

# 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 计算机类

学 号 1190201215

班 级 1903007

学 生 冯开来

指 导 教 师 吴锐

实 验 地 点 G709

实 验 日 期 2021.4.19

## 计算机科学与技术学院

# 目 录

<b>第 1 章 实验基本信息 .....</b>	<b>- 3 -</b>
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 -
<b>第 2 章 实验环境建立 .....</b>	<b>- 5 -</b>
2.1 UBUNTU 下 CODEBLOCKS 反汇编（10 分） .....	- 5 -
2.2 UBUNTU 下 EDB 运行环境建立（10 分） .....	- 6 -
<b>第 3 章 各阶段炸弹破解与分析 .....</b>	<b>- 7 -</b>
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.....	- 13 -
3.5 阶段 5 的破解与分析.....	- 17 -
3.6 阶段 6 的破解与分析.....	- 20 -
3.7 阶段 7 的破解与分析(隐藏阶段).....	- 23 -
<b>第 4 章 总结 .....</b>	<b>- 27 -</b>
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 -
<b>参考文献 .....</b>	<b>- 28 -</b>

## 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

#### 1.2.3 开发工具

GDB/OBJDUMP; EDB; KDD 等

### 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。

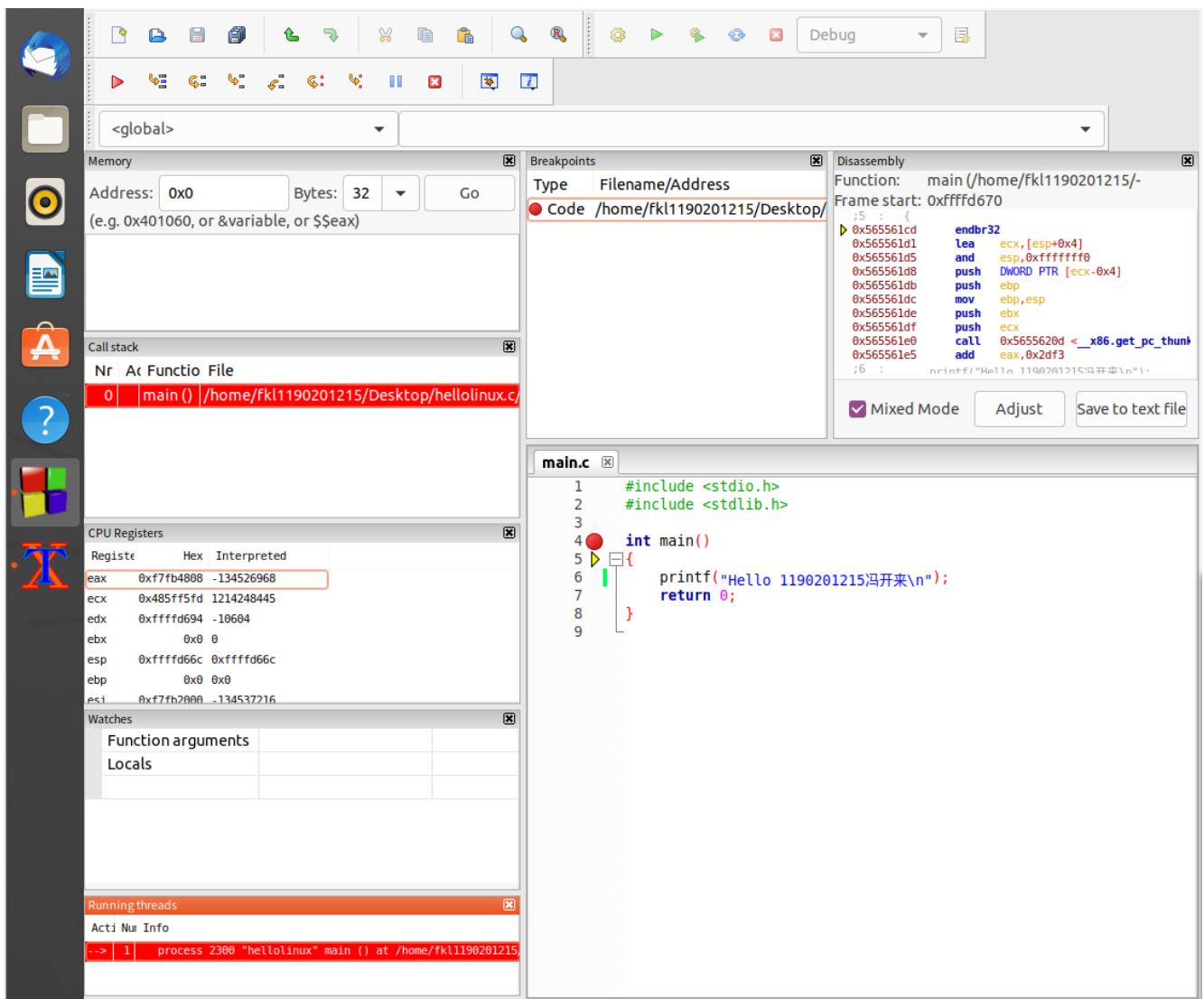


图 2-1 Ubuntu 下 CodeBlocks 反汇编截图

## 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

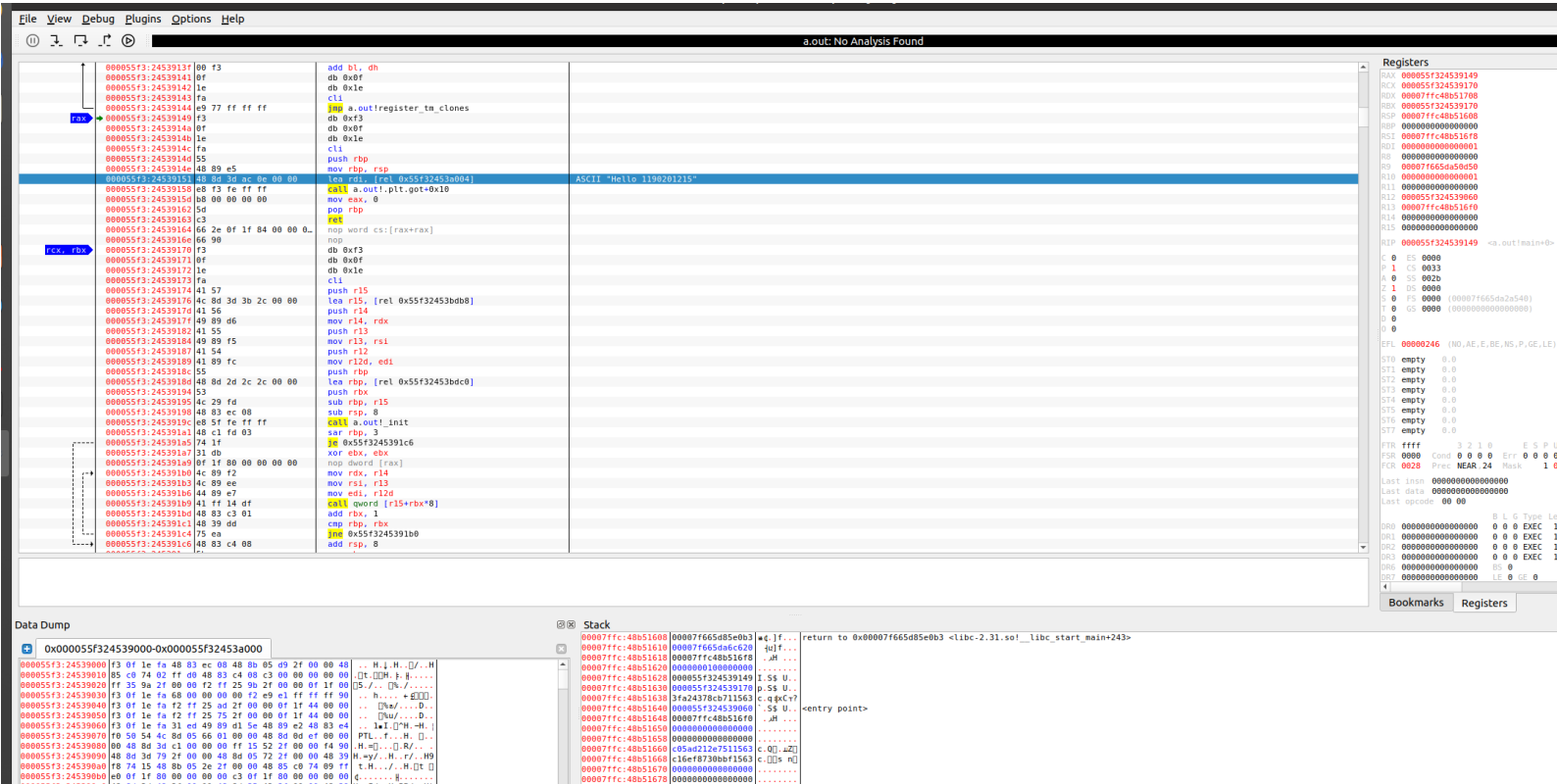


图 2-2 Ubuntu 下 EDB 截图

## 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：Wow! Brazil is big.

破解过程：

1020					
1021	000000000004013f9	<phase_1>:			
1022	4013f9:	55	将地址 0x40314c 送入 %esi, 对 (%esi) 寻址, 应该是一个字符串	push %rbp	比较两个字符串是否相同, 相同返回 0, 当 %eax != 0, 跳转爆炸, %eax=0, 弹栈返回
1023	4013fa:	48 89 e5		mov %rsp,%rbp	
1024	4013fd:	be 4c 31 40 00		mov \$0x40314c,%esi	
1025	401402:	e8 e8 04 00 00		callq 4018ef <strings_not_equal>	
1026	401407:	85 c0		test %eax,%eax	
1027	401409:	75 02		jne 40140d <phase_1+0x14>	
1028	40140b:	5d		pop %rbp	
1029	40140c:	c3		retq	
1030	40140d:	e8 d9 05 00 00		callq 4019eb <explode_bomb>	
1031	401412:	eb f7		jmp 40140b <phase_1+0x12>	
1032					

