00 00    mov     $0x2ce,%eax
4014d5: eb cf            jmp     4014a6 <phase_3+0x41>
4014d7: e8 28 04 00 00    callq  401904 <explode_bomb>
4014dc: b8 00 00 00 00    mov     $0x0,%eax
4014e1: eb c3            jmp     4014a6 <phase_3+0x41>
4014e3: b8 6a 03 00 00    mov     $0x6a,%eax
4014e8: eb bc            jmp     4014a6 <phase_3+0x41>
4014ea: e8 15 04 00 00    callq  401904 <explode_bomb>
4014ef: eb ba            jmp     4014ab <phase_3+0x46> (结束)

```

(1) 利用 gdb 查看 0x40332f 地址下存储的内容

```
Breakpoint 1, 0x0000000000401475 in phase_3 (
    input=0x405820 <input_strings+160> "das") at phases.c:66
66 phases.c: 没有那个文件或目录.
(gdb) ni
0x000000000040147a      66      in phases.c
(gdb) x/s 0x40332f
0x40332f:      "%d %d"
```

可以看出输入的应是两个 int 型的数，在这里不妨设为 $a[0]$ ， $a[1]$

(2) 分析 $a[0]$ 的取值要求

```
40148c: 83 f8 07      cmp     $0x7,%eax
40148f: 77 46      如果 %eax > 7 ja     4014d7 <phase_3+0x72> 炸
401491: 89 c0      a[0] > 7 mov     %eax,%eax 扩展
```

$a[0] \leq 7$

从这里可以知道， $a[0]$ 满足 $0 \leq a[0] \leq 7$

(3) 根据 $a[0]$ 的值分析跳转情况

```
401493: ff 24 c5 a0 31 40 00 jmpq    *0x4031a0(,%rax,8)
```

将 $0x4031a0 + 8\%rax$ 里的值作为跳转的地址，查看 $0x4031a0 \rightarrow 0x4031d8$ 里存的值

```
Contents of section .rodata:
4031a0 a1144000 00000000 e3144000 00000000 ..@.....@.....
4031b0 ad144000 00000000 b4144000 00000000 ..@.....@.....
4031c0 bb144000 00000000 c2144000 00000000 ..@.....@.....
4031d0 c9144000 00000000 d0144000 00000000 ..@.....@.....
```

由上，可以分析出 $a[0]$ 取不同的值时程序的跳转情况

以 $a[0] = 0$ 为例，程序跳转到 $0x4014a1$

```
0 4014a1: b8 b4 00 00 00 mov     $0xb4,%eax
    4014a6: 39 45 f8      cmp     %eax,-0x8(%rbp)
    4014a9: 75 3f      jne     4014ea <phase_3+0x85> 炸 %eax != -0x8 (%rbp)
    4014ab: c9      leaveq
    4014ac: c3      retq    结束 (%eax = -0x8 (%rbp))
```

比较 $a[1]$ 是否和 $0xb4$ （十进制表示为 180）相等，若相等则退出，不相等则爆炸，因此其中一个密码对就是 $[0, 180]$ ，同上可以分析出其他情况。

(1) 先分析<phase_4>

从这里我们可以猜想读入的是两个 int 类型的数, 利用 gdb 查看 0x40332f 地址下的内容, 猜想正确

从 401557—>401560 我们可以看出输入的第一个数 $a[0]$ 要满足 $0 \leq a[0] \leq 14$,

否则炸弹会爆炸

```

401577: 83 f8 1f          cmp     $0x1f,%eax 递归返回值
40157a: 75 06             jne     401582 <phase_4+0x52> 炸
40157c: 83 7d f8 1f      cmpl    $0x1f,-0x8(%rbp)
401580: 74 05             je      401587 <phase_4+0x57> 退出
炸 401582: e8 7d 03 00 00    callq   401904 <explode_bomb> 炸
401587: c9               leaveq
401588: c3               retq

```

从上面一段代码中我们可以分析出<func4>这个函数最终的返回值应该是 0x1f，而且输入的第二个数 $a[1]$ 是 0x1f

(2) 再分析<func4>

通过分析可以发现 func4 里面存在着递归调用，递归基础是 $\%rdi = \%rbx(\%rdi = \frac{\%rsi + \%rdx}{2})$ ，我们可以将 func4 转化成 C 语言版本的程序，

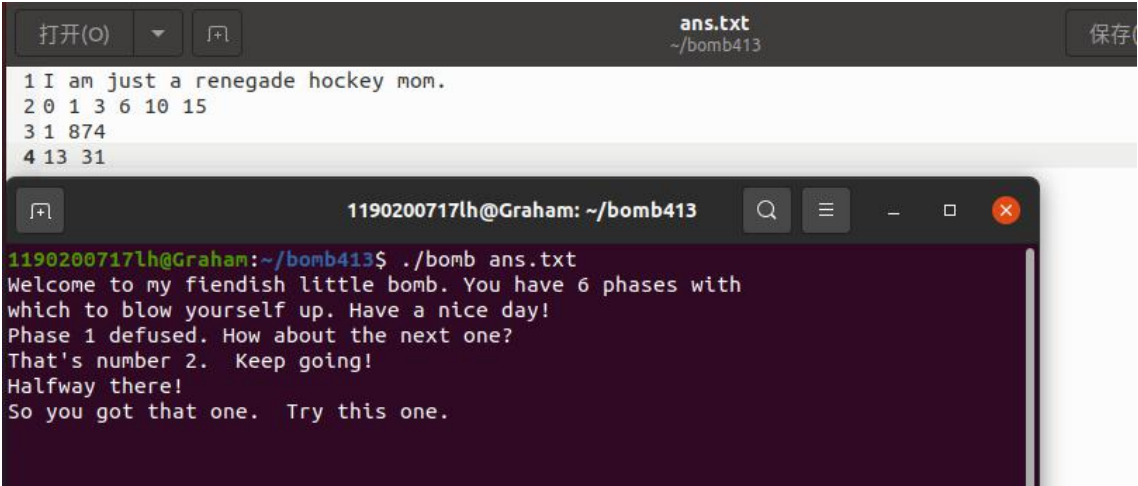
```

