

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 1190201816

班 级 1903012

学 生 樊红雨

指 导 教 师 史先俊

实 验 地 点 G709

实 验 日 期 2021.4.21

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.....	- 16 -
3.6 阶段 6 的破解与分析.....	- 16 -
3.7 阶段 7 的破解与分析(隐藏阶段).....	- 16 -
第 4 章 总结.....	- 17 -
4.1 请总结本次实验的收获.....	- 17 -
4.2 请给出对本次实验内容的建议.....	- 17 -
参考文献.....	- 18 -

第 1 章 实验基本信息

1.1 实验目的

熟练掌握计算机系统的 ISA 指令系统与寻址方式。

熟练掌握 Linux 下调试器的反汇编调试跟踪分析机器语言的方法

增强对程序机器级表示、汇编语言、调试器和逆向工程等的理解

1.2 实验环境与工具

1.2.1 硬件环境

X64 CPU; 2GHz; 2G RAM; 256GHD Disk 以上

1.2.2 软件环境

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

1.2.3 开发工具

GDB/OBJDUMP; EDB; KDD 等

1.3 实验预习

- 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、O3、Og、-m32/m64。再次查看生成的汇编语言与原来的区别。
- 注意 O1 之后缺省无栈帧，RBP 为普通寄存器。用 -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。