通过反汇编代码可以看到，在压栈等基本操作之后，系统调用了一个 `string_not_equal` 的函数，根据名字和具体代码可以知道，这个函数的作用是比较两个字符串是否相等，若不相等，返回 1，若相等，返回 0，如果返回 0，通过 `test` 和 `jne` 这两个判断以及跳转条件实现弹栈并顺利通过该函数，否则爆炸。所以 `phase_1` 的主要目的就是判断两个字符串是否相等。这里 `string_not_equal` 比较的两个参数一个是 `%esi` 一个是我们输入的，而 `%esi` 中的字符串可以通过查找 `0x40314c` 处得知。

```

fkl1190201215@ubuntu: ~
type apropos word to search for commands related to word ...
Reading symbols from bomb...
(gdb) b *0x40314c
Breakpoint 1 at 0x40314c
(gdb) r
Starting program: /home/fkl1190201215/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
abcde

BOOM!!!
The bomb has blown up.
[Inferior 1 (process 2955) exited with code 010]
(gdb) x/s 0x40314c
0x40314c: "Wow! Brazil is big."
  
```

为了查看 0x40314c 地址内容，在 0x40314c 处设置断点

以字符串格式查看，得到 0x40314c 内容，即需要匹配的字符串



我们通过在 0x40314c 处设置断点，可以并用 gdb 调试以及使用 x/s 查看可以得到字符串，即我们需要的答案。

```

fkl1190201215@ubuntu: ~
fkl1190201215@ubuntu:~$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Wow! Brazil is big.
Phase 1 defused. How about the next one?
  
```

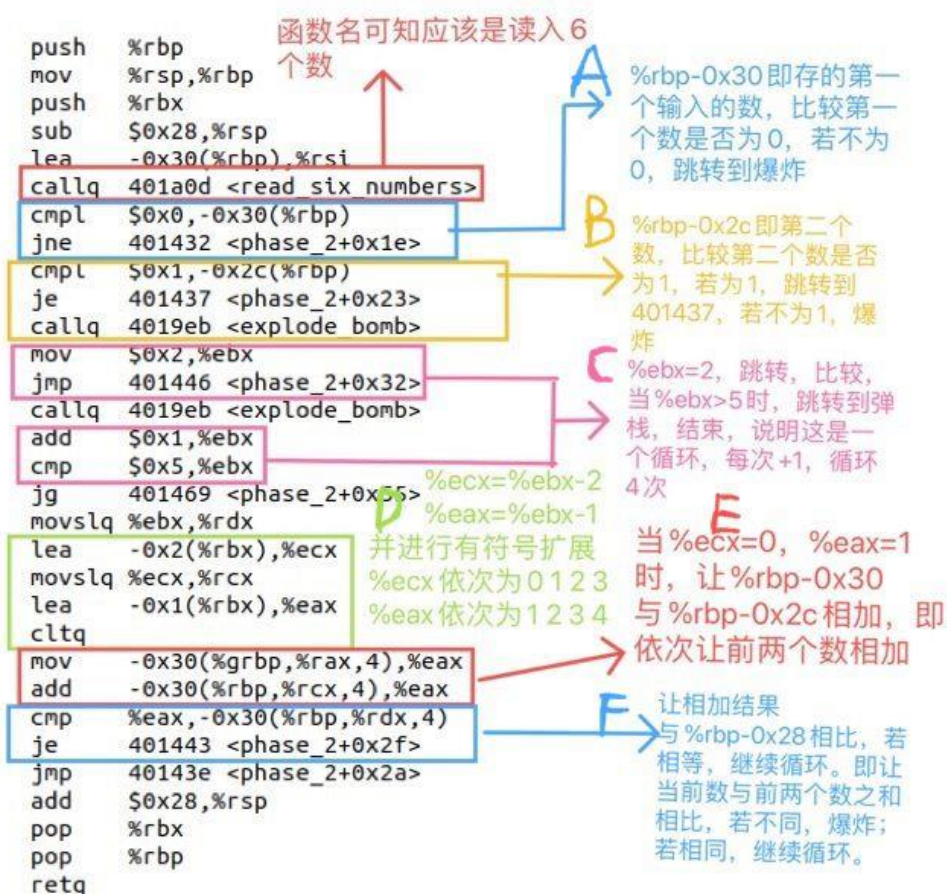
### 3.2 阶段 2 的破解与分析

密码如下：0 1 1 2 3 5

破解过程：

```

1033 0000000000401414 <phase_2>:
1034 401414: 55
1035 401415: 48 89 e5
1036 401418: 53
1037 401419: 48 83 ec 28
1038 40141d: 48 8d 75 d0
1039 401421: e8 e7 05 00 00
1040 401426: 83 7d d0 00
1041 40142a: 75 06
1042 40142c: 83 7d d4 01
1043 401430: 74 05
1044 401432: e8 b4 05 00 00
1045 401437: bb 02 00 00 00
1046 40143c: eb 08
1047 40143e: e8 a8 05 00 00
1048 401443: 83 c3 01
1049 401446: 83 fb 05
1050 401449: 7f 1e
1051 40144b: 48 63 d3
1052 40144e: 8d 4b fe
1053 401451: 48 63 c9
1054 401454: 8d 43 ff
1055 401457: 48 98
1056 401459: 8b 44 85 d0
1057 40145d: 03 44 8d d0
1058 401461: 39 44 95 d0
1059 401465: 74 dc
1060 401467: eb d5
1061 401469: 48 83 c4 28
1062 40146d: 5b
1063 40146e: 5d
1064 40146f: c3
1065
  
```



通过反汇编代码，先调用了一个 read\_six\_numbers 的函数，根据名字我们知道这应该是读入 6 个整型数，但是输入格式我们不知道，稍后我们会查看 read\_six\_numbers 这个函数得到输入格式。继续阅读反汇编代码，在输入之后，



先进行了两个比较，分别是  $-0x30(\%rbp)$  和 0 比较、 $-0x2c(\%rbp)$  和 1 比较。其实后续通过阅读 `read_six_numbers` 可以知道我们输入的是 6 个 `int` 型，每个占 4 个字节，所以  $-0x30$  和  $-0x2c$  正好差了 4 个字节，所以我们可以判定这两个数就是我们输入的第一个和第二个数，如果不等于 0、1，那么会跳转到爆炸，所以我们输入的第一个第二个数就是 0、1。继续阅读反汇编代码，他让  $\%ebx=2$ ，并且让  $\%ebx$  与 5 比较，在比较上面还有一个  $\%ebx++$ ，所以可以初步判定这是一个循环，当  $\%ebx \leq 5$  的时候，进入循环主体，如图片中所写，循环的主体就是判断前两个数相加是否等于下一个数，其中的索引就是通过循环变量  $\%ebx$ ， $\%ecx (\%ebx-2)$ ， $\%eax (\%ebx-1)$  来找到相应的数。因为第一第二个数是 0，1，我们很容易得到第三个数为 1，依次为 2，3，5。

然后我们来看一下 `read_six_numbers` 这个函数：

```
0000000000401a0d <read_six_numbers>:
401a0d: 55          push  %rbp
401a0e: 48 89 e5    mov   %rsp,%rbp
401a11: 48 89 f2    mov   %rsi,%rdx
401a14: 48 8d 4e 04 lea    0x4(%rsi),%rcx
401a18: 48 8d 46 14 lea    0x14(%rsi),%rax
401a1c: 50          push  %rax
401a1d: 48 8d 46 10 lea    0x10(%rsi),%rax
401a21: 50          push  %rax
401a22: 4c 8d 4e 0c lea    0xc(%rsi),%r9
401a26: 4c 8d 46 08 lea    0x8(%rsi),%r8
401a2a: be eb 32 40 00 mov    $0x4032eb,%esi
401a2f: b8 00 00 00 00 mov    $0x0,%eax
401a34: e8 d7 f6 ff ff callq  401110 <__isoc99_sscanf@plt>
401a39: 48 83 c4 10 add    $0x10,%rsp
401a3d: 83 f8 05    cmp    $0x5,%eax
401a40: 7e 02      jle    401a44 <read_six_numbers+0x37>
401a42: c9          leaveq
401a43: c3          retq
401a44: e8 a2 ff ff ff callq  4019eb <explode_bomb>
```

将地址 0x4032eb 传入 %esi，根据下面 scanf，可以猜测这是输入格式

```

fkl1190201215@ubuntu: ~
Reading symbols from bomb...
(gdb) b *0x4032eb
Breakpoint 1 at 0x4032eb
(gdb) r
Starting program: /home/fkl1190201215/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Wow! Brazil is big.
Phase 1 defused. How about the next one?
1234567

BOOM!!!
The bomb has blown up.
[Inferior 1 (process 2358) exited with code 010]
(gdb) x/s 0x4032eb
0x4032eb:      "%d %d %d %d %d %d"
(gdb) q
fkl1190201215@ubuntu:~$ gdb bomb
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

```

查看 0x4032eb 地址，以字符串显示，即输入格式，将6个整型数空格隔开输入

可以看到，我们6个数据的输入格式为%d 并且以空格隔开。所以我们的答案为  
0 1 1 2 3 5

```

fkl1190201215@ubuntu: ~
fkl1190201215@ubuntu:~$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Wow! Brazil is big.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2. Keep going!