#include <stdio.h>
static int rax = 0;
int func4(int rdi, int rsi, int rdx);
int main() {
    int res = func4(13, 0, 14);
    printf("%d\n", res);
}
int func4(int rdi, int rsi, int rdx) {
    int rbx = (rsi + rdx) / 2;
    if (rdi < rbx) { // 小
        rdx = rbx - 1;
        rax = func4(rdi, rsi, rbx - 1);
        rbx = rbx + rax;
        return rbx;
    }
    else if (rdi > rbx) { // 大
        rsi = rbx + 1;
        rax = func4(rdi, rbx + 1, rdx);
        rbx = rbx + rax;
        return rbx;
    }
    else {
        return rbx;
    }
}

```

由此，如果要想 func4 函数最终的返回值是 31（十进制）的话，输入的第一个数应该是 13

(3)验证成功



3.5 阶段 5 的破解与分析

密码如下： 5 115

破解过程:

0000000000401589 <phase_5>:

```
401589: 55          push    %rbp
40158a: 48 89 e5    mov     %rsp,%rbp
40158d: 48 83 ec 10 sub     $0x10,%rsp

401591: 48 8d 4d f8 lea     -0x8(%rbp),%rcx
401595: 48 8d 55 fc lea     -0x4(%rbp),%rdx
401599: be 2f 33 40 00 mov     $0x40332f,%esi  a[0]  a[0]
                                     %rdx  %rcx
40159e: b8 00 00 00 00 mov     $0x0,%eax
4015a3: e8 68 fb ff ff callq   401110 <_isoc99_sscanf@plt>  返回到被调函数位置
4015a8: 83 f8 01    cmp     $0x1,%eax
4015ab: 7e 2e      jle     4015db <phase_5+0x52> 炸
4015ad: 8b 45 fc    mov     -0x4(%rbp),%eax  %eax = a[0]
4015b0: 83 e0 0f    and     $0xf,%eax  a[0]和f按位与之后给%eax
4015b3: 89 45 fc    mov     %eax,-0x4(%rbp)  a[0] = %eax
4015b6: b9 00 00 00 00 mov     $0x0,%ecx  %ecx = 0
4015bb: ba 00 00 00 00 mov     $0x0,%edx  %edx = 0
4015c0: 8b 45 fc    mov     -0x4(%rbp),%eax  %eax = a[0]
4015c3: 83 f8 0f    cmp     $0xf,%eax
4015c6: 74 1a      je      4015e2 <phase_5+0x59>  a[0] = 0xf, 0xff, 0x??
                                     如果%eax = 0xf 则进
4015c8: 83 c2 01    add     $0x1,%edx  %edx = 0 + 1 = 1
4015cb: 48 98      cltq    第3个进位
4015cd: 8b 04 85 e0 31 40 00 mov     0x4031e0(%rax,4),%eax  (0x4031e0 + 4 - %rax) 中的内容给 %eax
4015d4: 89 45 fc    mov     %eax,-0x4(%rbp)  a[0] = %eax
4015d7: 01 c1      add     %eax,%ecx  %ecx = %eax + %ecx  进过进位,
4015d9: eb e5      jmp     4015c0 <phase_5+0x37>  0x0 ≤ %rax ≤ 0xf
4015db: e8 24 03 00 00 callq   401904 <explode_bomb> 炸
4015de: eb cb      jmp     4015ad <phase_5+0x24>
4015e2: 83 fa 0f    cmp     $0xf,%edx
4015e5: 75 05      jne     4015ec <phase_5+0x63>  %edx = 0xf
4015e7: 39 4d f8    cmp     %ecx,-0x8(%rbp)
4015ea: 74 05      je      4015f1 <phase_5+0x68>  a[0] = %ecx
4015ec: e8 13 03 00 00 callq   401904 <explode_bomb> 炸
4015f1: c9        leaveq  %eax,%edi  退出
4015f2: c3        retq
```

(1) 利用 gdb 查看 0x40332f 地址下存储的内容