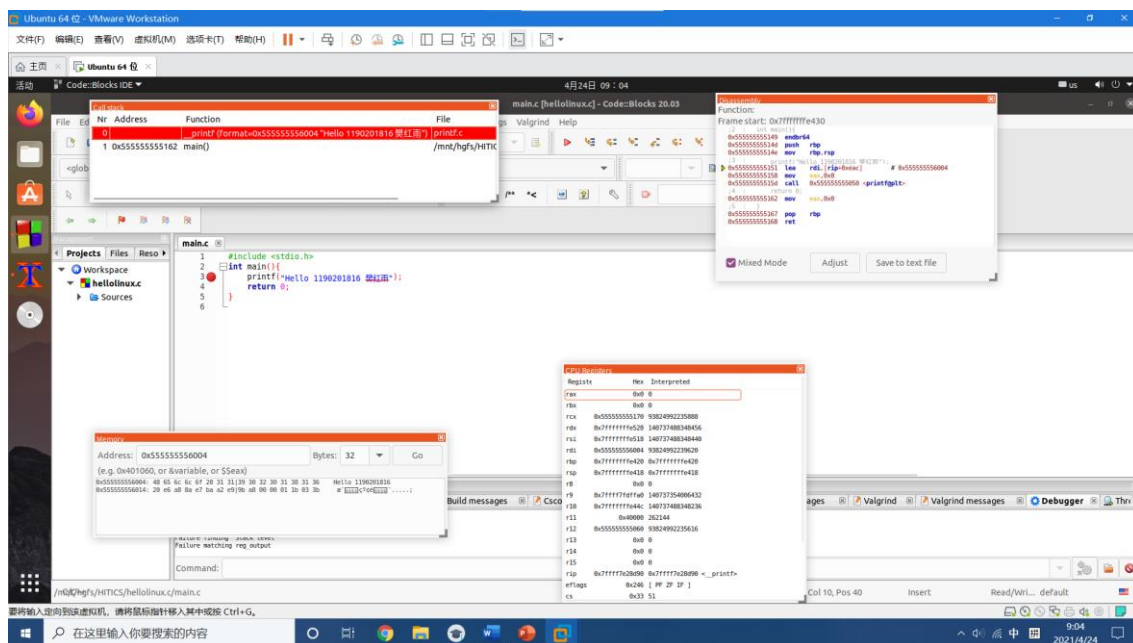


图 2-1 Ubuntu 下 CodeBlocks 反汇编截图

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

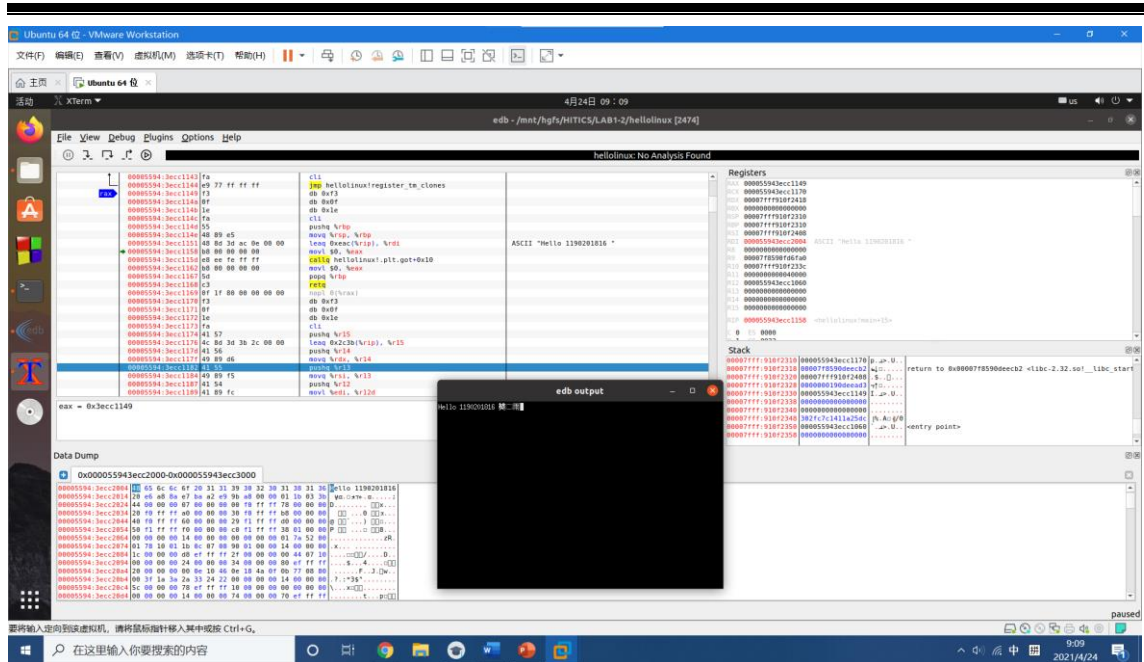


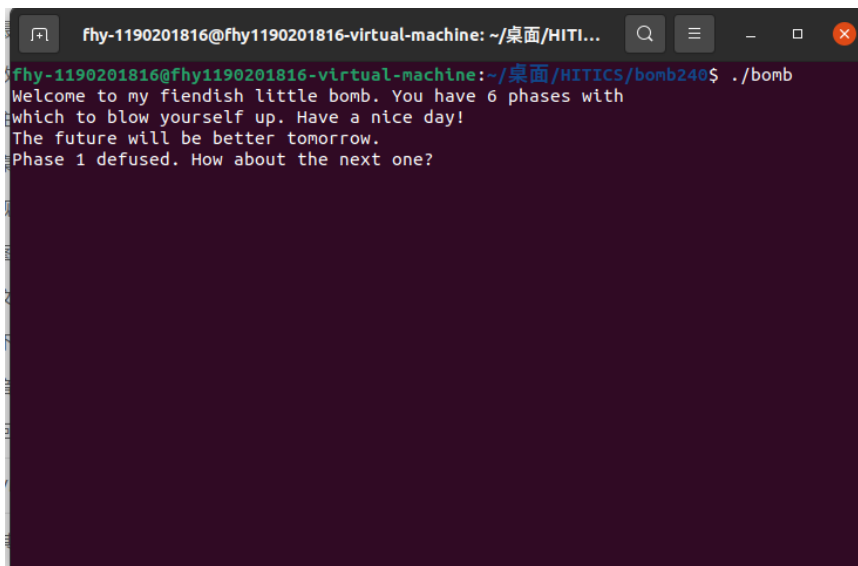
图 2-2 Ubuntu 下 EDB 截图

第 3 章 各阶段炸弹破解与分析

每阶段 40 分，密码 20 分，分析 20 分，总分不超过 80 分

3.1 阶段 1 的破解与分析

密码如下：The future will be better tomorrow.