```

### 3.3 阶段3的破解与分析

密码如下：0 f 556

破解过程：

<pre> 1066 00000000000401470 &lt;phase_3&gt;: 1067 401470:      55 1068 401471:      48 89 e5 1069 401474:      48 83 ec 10 1070 401478:      4c 8d 45 f8 1071 40147c:      48 8d 4d f7 1072 401480:      48 8d 55 fc 1073 401484:      be 60 31 40 00 1074 401489:      b8 00 00 00 00 </pre>	<pre> push    %rbp mov     %rsp,%rbp sub     \$0x10,%rsp lea     -0x8(%rbp),%r8 lea     -0x9(%rbp),%rcx lea     -0x4(%rbp),%rdx mov     \$0x403160,%esi mov     \$0x0,%eax </pre>	<p>根据字节数，猜测输入为两个int一个char</p> <p>根据上一题经验，这地址存的应该是输入格式</p>
--	---	---

首先可以看到三个加载有效地址，这应该是准备输入三个变量，根据前面几题经验，0x4030160中应该是输入格式。通过设置断点并且x/s查看可以知道输入格式为%d %c %d。即 int char int。

```

Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...
(gdb) b *0x403160
Breakpoint 1 at 0x403160
(gdb) r
Starting program: /home/fkl1190201215/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Wow! Brazil is big.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2. Keep going!
abcdefg

BOOM!!!
The bomb has blown up.
[Inferior 1 (process 2375) exited with code 610]
(gdb) x/s 0x403160
0x403160: "%d %c %d"
(gdb) b
No default breakpoint address now.
(gdb) q

```

在 0x403160 设置断点，与第二题相同，用 x/s 查看输入格式为 %d %c %d

之后我们继续回到反汇编代码：

1075	40148e:	e8 7d fc ff ff	callq 401110 <__isoc99_sscanf@plt>	
1076	401493:	83 f8 02	cmp \$0x2,%eax	
1077	401496:	7e 15	jle 4014ad <phase_3+0x3d>	
1078	401498:	8b 45 fc	mov -0x4(%rbp),%eax	
1079	40149b:	83 f8 07	cmp \$0x7,%eax	
1080	40149e:	0f 87 02 01 00 00	ja 4015a6 <phase_3+0x136>	显然，第一个数是和7进行比较，若比7大，则会跳转爆炸
1081	4014a4:	89 c0	mov %eax,%eax	
1082	4014a6:	ff 24 c5 70 31 40 00	jmpq *0x403170(,%rax,8) //跳转表开始	
1083	4014ad:	e8 39 05 00 00	callq 4019eb <explode_bomb>	
1084	4014b2:	eb e4	jmp 401498 <phase_3+0x28>	
1085	4014b4:	81 7d f8 2c 02 00 00	cmpl \$0x22c,-0x8(%rbp)	
1086	4014bb:	75 0a	jne 4014c7 <phase_3+0x57>	
1087	4014bd:	b8 66 00 00 00	mov \$0x66,%eax	
1088	4014c2:	e9 e9 00 00 00	jmpq 4015b0 <phase_3+0x140>	
1089	4014c7:	e8 1f 05 00 00	callq 4019eb <explode_bomb>	
1090	4014cc:	b8 66 00 00 00	mov \$0x66,%eax	
1091	4014d1:	e9 da 00 00 00	jmpq 4015b0 <phase_3+0x140>	
1092	4014d6:	81 7d f8 a3 00 00 00	cmpl \$0xa3,-0x8(%rbp)	

跳转表开始，我们可以通过 gdb 调试，查看具体跳转到什么地方，可以肯定的是第一个数不能大于 7

图中已经表述，将  $-0x4(\%rbp)$  送至  $\%eax$ ，并与  $0x7$  比较，如果  $\%eax > 7$ ，则跳转到爆炸，这里面根据上面加载有效地址可以知道  $-0x4(\%rbp)$  是我们输入的第一个数，所以我们输入的第一个数  $\leq 7$ 。随后将  $\%eax$  送至  $\%eax$ ，其实就是一个扩展。然后是无条件跳转  $\text{jmpq}$ ，根据后面这个  $*0x403170(, \%rax, 8)$ ，猜测这应该是  $\text{switch}$  的跳转表。因为我们输入的第一个  $\text{int}$  不会  $> 7$ ，所以我们可以先尝试一下第一个输入为  $0$ ，然后通过  $\text{gdb}$  的逐步调试查看当输入  $0$  的时候，我们会跳转到什么地方。我们将断点设置在跳转表之前（即将跳转）。



```

fkl1190201215@ubuntu: ~
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...
(gdb) b *0x4014a4
Breakpoint 1 at 0x4014a4: file phases.c, line 75.
(gdb) r
Starting program: /home/fkl1190201215/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Wow! Brazil is big.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2. Keep going!
0 a 123

Breakpoint 1, 0x00000000004014a4 in phase_3 (input=<optimized out>) at phases.c:75
phases.c: 没有那个文件或目录.
(gdb) si
0x00000000004014a6    75    in phases.c
(gdb) si
78    in phases.c
(gdb) si
0x00000000004014bb    78    in phases.c
(gdb) q
A debugging session is active.

Inferior 1 [process 2461] will be killed.

```

在在进入跳转表之前设置断点，然后逐步运行程序

即将进入跳转表

输入0后，跳转的地址

我们先随便输入 0 a 123，随后逐步调试，查看会跳转到哪个地址，根据上图即注释，我们发现输入 0 的结果是跳转到 0x4014bb。我们继续在反汇编代码中找到 0x4014bb，根据前后上下文判断另外输入的两个数。

1078	401498:	8b 45 fc	mov	-0x4(%rbp),%eax	
1079	40149b:	83 f8 07	cmp	\$0x7,%eax	
1080	40149e:	0f 87 02 01 00 00	ja	4015a6 <phase_3+0x136>	
1081	4014a4:	89 c0	mov	%eax,%eax	
1082	4014a6:	ff 24 c5 70 31 40 00	jmpq	*0x403170(,%rax,8) //跳转表开始	
1083	4014ad:	e8 39 05 00 00	callq	4019eb <explode_bomb>	
1084	4014b2:	eb e4	jmp	401498 <phase_3+0x28>	
1085	4014b4:	81 7d f8 2c 02 00 00	cmpl	\$0x22c,-0x8(%rbp)	输入0后，跳转的地址 4014bb
1086	4014bb:	75 0a	jne	4014c7 <phase_3+0x57>	
1087	4014bd:	b8 66 00 00 00	mov	\$0x66,%eax	
1088	4014c2:	e9 e9 00 00 00	jmpq	4015b0 <phase_3+0x140>	
1089	4014c7:	e8 1f 05 00 00	callq	4019eb <explode_bomb>	
1090	4014cc:	b8 66 00 00 00	mov	\$0x66,%eax	查看该地址附近的编码，上一行是-0x(%rbp)与0x22c比较，即第二个int与0x22c比较，若不相同，跳转爆炸。然后将0x66赋值给%ebx，并且跳转到4015b0
1091	4014d1:	e9 da 00 00 00	jmpq	4015b0 <phase_3+0x140>	
1092	4014d6:	81 7d f8 a3 00 00 00	cmpl	\$0xa3,-0x8(%rbp)	
1093	4014dd:	75 0a	jne	4014e9 <phase_3+0x79>	
1094	4014df:	b8 6a 00 00 00	mov	\$0x6a,%eax	
1095	4014e4:	e9 c7 00 00 00	jmpq	4015b0 <phase_3+0x140>	
1096	4014e9:	e8 fd 04 00 00	callq	4019eb <explode_bomb>	
1097	4014ee:	b8 6a 00 00 00	mov	\$0x6a,%eax	
1098	4014f3:	e9 b8 00 00 00	jmpq	4015b0 <phase_3+0x140>	
1099	4014f8:	83 7d f8 69	cmpl	\$0x69,-0x8(%rbp)	
1100	4014fc:	75 0a	jne	401508 <phase_3+0x98>	
1101	4014fe:	b8 6d 00 00 00	mov	\$0x6d,%eax	