```
Breakpoint 1, 0x0000000000401475 in phase_3 (
    input=0x405820 <input_strings+160> "das") at phases.c:66
66 phases.c: 没有那个文件或目录.
(gdb) ni
0x000000000040147a      66      in phases.c
(gdb) x/s 0x40332f
0x40332f:      "%d %d"
```

可以看出输入的应是两个 int 型的数，在这里不妨设为 $a[0]$ ， $a[1]$

(2) 查看 0x403140 附近地址下的内容

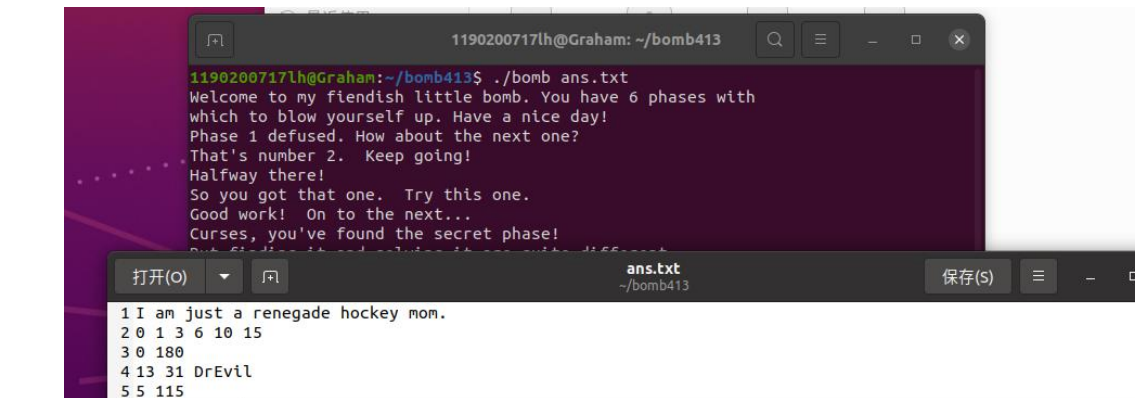
```
Contents of section .rodata:
4031e0 0a000000 02000000 0e000000 07000000 .....
4031f0 08000000 0c000000 0f000000 0b000000 .....
403200 00000000 04000000 01000000 0d000000 .....
403210 03000000 09000000 06000000 05000000 .....
```

对应着写好：

0x4031e0	0x0a	0	11
e4	0x02	1	13
e8	0x0e	2	14
ec	0x07	3	4
f0	0x08	4	9
f4	0x0c	5	2
f8	0x0f	6	16
fc	0x0b	7	5
200	0x00	8	10
204	0x04	9	8
208	0x01	a	12
20c	0x0d	b	6
210	0x03	c	3
214	0x09	d	7
218	0x06	e	15
21c	0x05	f	1

4015c0:	8b 45 fc	mov	-0x4(%rbp),%eax	<u>%eax = a[0]</u>
4015c3:	83 f8 0f	cmp	\$0xf,%eax	
4015c6:	74 1a	je	4015e2 <phase_5+0x59>	$a[0] = 0xf, 0xff, 0x??$
4015c8:	83 c2 01	add	\$0x1,%edx	$\%edx = 0 + 1 = 1$
4015cb:	48 98	cltq		符号扩展
4015cd:	8b 04 85 e0 31 40 00	mov	0x4031e0(,%rax,4),%eax	$(0 \times 4031e0 + 4 \cdot \%rax)$ 中的内容给 %eax
4015d4:	89 45 fc	mov	%eax,-0x4(%rbp)	$a[0] = \%eax$
4015d7:	01 c1	add	%eax,%ecx	$\%ecx = \%eax + \%ecx$
4015d9:	eb e5	jmp	4015c0 <phase_5+0x37>	
4015db:	e8 24 03 00 00	callq	401904 <explode_bomb>	炸
4015e0:	eb cb	jmp	4015ad <phase_5+0x24>	
4015e2:	83 fa 0f	cmp	\$0xf,%edx	
4015e5:	75 05	jne	4015ec <phase_5+0x63>	$\%edx = 0xf$
4015e7:	39 4d f8	cmp	%ecx,-0x8(%rbp)	
4015ea:	74 05	je	4015f1 <phase_5+0x68>	$a[1] = \%ecx$
4015ec:	e8 13 03 00 00	callq	401904 <explode_bomb>	炸
4015f1:	c9	leaveq		退出
4015f2:	c3	retq		

如果要正常退出，当`%eax = 0xf`的时候，`%edx` 也应是 `0xf`，满足这样的条件后，比较 `a[1]` 和 `%ecx` 的值。通过分析，只有当 `a[0] = 0x5`（十进制表示为 5）时，跳转多次后正好满足 `%eax = %edx = 0xf`，此时 `%ecx = 160 - 5 = 155`（最初的 5 没有算进去），所以密码应是 [5,155]



3.6 阶段 6 的破解与分析

密码如下：4 3 6 2 1 5

破解过程:

(1) 分析代码，可以将<phase_6>做的事分成三个部分

部分一：

00000000004015f3 <phase_6>:

```

4015f3: 55                push    %rbp
4015f4: 48 89 e5          mov     %rsp,%rbp
4015f7: 41 55            push    %r13
4015f9: 41 54            push    %r12
4015fb: 53              push    %rbx
4015fc: 48 83 ec 58      sub     $0x58,%rsp
401600: 48 8d 75 c0      lea     -0x40(%rbp),%rsi
401604: e8 1d 03 00 00   callq   401926 <read_six_numbers>
401609: 41 bc 00 00 00 00 mov     $0x0,%r12d
40160f: eb 29            jmp     40163a <phase_6+0x47>
401611: e8 ee 02 00 00   callq   401904 <explode_bomb>
401616: eb 37            jmp     40164f <phase_6+0x5c>

```

0×58
 $- 0 \times 40$
 $0 \times 18 \Rightarrow 24(+)$
 正好 6 个 int