可以看到，在 0x4014bb 前面的语句中，先比较了  $-0x8(\%rbp)$  与 0x22c 的大小，若不相同则跳转到 0x4014c7 即爆炸， $-0x8(\%rbp)$  是我们输入的第二个 int，所以当我们输入第一个 int 为 0 时，第二个 int 应该为 0x22c 即 556。

然后汇编代码中将 0x66 送至  $\%eax$ ，并跳转到 0x4015b0。

1141	401580:	e8 40 04 00 00	callq 401580 <explode_bomb>	
1142	4015ab:	b8 79 00 00 00	mov \$0x79,%eax	
1143	4015b0:	38 45 f7	cmp %al,-0x9(%rbp)	跳转后，将 %al 与 -0x9(%rbp) 比较，即我们输入的 char 与 0x66 比较，若不相同则爆炸，ascii 码 0x66 对应的是 f，所以我们输入的 char 为 f
1144	4015b3:	75 02	jne 4015b7 <phase_3+0x147>	
1145	4015b5:	c9	leaveq	
1146	4015b6:	c3	retq	
1147	4015b7:	e8 2f 04 00 00	callq 4015eb <explode_bomb>	
1148	4015bc:	eb f7	jmp 4015b5 <phase_3+0x145>	
1149				

跳转后，将  $\%eax$  与  $-0x9(\%rbp)$  比较，即我们输入的 char 与 0x66 比较，如果不相等则跳转到爆炸。那么应该可以知道这里 0x66 时 ascii 码，即对应的 f。所以我们输入的 char 为 f。

当然，我们第一个输入不止可以为 0，这个阶段的答案不唯一。

所以根据我们输入格式可以知道我们 phase\_3 答案为 0 f 556。

```

fkl1190201215@ubuntu: ~
fkl1190201215@ubuntu:~$ ./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!
0 f 556
Halfway there!
  
```

### 3.4 阶段 4 的破解与分析

密码如下：77 （隐藏阶段输入需要输入 77 DrEvil）

破解过程：

1179	00000000004015fd	<phase_4>:		
1180	4015fd:	55	push %rbp	
1181	4015fe:	48 89 e5	mov %rsp,%rbp	
1182	401601:	48 83 ec 10	sub \$0x10,%rsp	
1183	401605:	48 8d 4d f8	lea -0x8(%rbp),%rcx	根据前一题经验，这里输入格式应该是两个 int，并且可以查看地址 0x4032f7 判断输入格式
1184	401609:	48 8d 55 fc	lea -0x4(%rbp),%rdx	
1185	40160d:	be f7 32 40 00	mov \$0x4032f7,%esi	
1186	401612:	b8 00 00 00 00	mov \$0x0,%eax	
1187	401617:	e8 f4 fa ff ff	callq 401110 <__isoc99_sscanf@plt>	
1188	40161c:	83 f8 02	cmp \$0x2,%eax	
1189	40161f:	75 0c	jne 40162d <phase_4+0x30>	



根据前面几题经验以及这里两个压栈，可以知道这里输入格式是两个 int，并且可以通过查看 0x4032f7 确定输入格式。

```

Reading symbols from bomb...
(gdb) b *0x4032f7
Breakpoint 1 at 0x4032f7
(gdb) r
Starting program: /home/fkl1190201215/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Wow! Brazil is big.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
BOOM!!!
The bomb has blown up.
[Inferior 1 (process 8841) exited with code 010]
(gdb) x/s 0x4032f7
0x4032f7:      "%d %d"
(gdb)

```

在 0x4032f7 设置断点，用 gdb 调试，并用 x/s 查看，得到 %d %d，说明是输入两个 int

得到了输入格式，我们继续看反汇编代码。

```

1179 00000000004015fd <phase_4>:
1180 4015fd:      55                push    %rbp
1181 4015fe:      48 89 e5          mov     %rsp,%rbp
1182 401601:      48 83 ec 10       sub     $0x10,%rsp
1183 401605:      48 8d 4d f8       lea     -0x8(%rbp),%rcx
1184 401609:      48 8d 55 fc       lea     -0x4(%rbp),%rdx
1185 40160d:      be f7 32 40 00    mov     $0x4032f7,%esi
1186 401612:      b8 00 00 00 00    mov     $0x0,%eax
1187 401617:      e8 f4 fa ff ff   callq   401110 <__isoc99_sscanf@plt>
1188 40161c:      83 f8 02         cmp     $0x2,%eax
1189 40161f:      75 0c            jne     40162d <phase_4+0x30>
1190 401621:      8b 45 fc         mov     -0x4(%rbp),%eax
1191 401624:      85 c0            test    %eax,%eax
1192 401626:      78 05            js      40162d <phase_4+0x30>
1193 401628:      83 f8 0e         cmp     $0xe,%eax
1194 40162b:      7e 05            jle     401632 <phase_4+0x35>
1195 40162d:      e8 b9 03 00 00    callq   4019eb <explode_bomb>
1196 401632:      ba 0e 00 00 00    mov     $0xe,%edx
1197 401637:      be 00 00 00 00    mov     $0x0,%esi
1198 40163c:      8b 7d fc         mov     -0x4(%rbp),%edi
1199 40163f:      e8 7a ff ff ff   callq   4015be <func4>
1200 401644:      83 f8 07         cmp     $0x7,%eax
1201 401647:      75 06            jne     40164f <phase_4+0x52>
1202 401649:      83 7d f8 07       cmpl    $0x7,-0x8(%rbp)
1203 40164d:      74 05            je      401654 <phase_4+0x57>
1204 40164f:      e8 97 03 00 00    callq   4019eb <explode_bomb>
1205 401654:      c9              leaveq  %eax,%edi
1206 401655:      c3              retq

```

%eax=-0x4(%rbp)，即 %eax 现存第一个输入数。test %eax, %eax，如果 %eax 为负数，跳转到爆炸，如果大于 14，也爆炸，所以第一个输入数为 0 到 14。

%edx=14, %esi=0, %edi 为第一输入数，随机调用函数 func4。依据规定，%rdi, %rsi, %rdx 分别存传入的第一、二、三个参数

当调用函数返回值不为 7，则爆炸

第一个指令是将 -0x4(%rbp) 送至 %eax，并检验 %eax 是否为负数，如果 %eax 即 -0x4(%rbp) 为负数，那么则会条件跳转到 0x40162d，炸弹爆炸，再将 %eax 与 0xe 比较，如果 %eax > 0xe，则会爆炸，所以我们输入的第一个 int 范围是 0 到 14。

第二段是将立即数 0xe 送至%edx, 0x0 送至%esi, -0x4(%rbp)送至%edi, 我们知道在计算机中%edi, %esi, %edx 分别存调用函数的第一、二、三个参数, 在下面很快调用了 func4 函数, 且第一、二、三个参数分别是-0x4(%rbp), 0, 14。

调用函数之后, 是将返回值与 7 进行比较, 如果不相同, 则跳转到爆炸, 相同则继续下面指令。我们先看函数 func4。

1150 00000000004015be <func4>:			
1151 4015be:	55	push %rbp	初始化, %eax=%edx, %ebx=%eax, 我们第一个输入的参数
1152 4015bf:	48 89 e5	mov %rsp,%rbp	
1153 4015c2:	53	push %rbx	
1154 4015c3:	48 83 ec 08	sub \$0x8,%rsp	
1155 4015c7:	89 d0	mov %edx,%eax	
1156 4015c9:	29 f0	sub %esi,%eax	先将 %ebx 逻辑右移 0x1f 即 31 位, 因为 int 是 32 位, 所以只剩 1 个符号位, 若输入为正数, 则 %ebx=0, 若输入为负数, 则 %ebx=1, 并加上 %eax, 其实本质就是除法中的加上偏置
1157 4015cb:	89 c3	mov %eax,%ebx	
1158 4015cd:	c1 eb 1f	shr \$0x1f,%ebx	sar 默认算数右移一位, 即 %ebx 输入的数除以 2, 并将他与 %edi 比较, 如果等于, 则返回 %ebx 即输入的数, 若不等于, 分别通过大于小于跳转
1159 4015d0:	01 c3	add %eax,%ebx	
1160 4015d2:	d1 fb	sar %ebx	若 %ebx > %edi, 则 %edx = %rbx - 1, 并将 %edi, %esi, %edx 继续调用 func4
1161 4015d4:	01 f3	add %esi,%ebx	
1162 4015d6:	39 fb	cmp %edi,%ebx	若 %ebx < %edi, 则 %esi = %rbx + 1, 并将 %edi, %esi, %edx 继续调用 func4
1163 4015d8:	7f 0b	jg 4015e5 <func4+0x27>	
1164 4015da:	7c 15	jle 4015f1 <func4+0x33>	
1165 4015dc:	89 d8	mov %ebx,%eax	
1166 4015de:	48 83 c4 08	add \$0x8,%rsp	
1167 4015e2:	5b	pop %rbx	
1168 4015e3:	5d	pop %rbp	
1169 4015e4:	c3	retq	
1170 4015e5:	8d 53 ff	lea -0x1(%rbx),%edx	
1171 4015e8:	e8 d1 ff ff ff	callq 4015be <func4>	
1172 4015ed:	01 c3	add %eax,%ebx	
1173 4015ef:	eb eb	jmp 4015dc <func4+0x1e>	
1174 4015f1:	8d 73 01	lea 0x1(%rbx),%esi	
1175 4015f4:	e8 c5 ff ff ff	callq 4015be <func4>	
1176 4015f9:	01 c3	add %eax,%ebx	
1177 4015fb:	eb df	jmp 4015dc <func4+0x1e>	

func4 函数如上图所示。

一开始还是初始化的压栈和赋值操作。然后是 shr \$0x1f %ebx。即将 %ebx 逻辑右移 31 为, 因为这里面 int 是 32 位, 所以移位后也只剩下一个符号位, 随后将 %eax 加上这个移位的结果, 如果 %eax < 0, 则加一, 若 %eax ≥ 0, 则不变, 然后将最新的结果算数右移 1 位。整体来看, 之前的加 1 或者加 0 实质上是加上一个偏置然后再除以 2, 因为简单的移位操作会使负数的结果向负无穷舍入, 所以需要加偏置向 0 舍入。所以本段汇编代码的意思就是 (%edx - %esi) / 2, 送至 %edx, 然后再将结果与 %edi (我们输入的数) 进行比较。如果相等, 则返回 %ebx, 如果 %ebx > %edi, 返回 func4 (%edi, %esi, %ebx - 1),

如果 %ebx < %edi, 返回 func4 (%edi, %ebx - 1, %edx), 这是一个递归函数, 我们可以编写一个 c 语言程序, 输入为 0 到 14 的整形数, 模拟调用 func4 这



```

1  #include <stdio.h>
2  int fun4(int edi, int esi, int edx)
3  {
4      int ebx;
5      ebx = (edx - esi) / 2 + esi;
6      if (ebx > edi)
7          return fun4(edi, esi, ebx - 1);
8      else if (ebx < edi)
9          return fun4(edi, ebx + 1, edx);
10     else
11         return ebx;
12 }
13
14 int main()
15 {
16     for (int i = 0; i <= 14; i++)
17     {
18         if (fun4(i, 0, 14) == 7)
19             printf("%d\n", i);
20     }
21 }
22
23
24

```

我们用c语言编写 fun4 函数

因为输入的数只有 0 到 14，我们做一个循环依次作为输入调用 fun4，直到返回值为 7，则打印这个循环变量，即我们需要的数

C:\Users\冯开来\Desktop\fun4.exe 最后运行函数得到结果 — 为 7，即我们需要输入的第一个数为 7

7

Process returned 0 (0x0) execution time : 0.021 s  
Press any key to continue.

个函数，并且当返回值为 7 时，说明我们输入的数正确。

运行程序，我们得到结果为 7，说明当我们输入 7 的时候，func4 函数的返回值是 7，所以我们需要输入的第二个 int 为 7。

1195	40162d:	e8 b9 03 00 00	callq 4019eb <explode_bomb>	
1196	401632:	ba 0e 00 00 00	mov \$0xe,%edx	
1197	401637:	be 00 00 00 00	mov \$0x0,%esi	
1198	40163c:	8b 7d fc	mov -0x4(%rbp),%edi	
1199	40163f:	e8 7a ff ff ff	callq 4015be <func4>	
1200	401644:	83 f8 07	cmp \$0x7,%eax	
1201	401647:	75 06	jne 40164f <phase_4+0x52>	
1202	401649:	83 7d f8 07	cmpl \$0x7,-0x8(%rbp)	比较第二个数和7的大小，若不同则爆炸，说明我们需要输入得第二个数为7。最后根据输入格式，答案为77
1203	40164d:	74 05	je 401654 <phase_4+0x57>	
1204	40164f:	e8 97 03 00 00	callq 4019eb <explode_bomb>	
1205	401654:	c9	leaveq	
1206	401655:	c3	retq	

继续看反汇编代码，下一个指令是将 -0x8(%rbp) 与立即数 0x7 比较，如果相等就跳转到 0x401654，弹栈结束该 phase。所以我们第二个输入的 int 应该为 7。

所以 phase\_4 答案是 77

```

fkl1190201215@ubuntu:~$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Wow! Brazil is big.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2. Keep going!
0 f 556
Halfway there!
7 7
So you got that one. Try this one.

```

### 3.5 阶段 5 的破解与分析

密码如下：71=657

破解过程：

0000000000401656 <phase\_5>:

401656: 55	push %rbp	根据 string_length, 可以判断出这是读入字符串, 并且应该是 6 个字符
401657: 48 89 e5	mov %rsp,%rbp	
40165a: 53	push %rbx	
40165b: 48 83 ec 18	sub \$0x18,%rsp	
40165f: 48 89 fb	mov %rdi,%rbx	
401662: e8 74 02 00 00	callq 4018db <string_length>	
401667: 83 f8 06	cmp \$0x6,%eax	
40166a: 75 24	jne 401690 <phase_5+0x3a>	

根据前面, 以及 string\_length, 我们可以知道这里是读入一个字符串, 并且应该是 6 个字符。

首先是 %eax=0, 将 %eax 与 5 比较, 大于 5 跳出循环, 所以这是一个 6 次的循环。在每次循环中, 让 %rcx=%rax 即为循环变量, %edx 为 %rbx+%rcx, 因为 %rbx

401667: 83 f8 06	cmp \$0x6,%eax	
40166a: 75 24	jne 401690 <phase_5+0x3a>	一个 6 次循环, 当 %eax 大于 5 时, 跳出循环
40166c: b8 00 00 00 00	mov \$0x0,%eax	
401671: 83 f8 05	cmp \$0x5,%eax	
401674: 7f 21	jg 401697 <phase_5+0x41>	
401676: 48 63 c8	movslq %eax,%rcx	%rcx=%rax 为循环变量, 即 %edx 依次为输入字符串的字符, 并取低 4 位, 即一个 0 到 16 的数, 在后续作为索引
401679: 0f b6 14 0b	movzbl (%rbx,%rcx,1),%edx	
40167d: 83 e2 0f	and \$0xf,%edx	
401680: 0f b6 92 b0 31 40 00	movzbl 0x4031b0(%rdx),%edx	
401687: 88 54 0d e9	mov %dl,-0x17(%rbp,%rcx,1)	将 %rcx 作为索引, 寻找地址 0x4031b0 后相应的内容, 并存入 %edx, 再存入首地址为 -0x17 (%rbp) 的数组。最后循环变量 +1
40168b: 83 c0 01	add \$0x1,%eax	
40168e: eb e1	jmp 401671 <phase_5+0x1b>	
401690: e8 56 03 00 00	callq 4019eb <explode_bomb>	
401695: eb d5	jmp 40166c <phase_5+0x16>	
401697: c6 45 ef 00	movb \$0x0,-0x11(%rbp)	循环结束, 将 0 送至 -0x11 (%rbp), 即 6 次循环得到由 6 个字符构成的数组的终止符, 将地址 0x403169 内容送至 %esi, 将我们得到新数组送至 %rdi, 然后将 %rdi 和 %esi 送入 strings_not_equal 函数
40169b: be 69 31 40 00	mov \$0x403169,%esi	
4016a0: 48 8d 7d e9	lea -0x17(%rbp),%rdi	
4016a4: e8 46 02 00 00	callq 4018ef <strings_not_equal>	
4016a9: 85 c0	test %eax,%eax	
4016ab: 75 07	jne 4016b4 <phase_5+0x5e>	
4016ad: 48 83 c4 18	add \$0x18,%rsp	
4016b1: 5b	pop %rbx	
4016b2: 5d	pop %rbp	
4016b3: c3	retq	
4016b4: e8 32 03 00 00	callq 4019eb <explode_bomb>	
4016b9: eb f2	jmp 4016ad <phase_5+0x57>	

是我们输入字符串的首地址，所以每次加一个循环变量即是字符串中依次的字符，并将每个字符取低四位，后续我们知道，这个第四位构成一个数值为 0 到 16 的索引。

得到的%rdx 作为索引，通过间接寻址 0x4031b0(%rdx)，可以猜测到 0x4031b0 处应该存了一个字符串，当%rdx 为 1 时，即字符串第二个字符，依次类推。每次循环中，得到一个间接寻址的字符，存入心得寄存器，首地址为 -0x17(%rbp)，因为是 6 次循环，所以我们得到的新的字符串应该也是长度为 6。

```

Reading symbols from bomb...
(gdb) b *0x4031b0
Breakpoint 1 at 0x4031b0
(gdb) r
Starting program: /home/fkl1190201215/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Wow! Brazil is big.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2. Keep going!
0 f 556
Halfway there!
7 7
So you got that one. Try this one.
7031=37

BOOM!!!
The bomb has blown up.
[Inferior 1 (process 2208) exited with code 010]
(gdb) x/s 0x4031b0
0x4031b0 <array.3404>: "maduiersnfotvbylWow! You've defused the secret stage!"
(gdb) |

```

我们通过设置断点可以查看 0x4031b0 的字符串为 maduiersnfotvbyl，我们输入的 6 个字符分别取低四位可以得到索引，通过索引找到上述这个字符串对应的字符。所以我们现在的工作就是确定这几个索引的值。