```

401618: 83 c3 01        add     $0x1,%ebx
40161b: 83 fb 05        cmp     $0x5,%ebx
40161e: 7f 17          如果 %ebx > 0x5  jg     401637 <phase_6+0x44>
401620: 49 63 c4        movslq %r12d,%rax
401623: 48 63 d3        movslq %ebx,%rdx
401626: 8b 7c 95 c0     mov     -0x40(%rbp,%rdx,4),%edi
40162a: 39 7c 85 c0     cmp     %edi,-0x40(%rbp,%rax,4)
40162e: 75 e8          jne     401618 <phase_6+0x25>
401630: e8 cf 02 00 00   callq   401904 <explode_bomb>
401635: eb e1          jmp     401618 <phase_6+0x25>
401637: 45 89 ec        mov     %r13d,%r12d
40163a: 41 83 fc 05     cmp     $0x5,%r12d
40163e: 7f 19          如果 %r12 > 0x5  jg     401659 <phase_6+0x66>
401640: 49 63 c4        movslq %r12d,%rax
401643: 8b 44 85 c0     mov     -0x40(%rbp,%rax,4),%eax
401647: 83 e8 01        sub     $0x1,%eax
40164a: 83 f8 05        cmp     $0x5,%eax
40164d: 77 c2          如果 %eax > 0x5  ja     401611 <phase_6+0x1e>
40164f: 45 8d 6c 24 01   lea     0x1(%r12),%r13d
401654: 44 89 eb        mov     %r13d,%ebx
401657: eb c2          jmp     40161b <phase_6+0x28>

```

$a[0] \neq a[1]$
 $a[2] \neq a[0]$
 $a[3] \neq a[0]$
 $a[4] \neq a[0]$
 $a[5] \neq a[0]$

$\%ebp + 4 \cdot \%rax - 0x40$
 $\%ebp + 4 \cdot \%rax - 0x40$
 $0 \leq a[i] - 1 \leq 5$
 $\%r12 + 0x1$

通过分析可知输入的是 6 个 int 类型的数，不妨设为 $a[0], a[1], a[2], a[3], a[4], a[5]$ ，这六个数互不相等，而且满足 $1 \leq a[0], a[1], a[2], a[3], a[4], a[5] \leq 6$ ，所以应该是 1, 2, 3, 4, 5, 6 的某个排列。

部分二：

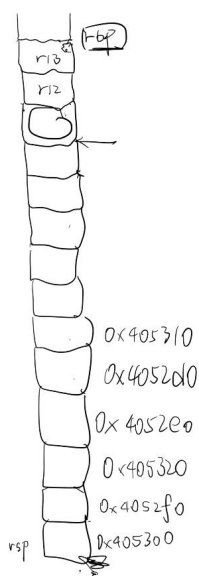
```

401659: be 00 00 00 00    mov     $0x0,%esi    %r12 = 0x6    %esi = 0
40165e: eb 08            jmp     401668 <phase_6+0x75>
401660: 48 89 54 cd 90    mov     %rdx,-0x70(%rbp,%rcx,8)
401665: 83 c6 01          add     $0x1,%esi
401668: 83 fe 05          cmp     $0x5,%esi
40166b: 7f 1c            jg      401689 <phase_6+0x96>    %esi > 0x5
40166d: b8 01 00 00 00    mov     $0x1,%eax    %eax = 1
401672: ba d0 52 40 00    mov     $0x4052d0,%edx    %edx = ?
401677: 48 63 ce          movslq  %esi,%rcx    %rcx = 0
40167a: 39 44 8d c0        cmp     |%eax,-0x40(%rbp,%rcx,4)    a(0)
40167e: 7e e0            jle     401660 <phase_6+0x6d>
401680: 48 8b 52 08        mov     0x8(%rdx),%rdx
401684: 83 c0 01          add     $0x1,%eax
401687: eb ee            jmp     401677 <phase_6+0x84>
401689: 48 8b 5d 90        mov     -0x70(%rbp),%rbx
40168d: 48 89 d9          mov     %rbx,%rcx
401690: b8 01 00 00 00    mov     $0x1,%eax    %eax = 1
401695: eb 12            jmp     4016a9 <phase_6+0xb6>
401697: 48 63 d0          movslq  %eax,%rdx
40169a: 48 8b 54 d5 90    mov     -0x70(%rbp,%rdx,8),%rdx
40169f: 48 89 51 08        mov     %rdx,0x8(%rcx)
4016a3: 83 c0 01          add     $0x1,%eax
4016a6: 48 89 d1          mov     %rdx,%rcx

4016a9: 83 f8 05          cmp     $0x5,%eax
4016ac: 7e e9            jle     401697 <phase_6+0xa4>    %eax < 5
4016ae: 48 c7 41 08 00 00 00    movq    $0x0,0x8(%rcx)
4016b5: 00

```

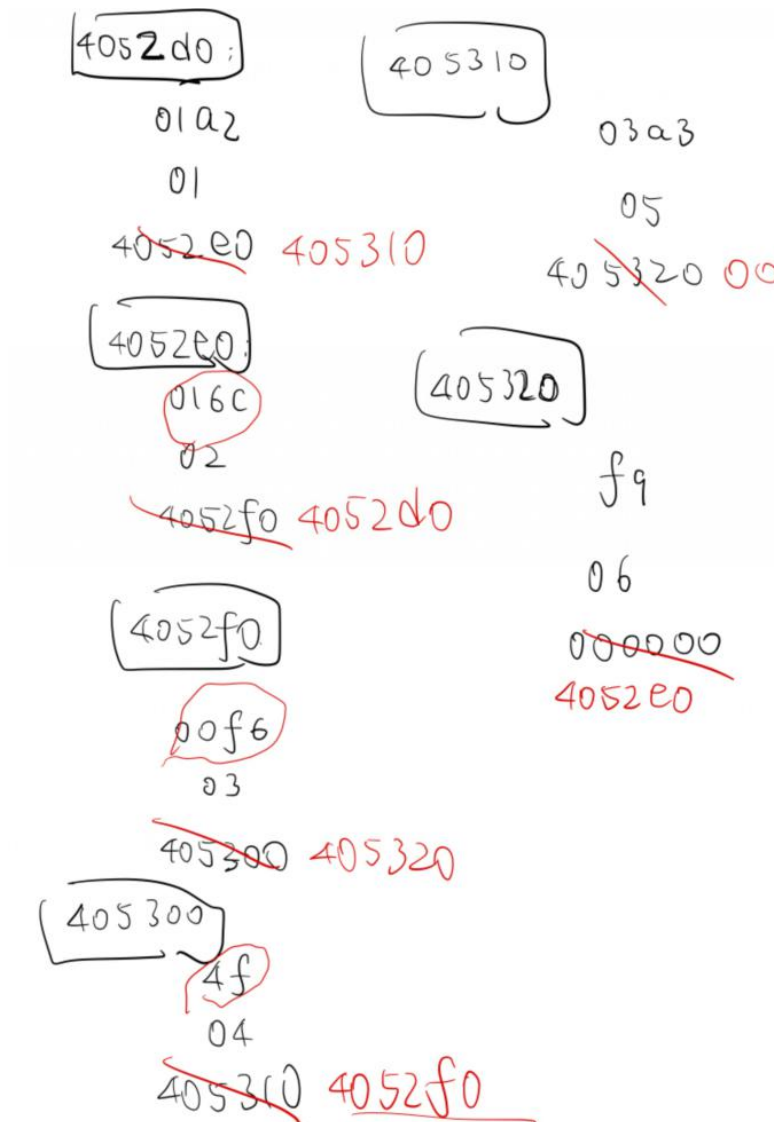
这里一共完成了两个工作，首先将 0x4052d0,0x4052e0,0x4052f0,0x405300,0x405310,0x405320 这 6 个地址存入



这样的结构中，然后可以通过 `objdump` 查看这些地址下存储的内容，如下图所示：

```
Contents of section .data:
4052d0 a2010000 01000000 e0524000 00000000 .....R@.....
4052e0 6c010000 02000000 f0524000 00000000 l.....R@.....
4052f0 f6000000 03000000 00534000 00000000 .....S@.....
405300 4f000000 04000000 10534000 00000000 O.....S@.....
405310 a3030000 05000000 20534000 00000000 .....S@.....
405320 f9000000 06000000 00000000 00000000 .....
```

我们可以看出这里面是类似于链表的形式，链表的每个节点存储两个数据和下一个节点的地址，部分二做的第二件事就是根据输入的数字顺序对链表进行重新链接，如果输入的数据为：4 3 6 2 1 5，链表就会重新排列成下图所示，



部分三:

开始比较新的
链表中的元素
是否按照升序
排序

```

4016b6: 41 bc 00 00 00 00    mov     $0x0,%r12d
4016bc: eb 08                jmp     4016c6 <phase_6+0xd3>
4016be: 48 8b 5b 08          mov     0x8(%rbx),%rbx
4016c2: 41 83 c4 01          add     $0x1,%r12d
4016c6: 41 83 fc 04          cmp     $0x4,%r12d
4016ca: 7f 11                jg      4016dd <phase_6+0xea>
4016cc: 48 8b 43 08          mov     0x8(%rbx),%rax
4016d0: 8b 00                mov     (%rax),%eax
4016d2: 39 03                cmp     %eax,(%rbx)
4016d4: 7e e8                jle     4016be <phase_6+0xcb>
4016d6: e8 29 02 00 00      callq   401904 <explode_bomb>
4016db: eb e1                jmp     4016be <phase_6+0xcb>
4016dd: 48 83 c4 58          add     $0x58,%rsp
4016e1: 5b                  pop     %rbx
4016e2: 41 5c                pop     %r12
4016e4: 41 5d                pop     %r13
4016e6: 5d                  pop     %rbp
4016e7: c3                  retq

```

4016ca: 7f 11 $r12 > 4$ $4016dd$ $4052f0$ $00 \dots f6$ $0x8(\%rbx),\%rax$ $\%eax,(\%rbx)$ $0x4f$ \leq $4016be$ $<phase_6+0xcb>$ $炸$ $校验完毕, ok$ $妙!$

对现有的链表前后结点里的数据进行比较，看是否按照升序排序，如果是，则破解成功，如果不是就爆炸。

(2) 根据升序的要求进行破解

```

Contents of section .data:
4052d0 a2010000 01000000 e0524000 00000000 .....R@.....
4052e0 6c010000 02000000 f0524000 00000000 l.....R@.....
4052f0 f6000000 03000000 00534000 00000000 .....S@.....
405300 4f000000 04000000 10534000 00000000 O.....S@.....
405310 a3030000 05000000 20534000 00000000 ..... S@.....
405320 f9000000 06000000 00000000 00000000 .....

```

因为 $0x4f < 0xf6 < 0xf9 < 0x016c < 0x01a2 < 0x03a3$ ，所以如果要使得重新排列的链表里的数据满足升序条件，那么输入的数据应该是 4 3 6 2 1 5

(3) 验证成功

```
1190200717lh@Graham:~/bomb413$ ./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...
```

打开(O)

ans.txt

~/bomb413