继续看反汇编代码。

40167d: 83 e2 0f	and \$0xf,%edx	
401680: 0f b6 92 b0 31 40 00	movzbl 0x4031b0(%rdx),%edx	
401687: 88 54 0d e9	mov %dl,-0x17(%rbp,%rcx,1)	
40168b: 83 c0 01	add \$0x1,%eax	
40168e: eb e1	jmp 401671 <phase_5+0x1b>	
401690: e8 56 03 00 00	callq 4019eb <explode_bomb>	
401695: eb d5	jmp 40166c <phase_5+0x16>	
401697: c6 45 ef 00	movb \$0x0,-0x11(%rbp)	
40169b: be 69 31 40 00	mov \$0x403169,%esi	
4016a0: 48 8d 7d e9	lea -0x17(%rbp),%rdi	
4016a4: e8 46 02 00 00	callq 4018ef <strings_not_equal>	
4016a9: 85 c0	test %eax,%eax	
4016ab: 75 07	jne 4016b4 <phase_5+0x5e>	
4016ad: 48 83 c4 18	add \$0x18,%rsp	
4016b1: 5b	pop %rbx	
4016b2: 5d	pop %rbp	
4016b3: c3	retq	
4016b4: e8 32 03 00 00	callq 4019eb <explode_bomb>	
4016b9: eb f2	jmp 4016ad <phase_5+0x57>	

0x4031b0 存的是我们需要寻找索引的字符串  
0x403169 存的是我们需要比较的字符串

最后比较字符串，若两个不想等，则 %eax=0，并条件跳转到爆炸，相等则弹栈结束



在跳出循环之后，将 0x403169 送入 %esi，-0x17(%rbp) 送入 %rdi，再调用 strings\_not\_equal 这个函数，显然这里是要我们将得到的新字符串与 0x403169 的字符串相比较，如果相同，则弹栈结束该 phase，我们可以通过 gdb 查看 0x403169 地址处的字符串，这个字符串即是我们得到的新的字符串，然后通过字符一一比对原来那个比较长的字符串，得到索引值，然后我们输入的 6 个字符的低四位即是这个索引值。

```

Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...
(gdb) b *0x403169
Breakpoint 1 at 0x403169
(gdb) r
Starting program: /home/fkl1190201215/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Wow! Brazil is big.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2. Keep going!
0 f 556
Halfway there!
7 7
So you got that one. Try this one.
ald;jg

BOOM!!!
The bomb has blown up.
[Inferior 1 (process 2040) exited with code 010]
(gdb) x/s 0x403169
0x403169: "sabres"
(gdb)

```

这个字符串为 sabres，原来的字符串为 maduiersnfotvbyl，一一比对后，我们得到对应的索引值依次为，7 1 13 6 5 7，对应的 16 进制数为，0x0111，0x0001，0x1101，0x0110，0x0101，0x0111。这就要求我们输入的 6 个字符的 ascii 码值的低 4 位就是上述 6 个数，通过查阅 ascii 表，6 个字符依次可以为 7 1 = 6 5 7，通过字符串输入为 71=657。

```

fkl1190201215@ubuntu:~$ ./bommb
bash: ./bommb: 没有那个文件或目录
fkl1190201215@ubuntu:~$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Wow! Brazil is big.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2. Keep going!
0 f 556
Halfway there!
7 7 DrEvil
So you got that one. Try this one.
71=657
Good work! On to the next...

```

### 3.6 阶段 6 的破解与分析

密码如下：1 5 2 6 4 3

破解过程：

```

1242 000000000000401000 <phase_0>:
1243 4016bb: 55 push %rbp
1244 4016bc: 48 89 e5 mov %rsp,%rbp
1245 4016bf: 41 55 push %r13
1246 4016c1: 41 54 push %r12
1247 4016c3: 53 push %rbx
1248 4016c4: 48 83 ec 58 sub $0x58,%rsp
1249 4016c8: 48 8d 75 c0 lea -0x40(%rbp),%rsi
1250 4016cc: e8 3c 03 00 00 callq 401a0d <read_six_numbers>
1251 4016d1: 41 bc 00 00 00 00 mov $0x0,%r12d
1252 4016d7: eb 29 jmp 401702 <phase_6+0x47>
1253 4016d9: e8 0d 03 00 00 callq 4019eb <explode_bomb>
1254 4016de: eb 37 jmp 401717 <phase_6+0x5c>
1255 4016e0: e8 06 03 00 00 callq 4019eb <explode_bomb>
1256 4016e5: 83 c3 01 add $0x1,%ebx
1257 4016e8: 83 fb 05 cmp $0x5,%ebx
1258 4016eb: 7f 12 jg 4016ff <phase_6+0x44>
1259 4016ed: 49 63 c4 movslq %r12d,%rax
1260 4016f0: 48 63 d3 movslq %ebx,%rdx
1261 4016f3: 8b 7c 95 c0 mov -0x40(%rbp,%rdx,4),%edi
1262 4016f7: 39 7c 85 c0 cmp %edi,-0x40(%rbp,%rax,4)
1263 4016fb: 75 e8 jne 4016e5 <phase_6+0x2a>
1264 4016fd: eb e1 jmp 4016e0 <phase_6+0x25>
1265 4016ff: 45 89 ec mov %r13d,%r12d
1266 401702: 41 83 fc 05 cmp $0x5,%r12d
1267 401706: 7f 19 jg 401721 <phase_6+0x66>
1268 401708: 49 63 c4 movslq %r12d,%rax
1269 40170b: 8b 44 85 c0 mov -0x40(%rbp,%rax,4),%eax
1270 40170f: 83 e8 01 sub $0x1,%eax
1271 401712: 83 f8 05 cmp $0x5,%eax
1272 401715: 77 c2 ja 4016d9 <phase_6+0x1e>
1273 401717: 45 8d 6c 24 01 lea 0x1(%r12),%r13d
1274 40171c: 44 89 eb mov %r13d,%ebx
1275 40171f: eb c7 jmp 4016e8 <phase_6+0x2d>
1276 401721: b8 00 00 00 00 mov $0x0,%eax
1277 401726: eb 13 jmp 40173b <phase_6+0x80>
1278 401728: 48 63 c8 movslq %eax,%rcx
1279 40172b: ba 07 00 00 00 mov $0x7,%edx

```

前面很常规的压栈，初始化，读入 6 个整型数，用空格隔开。

将  $\%r12d = 0$ ，跳转到  $0x401702$ ，与 5 比较，当  $\%r12d > 5$ ，跳出循环。

循环主体是让  $\%rax = \%r12d$ ， $\%eax = -0x40(\%rbp, \%rax, 4)$ ，即依次为我们输入的 6 个数，并让  $\%eax - 1$  与 5 比较，大于的时候跳转到爆炸，并让  $\%r13d = \%r12d + 1$ ， $\%ebx = \%r13d$ ，且与 5 比较（循环 5 次），

让  $-0x40(\%rbp, \%r12d, 4)$  与  $-0x40(\%rbp, \%r13d, 4)$  比较，两者要不相等，否则爆炸。

所以第一个大循环的目的是让每个数小于等于 6，且相邻两个数不能相等。

```

1274 40171c: 44 89 ed      mov     %r13d,%edx
1275 40171f: eb c7        jmp     4016e8 <phase_6+0x2d>
1276 401721: b8 00 00 00 00 mov     $0x0,%eax
1277 401726: eb 13        jmp     40173b <phase_6+0x80>
1278 401728: 48 63 c8     movslq  %eax,%rcx
1279 40172b: ba 07 00 00 00 mov     $0x7,%edx
1280 401730: 2b 54 8d c0   sub     -0x40(%rbp,%rcx,4),%edx
1281 401734: 89 54 8d c0   mov     %edx,-0x40(%rbp,%rcx,4)
1282 401738: 83 c0 01     add     $0x1,%eax
1283 40173b: 83 f8 05     cmp     $0x5,%eax
1284 40173e: 7e e8        jle     401728 <phase_6+0x6d>
1285 401740: be 00 00 00 00 mov     $0x0,%esi

```

这是第二个大循环，从 $\%eax=0$  到 $\%eax \leq 5$ ，循环 6 次，循环主体是 $\%edx$  减去  $-0x40(\%rbp, \%rcx, 4)$ ， $\%rcx$  是循环变量的值， $\%edx=7$ ，所以这一段代码的意识是依次用 7 减去我们输入的 6 个数。假设我们输入  $a_0, a_1, a_2, a_3, a_4, a_5$ 。