```
1 I am just a renegade hockey mom.
2 0 1 3 6 10 15
3 0 180
4 13 31 DrEvil
5 5 115
6 4 3 6 2 1 5
```

3.7 阶段 7 的破解与分析(隐藏阶段)

密码如下: 1001

破解过程:

(1) 先找到隐藏阶段的入口

```
000000000401a8d <phase_defused>:
401a8d: 83 3d d8 3c 00 00 06 cmpl    $0x6,0x3cd8(%rip)    # 40576c
<num_input_strings>
401a94: 74 01                je     401a97 <phase_defused+0xa>
401a96: c3                  retq   退出
401a97: 55                  push   %rbp
401a98: 48 89 e5            mov    %rsp,%rbp
401a9b: 48 83 ec 60         sub    $0x60,%rsp
401a9f: 4c 8d 45 b0         lea    -0x50(%rbp),%r8
401aa3: 48 8d 4d a8         lea    -0x58(%rbp),%rcx
401aa7: 48 8d 55 ac         lea    -0x54(%rbp),%rdx
401aab: be 79 33 40 00      mov    $0x403379,%esi    0x403379: %edi %edi %edi
401ab0: bf 70 58 40 00      mov    $0x405870,%edi    ""
401ab5: b8 00 00 00 00      mov    $0x0,%eax
401aba: e8 51 f6 ff ff     callq 401110 <__isoc99_sscanf@plt>
401abf: 83 f8 03           cmp    $0x3,%eax
401ac2: 74 0c              je     401ad0 <phase_defused+0x43>
401ac4: bf b8 32 40 00      mov    $0x4032b8,%edi    Congratulations
401ac9: e8 92 f5 ff ff     callq 401060 <puts@plt>
401ace: c9                  leaveq
401acf: c3                  retq
401ad0: be 82 33 40 00      mov    $0x403382,%esi    DrEvil
401ad5: 48 8d 7d b0         lea    -0x50(%rbp),%rdi
401ad9: e8 2a fd ff ff     callq 401808 <strings_not_equal>
401ade: 85 c0              test   %eax,%eax
401ae0: 75 e2              jne    401ac4 <phase_defused+0x37>
401ae2: bf 58 32 40 00      mov    $0x403258,%edi    Curses, find the secret
401ae7: e8 74 f5 ff ff     callq 401060 <puts@plt>
401aec: bf 80 32 40 00      mov    $0x403280,%edi
401af1: e8 6a f5 ff ff     callq 401060 <puts@plt>
401af6: b8 00 00 00 00      mov    $0x0,%eax
401afb: e8 22 fc ff ff     callq 401722 <secret_phase>
401b00: eb c2              jmp    401ac4 <phase_defused+0x37>
```

查找<secret_phase>在代码中出现的位置，我们可以发现<phase_defused>中调用了<secret_phase>，利用 gdb 查看 0x403379 位置下存储的值，如下图所示：

```
Breakpoint 1, main (argc=1, argv=0x7fffffffdf58) at bomb.c:37
37      {
(gdb) x/s 0x403379
0x403379:      "%d %d %s"
(gdb)
```

通过对六个关卡输入的密码格式分析，只有第四个关卡的密码后可以跟一个字符串，查看 0x4032b8 下的内容，

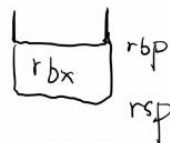
```
(gdb) x/s 0x403382
0x403382:      "DrEvil"
(gdb)
```

所以只要在第四关的密码后面跟上 DrEvil 就能解锁隐藏炸弹

(2) 破解隐藏炸弹

000000000401722 <secret_phase>:

401722:	55	push %rbp	
401723:	48 89 e5	mov %rsp,%rbp	
401726:	53	push %rbx	
401727:	48 83 ec 08	sub \$0x8,%rsp	
40172b:	e8 32 02 00 00	callq 401962 <read_line>	eax 读到了字符串?
401730:	48 89 c7	mov %rax,%rdi	rdi = 字符串?
401733:	e8 08 fa ff ff	callq 401140 <atoi@plt>	将字符串转化为 int, 返回给 eax
401738:	89 c3	mov %eax,%ebx	ebx = eax, 某个 int
40173a:	8d 40 ff	lea -0x1(%rax),%eax	eax = rax - 1
40173d:	3d e8 03 00 00	cmp \$0x3e8,%eax	
401742:	77 27	ja 40176b <secret_phase+0x49>	%eax > 0x3e8 炸
401744:	89 de	mov %ebx,%esi	esi = ebx (某个 int 型)
401746:	bf f0 50 40 00	mov \$0x4050f0,%edi	
40174b:	e8 98 ff ff ff	callq 4016e8 <fun7>	返回值为 eax = 0?
401750:	83 f8 07	cmp \$0x7,%eax	
401753:	75 1d	jne 401772 <secret_phase+0x50>	eax != 0x7 炸
401755:	bf 78 31 40 00	mov \$0x403178,%edi	= 7 打印: 成功破解隐藏阶段
40175a:	e8 01 f9 ff ff	callq 401060 <puts@plt>	
40175f:	e8 29 03 00 00	callq 401a8d <phase_defused>	
401764:	48 83 c4 08	add \$0x8,%rsp	
401768:	5b	pop %rbx	
401769:	5d	pop %rbp	
40176a:	c3	retq	
40176b:	e8 94 01 00 00	callq 401904 <explode_bomb>	炸
401770:	eb d2	jmp 401744 <secret_phase+0x22>	
401772:	e8 8d 01 00 00	callq 401904 <explode_bomb>	炸
401777:	eb dc	jmp 401755 <secret_phase+0x33>	



由上我们可以看出当<fun7>函数的返回值为 0x7 时，成功破解隐藏关卡。初始调用 fun7 函数的两个参数%edi = 0x4050f0，%esi = 某个 int 类型的数

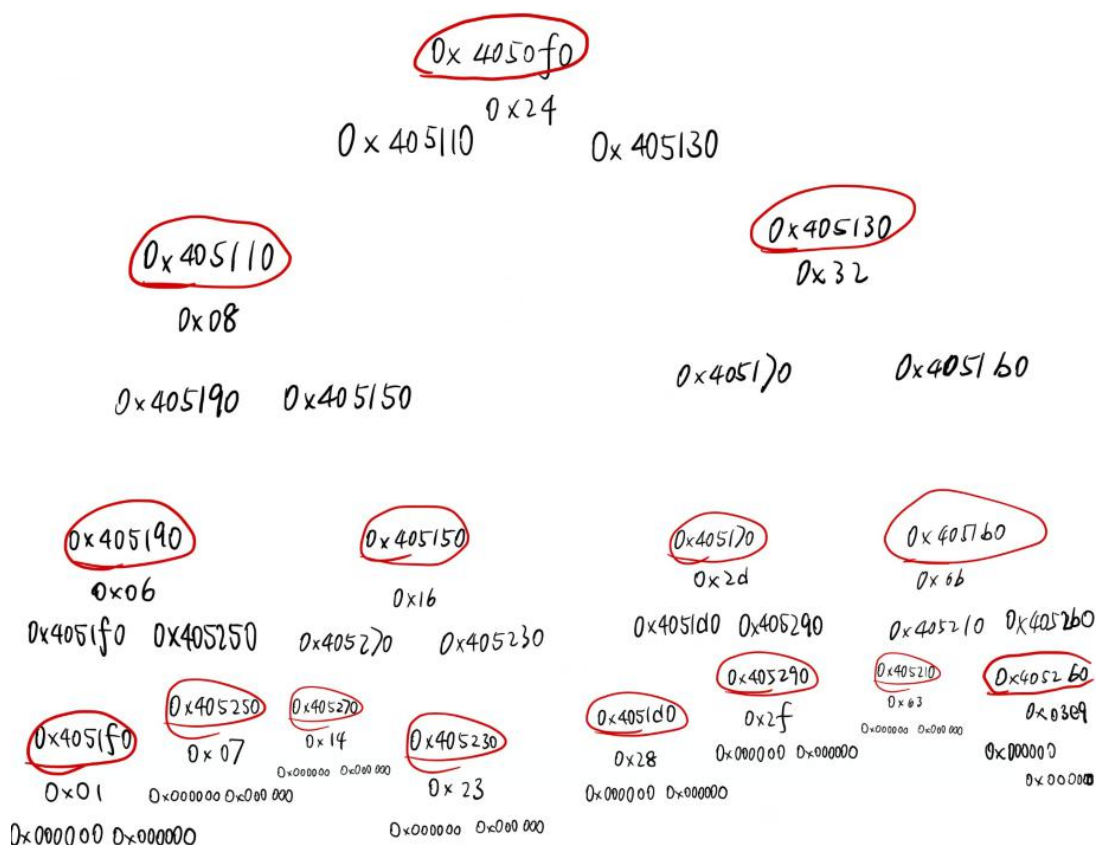
(3) 查看<fun7>的汇编代码

0000000004016e8 <fun7>: edi = 0x4050f0

4016e8: 48 85 ff	test %rdi,%rdi	
je: 2f=13488	je 40171c <fun7+0x34>	
4016eb: 74 2f	push %rbp	
4016ed: 55	mov %rsp,%rbp	
4016ee: 48 89 e5	mov (%rdi),%eax	eax = M(rdi)
4016f1: 8b 07	cmp %esi,%eax	
4016f3: 39 f0	jg 401700 <fun7+0x18>	
4016f5: 7f 09	jne 40170d <fun7+0x25>	
4016f7: 75 14	mov \$0x0,%eax	eax = 0
4016f9: b8 00 00 00 00	pop %rbp	
4016fe: 5d	retq	退出
4016ff: c3		
401700: 48 8b 7f 08	mov 0x8(%rdi),%rdi	rdi = M(rdi+0x8)
401704: e8 df ff ff ff	callq 4016e8 <fun7>	重新执行
401709: 01 c0	add %eax,%eax	
40170b: eb f1	jmp 4016fe <fun7+0x16>	
40170d: 48 8b 7f 10	mov 0x10(%rdi),%rdi	rdi = M(rdi+0x10)
401711: e8 d2 ff ff ff	callq 4016e8 <fun7>	重新执行
401716: 8d 44 00 01	lea 0x1(%rax,%rax,1),%eax	
40171a: eb e2	jmp 4016fe <fun7+0x16>	
40171c: b8 ff ff ff ff	mov \$0xffffffff,%eax	
401721: c3	retq	退出 ?

可以将其化为下面的形式