那么这一段之后得到  $b_0, b_1, b_2, b_3, b_4, b_5$ 。 $b_i=7-a_i$ 。

```

1283 40173b: 83 f8 05     cmp     $0x5,%eax
1284 40173e: 7e e8        jle     401728 <phase_6+0x6d>
1285 401740: be 00 00 00 00 mov     $0x0,%esi
1286 401745: eb 18        jmp     40175f <phase_6+0xa4>
1287 401747: 48 8b 52 08   mov     0x8(%rdx),%rdx
1288 40174b: 83 c0 01     add     $0x1,%eax
1289 40174e: 48 63 ce     movslq  %esi,%rcx
1290 401751: 39 44 8d c0   cmp     %eax,-0x40(%rbp,%rcx,4)
1291 401755: 7f f0        jg      401747 <phase_6+0x8c>
1292 401757: 48 89 54 cd 90 mov     %rdx,-0x70(%rbp,%rcx,8)
1293 40175c: 83 c6 01     add     $0x1,%esi
1294 40175f: 83 fe 05     cmp     $0x5,%esi
1295 401762: 7f 0c        jg      401770 <phase_6+0xb5>
1296 401764: b8 01 00 00 00 mov     $0x1,%eax
1297 401769: ba d0 52 40 00 mov     $0x4052d0,%edx
1298 40176e: eb de        jmp     40174e <phase_6+0x93>
1299 401770: 48 8b 5d 90   mov     -0x70(%rbp),%rbx

```

第二个循环结束后跳转到  $0x401740$ ，开始第三个循环， $\%esi=0$ ，当 $\%esi \leq 5$ ，循环 6 次。这里是整个 phase 最关键的地方，这里将 1 送至 $\%eax$ ， $0x4052d0$  送至 $\%edx$ ，让 $\%eax$  和  $-0x40(\%rbp, \%rcx, 4)$  进行比较 [ 直到 $\%eax$  等于  $-0x40(\%rbp, \%rcx, 4)$  ]，每次循环一次将 $\%rdx$  的地址加 8 并存入 $\%rdx$ ，所以这里可以判断 $\%rdx$  是一个链表，每次 $\%eax$  的循环会让 $\%rdx$  进入下一个节点，然后这个节点的数存入  $-0x70(\%rbp, \%rcx, 8)$  我们可以理解为这里是一个新的数组，首地址是  $-0x70(\%rbp)$ ，每次间隔 8 个地址，我们暂且命名为  $c_0, c_1, c_2, c_3, c_4, c_5$ 。 $c[i]$  的得到方式是通过 1 与  $b[i]$  的差值， $c_i$  等于内存为  $8 * (b_i) + 0x4052d0$  的值。所以我们需要直到  $0x4052d0$  存的链表的值和地址。

```

1298 40176e: eb de      jmp     40174e <phase_6+0x93>
1299 401770: 48 8b 5d 90 mov     -0x70(%rbp),%rbx
1300 401774: 48 89 d9    mov     %rbx,%rcx
1301 401777: b8 01 00 00 00 mov     $0x1,%eax
1302 40177c: eb 12      jmp     401790 <phase_6+0xd5>
1303 40177e: 48 63 d0    movslq  %eax,%rdx
1304 401781: 48 8b 54 d5 90 mov     -0x70(%rbp,%rdx,8),%rdx
1305 401786: 48 89 51 08 mov     %rdx,0x8(%rcx)
1306 40178a: 83 c0 01    add     $0x1,%eax
1307 40178d: 48 89 d1    mov     %rdx,%rcx
1308 401790: 83 f8 05    cmp     $0x5,%eax
1309 401793: 7e e9      jle     40177e <phase_6+0xc3>
1310 401795: 48 c7 41 08 00 00 00 movq    $0x0,0x8(%rcx)
1311 40179c: 00

```

得到新的数组 ci 后，跳出循环，我们进入第四个循环。说实话看完这一个循环我实在不知道这里的意义在何处。这里的意思是将 c[i+1] 的值覆盖 c[i]，所以从数值上来说我们得到了 c1, c2, c3, c4, c5, c5, 0（因为最后将 0 送入了数组的末尾）。

```

1311 40179c: 00
1312 40179d: 41 bc 00 00 00 00 mov     $0x0,%r12d
1313 4017a3: eb 08      jmp     4017ad <phase_6+0xf2>
1314 4017a5: 48 8b 5b 08 mov     0x8(%rbx),%rbx
1315 4017a9: 41 83 c4 01 add     $0x1,%r12d
1316 4017ad: 41 83 fc 04 cmp     $0x4,%r12d
1317 4017b1: 7f 11      jg      4017c4 <phase_6+0x109>
1318 4017b3: 48 8b 43 08 mov     0x8(%rbx),%rax
1319 4017b7: 8b 00      mov     (%rax),%eax
1320 4017b9: 39 03      cmp     %eax,(%rbx)
1321 4017bb: 7d e8      jge     4017a5 <phase_6+0xea>
1322 4017bd: e8 29 02 00 00 callq   4019eb <explode_bomb>
1323 4017c2: eb e1      jmp     4017a5 <phase_6+0xea>
1324 4017c4: 48 83 c4 58 add     $0x58,%rsp
1325 4017c8: 5b        pop     %rbx
1326 4017c9: 41 5c      pop     %r12
1327 4017cb: 41 5d      pop     %r13
1328 4017cd: 5d        pop     %rbp
1329 4017ce: c3        retq
1330

```

最后一个循环开始之前是将 0 送至 %r12d，然后将 %r12d 与 4 比较，这里循环其实就 5 次，循环的主体是将 c[i] 与 c[i+1] 比较，只有当 c[i] ≥ c[i+1] 时，成立，否则爆炸，所以我们干脆规定 c0 ≥ c1 ≥ c2 ≥ c3 ≥ c4 ≥ c5，而 ci 的值是通过链表寻址得到的，所以我们需要知道所有链表的数。

```

[Inferior 1 (process 12770) exited with code 010]
(gdb) x/24xw 0x4052d0
0x4052d0 <node1>: 0x00000154 0x00000001 0x004052e0 0x00000000
0x4052e0 <node2>: 0x000002a4 0x00000002 0x004052f0 0x00000000
0x4052f0 <node3>: 0x000000ab 0x00000003 0x00405300 0x00000000
0x405300 <node4>: 0x0000004d 0x00000004 0x00405310 0x00000000
0x405310 <node5>: 0x00000223 0x00000005 0x00405320 0x00000000
0x405320 <node6>: 0x000003e0 0x00000006 0x00000000 0x00000000
(gdb) q
fk1190201215@ubuntu:~$ ./bomb ans.txt
Welcome to my fiendish little bomb. You have 6 phases with

```

根据数值比较，我们得到 node6 ≥ node2 ≥ node5 ≥ node1 ≥ node3 ≥ node4。然后我们可以根据更高 ci 和 bi 的公式得到 bi 的值依次为 6 2 5 1 3 4，然后根据 ai 和 bi



的转换公式得到 1 5 2 6 4 3。即我们需要输入的 6 个数，以空格隔开就是答案。

```

which to blow yourself up. Have a nice day!
Wow! Brazil is big.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2. Keep going!
0 f 556
Halfway there!
7 7 DrEvil
So you got that one. Try this one.
71=657
Good work! On to the next...
1 5 2 6 4 3
Curses, you've found the secret phase!
But finding it and solving it are quite different...

```

### 3.7 阶段 7 的破解与分析(隐藏阶段)

密码如下：7

破解过程：

最后一个隐藏关先是找到入口，在 `phase_defused` 里面

```

1619 000000000401b74 <phase_defused>:
1620 401b74: 83 3d f1 3b 00 00 06  cmpl $0x6,0x3bf1(%rip)      # 40576c <num_input_strings>
1621 401b7b: 74 01                  je 401b7e <phase_defused+0xa>
1622 401b7d: c3                     retq
1623 401b7e: 55                     push %rbp
1624 401b7f: 48 89 e5               mov %rsp,%rbp
1625 401b82: 48 83 ec 60            sub $0x60,%rsp
1626 401b86: 4c 8d 45 b0            lea -0x50(%rbp),%r8
1627 401b8a: 48 8d 4d a8            lea -0x58(%rbp),%rcx
1628 401b8e: 48 8d 55 ac            lea -0x54(%rbp),%rdx
1629 401b92: be 41 33 40 00         mov $0x403341,%esi
1630 401b97: bf 70 58 40 00         mov $0x405870,%edi
1631 401b9c: b8 00 00 00 00         mov $0x0,%eax
1632 401ba1: e8 6a f5 ff ff        callq 401110 <__isoc99_sscanf@plt>
1633 401ba6: 83 f8 03               cmp $0x3,%eax
1634 401ba9: 74 0c                  je 401bb7 <phase_defused+0x43>
1635 401bab: bf 80 32 40 00         mov $0x403280,%edi
1636 401bb0: e8 ab f4 ff ff        callq 401060 <puts@plt>
1637 401bb5: c9                     leaveq
1638 401bb6: c3                     retq
1639 401bb7: be 4a 33 40 00         mov $0x40334a,%esi
1640 401bbc: 48 8d 7d b0            lea -0x50(%rbp),%rdi
1641 401bc0: e8 2a fd ff ff        callq 4018ef <strings_not_equal>
1642 401bc5: 85 c0                  test %eax,%eax
1643 401bc7: 75 e2                  jne 401bab <phase_defused+0x37>
1644 401bc9: bf 20 32 40 00         mov $0x403220,%edi
1645 401bce: e8 8d f4 ff ff        callq 401060 <puts@plt>
1646 401bd3: bf 48 32 40 00         mov $0x403248,%edi
1647 401bd8: e8 83 f4 ff ff        callq 401060 <puts@plt>
1648 401bdd: b8 00 00 00 00         mov $0x0,%eax
1649 401be2: e8 22 fc ff ff        callq 401809 <secret_phase>
1650 401be7: eb c2                  jmp 401bab <phase_defused+0x37>

```

当输入 0x40334a 处存的字符串可以调出 `secret_phase`，通过 `gdb` 可以查到 0x40334a 处的字符串为 `DrEvil`，所以我们在第三关结束的时候输入 7 7 `DrEvil`（即在原本的第四关答案后面加上 `DrEvil` 就可以进入隐藏关）。在 6 个炸弹拆

除后，我们可以进入隐藏关

```

bash: ./bommb: 没有那个文件或目录
fkl1190201215@ubuntu:~$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Wow! Brazil is big.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2. Keep going!
0 f 556
Halfway there!
7 7 DrEvil
So you got that one. Try this one.
71=657
Good work! On to the next...
1 5 2 6 4 3
Curses, you've found the secret phase!
But finding it and solving it are quite different...