```
//初始调用fun7函数的两个参数%edi = 0x4050f0
//%esi = 某个int类型的数
fun7(rdi,rsi){
    if(rdi=0){
        return 0xffffffff;
    }
    eax = Memory(rdi);//将rdi的值作为地址，取地址里的内容
    if(eax>esi){
        rdi = Memory(rdi+0x8);
        eax = fun7(rdi,rsi);
        eax = eax + eax;
        return eax;
    }
    if(eax<esi){
        rdi = Memory(rdi+0x10);
        eax = fun7(rdi,rsi);
        eax = 2*eax + 0x1;
        return eax;
    }
    if(eax==esi){
        return 0;
    }
}
```

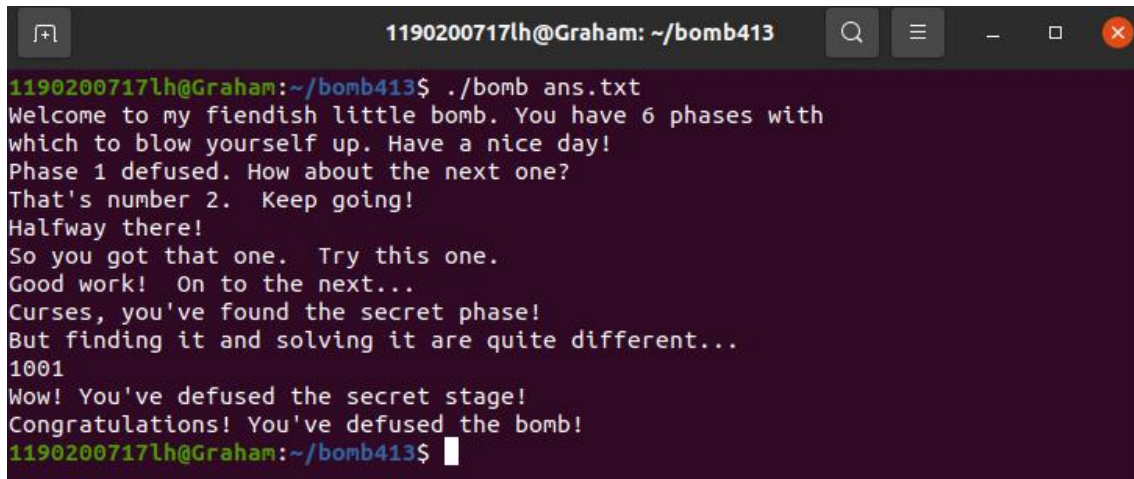



```
1190200717lh@Graham:~/bomb413$ objdump --start-address=0x4050f0 -s bomb
bomb:      文件格式 elf64-x86-64

Contents of section .data:
4050f0 24000000 00000000 10514000 00000000 $.Q@.....
405100 30514000 00000000 00000000 00000000 @Q@.....
405110 08000000 00000000 90514000 00000000 .....Q@.....
405120 50514000 00000000 00000000 00000000 PQ@.....
405130 32000000 00000000 70514000 00000000 2.....pQ@.....
405140 b0514000 00000000 00000000 00000000 .Q@.....
405150 16000000 00000000 70524000 00000000 .....pR@.....
405160 30524000 00000000 00000000 00000000 @R@.....
405170 2d000000 00000000 d0514000 00000000 ~.....Q@.....
405180 90524000 00000000 00000000 00000000 .R@.....
405190 06000000 00000000 f0514000 00000000 .....Q@.....
4051a0 50524000 00000000 00000000 00000000 PR@.....
4051b0 6b000000 00000000 10524000 00000000 k.....R@.....
4051c0 b0524000 00000000 00000000 00000000 .R@.....
4051d0 28000000 00000000 00000000 00000000 (.Q@.....
4051e0 00000000 00000000 00000000 00000000 .....
4051f0 01000000 00000000 00000000 00000000 .....
405200 00000000 00000000 00000000 00000000 .....
405210 63000000 00000000 00000000 00000000 C.....
405220 00000000 00000000 00000000 00000000 .....
405230 23000000 00000000 00000000 00000000 #.....
405240 00000000 00000000 00000000 00000000 .....
405250 07000000 00000000 00000000 00000000 .....
405260 00000000 00000000 00000000 00000000 .....
405270 14000000 00000000 00000000 00000000 .....
405280 00000000 00000000 00000000 00000000 .....
405290 2f000000 00000000 00000000 00000000 /.....
4052a0 00000000 00000000 00000000 00000000 .....
4052b0 e9030000 00000000 00000000 00000000 .....
4052c0 00000000 00000000 00000000 00000000 .....
```

可以看出来这是一种类似于二叉树的结构，如果 $\text{Memory}(\text{rdi}) > \text{esi}$ 就走左支，如果 $\text{Memory}(\text{rdi}) < \text{esi}$ 就走右支，要想使得 `fun7` 最终的返回值为 `0x7`，那么 `rsi = 0x03e9`（十进制表示为：1001）

（4）验证成功



```
1190200717lh@Graham: ~/bomb413
1190200717lh@Graham:~/bomb413$ ./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...
1001
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
1190200717lh@Graham:~/bomb413$
```

第 4 章 总结

4.1 请总结本次实验的收获

通过本次实验我更加熟悉了 gdb 的调试，学会了通过分析反汇编代码获悉原函数的实现方式，学会了 switch、递归等结构的实现方式，同时对于各种指令也更加熟练，明白了汇编语言中各个函数之间的调用关系，对堆栈有了更深的理解，对上课所学的内容领悟的更加深刻了。

4.2 请给出对本次实验内容的建议

希望 PPT 对于实验要求可以再详细一些

注：本章为酌情加分项。

参考文献

- [1] 深入理解计算机系统