```

然后我们来看隐藏关的反汇编代码。

```

1354 0000000000401809 <secret_phase>:
1355 401809: 55          push    %rbp
1356 40180a: 48 89 e5    mov     %rsp,%rbp
1357 40180d: 53          push    %rbx
1358 40180e: 48 83 ec 08 sub     $0x8,%rsp
1359 401812: e8 32 02 00 00 callq   401a49 <read_line>
1360 401817: 48 89 c7    mov     %rax,%rdi
1361 40181a: e8 21 f9 ff ff callq   401140 <atoi@plt>
1362 40181f: 89 c3      mov     %eax,%ebx
1363 401821: 8d 40 ff    lea     -0x1(%rax),%eax
1364 401824: 3d e8 03 00 00 cmp     $0x3e8,%eax
1365 401829: 77 27      ja      401852 <secret_phase+0x49>
1366 40182b: 89 de      mov     %ebx,%esi
1367 40182d: bf f0 50 40 00 mov     $0x4050f0,%edi
1368 401832: e8 98 ff ff ff callq   4017cf <fun7>
1369 401837: 83 f8 04    cmp     $0x4,%eax
1370 40183a: 75 1d      jne     401859 <secret_phase+0x50>
1371 40183c: bf c0 31 40 00 mov     $0x4031c0,%edi
1372 401841: e8 1a f8 ff ff callq   401060 <puts@plt>
1373 401846: e8 29 03 00 00 callq   401b74 <phase_defused>
1374 40184b: 48 83 c4 08 add     $0x8,%rsp
1375 40184f: 5b        pop     %rbx
1376 401850: 5d        pop     %rbp
1377 401851: c3        retq
1378 401852: e8 94 01 00 00 callq   4019eb <explode_bomb>
1379 401857: eb d2      jmp     40182b <secret_phase+0x22>
1380 401859: e8 8d 01 00 00 callq   4019eb <explode_bomb>
1381 40185e: eb dc      jmp     40183c <secret_phase+0x33>

```

和第六关很像，关键就是 0x4050f0 处存的一个链表，这个链表作为参数传入 fun7 这个函数，在函数中又用到了这个链表。其实我们通过查看这个地址以及后面的 100 字节，我们可以发现这是一个特殊的链表——二叉树。

```

0x4050f0 <n1+16>: 0x00000000
(gdb) x/200xw 0x4050f0
0x4050f0 <n1>: 0x00000024 0x00000000 0x00405110 0x00000000
0x405100 <n1+16>: 0x00405130 0x00000000 0x00000000 0x00000000
0x405110 <n21>: 0x00000008 0x00000000 0x00405190 0x00000000
0x405120 <n21+16>: 0x00405150 0x00000000 0x00000000 0x00000000
0x405130 <n22>: 0x00000032 0x00000000 0x00405170 0x00000000
0x405140 <n22+16>: 0x004051b0 0x00000000 0x00000000 0x00000000
0x405150 <n32>: 0x00000016 0x00000000 0x00405270 0x00000000
0x405160 <n32+16>: 0x00405230 0x00000000 0x00000000 0x00000000
0x405170 <n33>: 0x0000002d 0x00000000 0x004051d0 0x00000000
0x405180 <n33+16>: 0x00405290 0x00000000 0x00000000 0x00000000
0x405190 <n31>: 0x00000006 0x00000000 0x004051f0 0x00000000
0x4051a0 <n31+16>: 0x00405250 0x00000000 0x00000000 0x00000000
0x4051b0 <n34>: 0x0000006b 0x00000000 0x00405210 0x00000000
0x4051c0 <n34+16>: 0x004052b0 0x00000000 0x00000000 0x00000000
0x4051d0 <n45>: 0x00000028 0x00000000 0x00000000 0x00000000
0x4051e0 <n45+16>: 0x00000000 0x00000000 0x00000000 0x00000000
0x4051f0 <n41>: 0x00000001 0x00000000 0x00000000 0x00000000
0x405200 <n41+16>: 0x00000000 0x00000000 0x00000000 0x00000000
0x405210 <n47>: 0x00000063 0x00000000 0x00000000 0x00000000
0x405220 <n47+16>: 0x00000000 0x00000000 0x00000000 0x00000000
0x405230 <n44>: 0x00000023 0x00000000 0x00000000 0x00000000
0x405240 <n44+16>: 0x00000000 0x00000000 0x00000000 0x00000000
0x405250 <n42>: 0x00000007 0x00000000 0x00000000 0x00000000
0x405260 <n42+16>: 0x00000000 0x00000000 0x00000000 0x00000000
0x405270 <n43>: 0x00000014 0x00000000 0x00000000 0x00000000
0x405280 <n43+16>: 0x00000000 0x00000000 0x00000000 0x00000000
0x405290 <n46>: 0x0000002f 0x00000000 0x00000000 0x00000000
0x4052a0 <n46+16>: 0x00000000 0x00000000 0x00000000 0x00000000
0x4052b0 <n48>: 0x0000003e9 0x00000000 0x00000000 0x00000000
0x4052c0 <n48+16>: 0x00000000 0x00000000 0x00000000 0x00000000

```

然后查看 fun7 这个函数

```

1330
1331 00000000004017cf <fun7>:
1332 4017cf: 48 85 ff          test    %rdi,%rdi
1333 4017d2: 74 2f             je      401803 <fun7+0x34>
1334 4017d4: 55               push   %rbp
1335 4017d5: 48 89 e5         mov     %rsp,%rbp
1336 4017d8: 8b 07           mov     (%rdi),%eax
1337 4017da: 39 f0           cmp     %esi,%eax
1338 4017dc: 7f 09           jg      4017e7 <fun7+0x18>
1339 4017de: 75 14           jne     4017f4 <fun7+0x25>
1340 4017e0: b8 00 00 00 00  mov     $0x0,%eax
1341 4017e5: 5d             pop     %rbp
1342 4017e6: c3             retq
1343 4017e7: 48 8b 7f 08     mov     0x8(%rdi),%rdi
1344 4017eb: e8 df ff ff ff  callq   4017cf <fun7>
1345 4017f0: 01 c0           add     %eax,%eax
1346 4017f2: eb f1           jmp     4017e5 <fun7+0x16>
1347 4017f4: 48 8b 7f 10     mov     0x10(%rdi),%rdi
1348 4017f8: e8 d2 ff ff ff  callq   4017cf <fun7>
1349 4017fd: 8d 44 00 01     lea     0x1(%rax,%rax,1),%eax
1350 401801: eb e2           jmp     4017e5 <fun7+0x16>
1351 401803: b8 ff ff ff ff  mov     $0xffffffff,%eax
1352 401808: c3             retq

```

相等返回 0，大于跳到左子树，返回  $2 \times$  左子树的值，小于跳到右子树，返回  $2 \times$  右子树 + 1，最后的返回值是 4 的时候则成功。  $4 = 2 * 2 * (2 * 0 + 1)$ ，所以可以知道路径是先左子树再左子树再左子树，找到节点为 7，所以输入结果为 7。

```
fk1190201215@ubuntu:~$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Wow! Brazil is big.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2. Keep going!
0 f 556
Halfway there!
7 7 DrEvil
So you got that one. Try this one.
71=657
Good work! On to the next...
1 5 2 6 4 3
Curses, you've found the secret phase!
But finding it and solving it are quite different...
7
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

- ✓ 说实话课上听反汇编的内容听的不是很懂，但是通过这次实验，真正让我看懂了反汇编代码，虽然之后的日子里可能除了考试我也很少有机会会接触反汇编代码，但是这段时光——一个人一杯热水一段代码看一天的日子，我永远会记得。实验很有趣，因为需要像闯关一样破解密码，期间持续性的成就感和满足感不停激励我，让我一直坚持下去，感谢老师，感谢自己。
- ✓ 更加熟悉了 gdb 调试工具和 edb 的使用方法，对计算机运作的原理可能更深了一点点吧。

### 4.2 请给出对本次实验内容的建议

本次实验还是可以的，趣味性极强，难度有阶梯型，适合从入手小白开始慢慢上手。

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>.