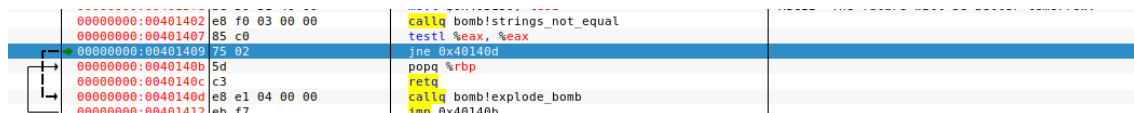
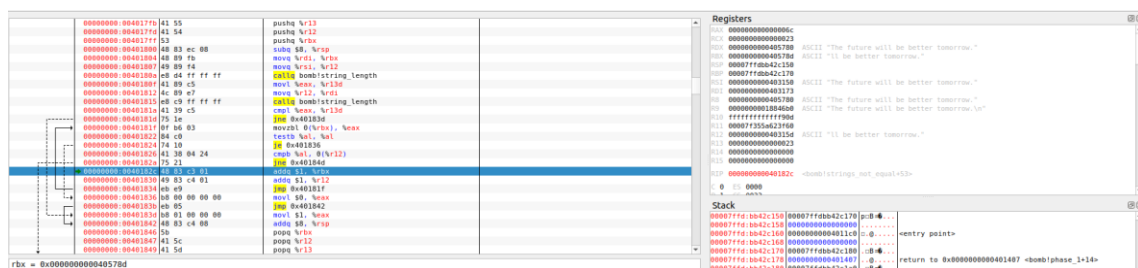


```
fhy-1190201816@fhy1190201816-virtual-machine: ~/桌面/HITI...
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/bomb240$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
The future will be better tomorrow.
Phase 1 defused. How about the next one?
```

破解过程：利用反汇编进入 `phrase_1` 函数后发现程序将字符串常量“The future will be better tomorrow.” 送给寄存器 `%esi`，并且调用函数 `bomb!strings_not_equal`。

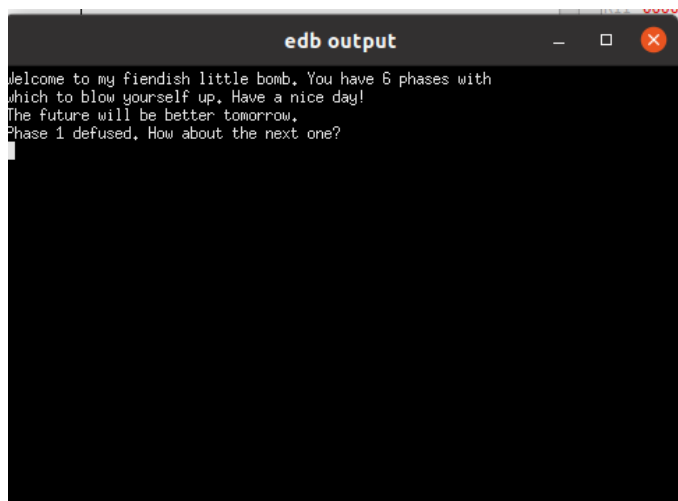
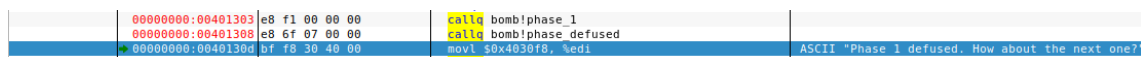
<code>pushq %rbp</code>	
<code>movq %rsp, %rbp</code>	
<code>movl \$0x403150, %esi</code>	ASCII "The future will be better tomorrow."
<code>callq bomb!strings_not_equal</code>	

在 `bomb!strings_not_equal` 函数中，比较用户输入的字符串是否与 “The future will be better tomorrow.” 相等。



如果两个字符串不相等，则跳转到 `bomb!explode_bomb` 函数，炸弹爆炸。

若相等，则退出 `phrase_1`，阶段 1 拆除成功。



3.2 阶段 2 的破解与分析

密码如下：1 2 4 7 11 16


```

fhy-1190201816@fhy1190201816-virtual-machine: ~/桌面/HITI...
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/bomb240$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
The future will be better tomorrow.
Phase 1 defused. How about the next one?
1 2 4 7 11 16
That's number 2. Keep going!

```

破解过程：进入到 `phrase_2` 函数中，发现需要调用 `bomb!read_six_numbers` 函数，并且发现该函中出现了 6 个 `%d`，推测阶段 2 需要读入 6 个数字。

00000000:00401915	55		pushq %rbp	
00000000:00401916	48 89 e5		movq %rsp, %rbp	
00000000:00401919	48 89 f2		movq %rsi, %rdx	
00000000:0040191c	48 8d 4e 04		leaq 4(%rsi), %rcx	
00000000:00401920	48 8d 46 14		leaq 0x14(%rsi), %rax	
00000000:00401924	50		pushq %rax	
00000000:00401925	48 8d 46 10		leaq 0x10(%rsi), %rax	
00000000:00401929	50		pushq %rax	
00000000:0040192a	4c 8d 4e 0c		leaq 0xc(%rsi), %r9	
00000000:0040192e	4c 8d 46 08		leaq 8(%rsi), %r8	
00000000:00401932	be 23 33 40 00		movl \$0x403323, %esi	ASCII "%d %d %d %d %d %d"

读入 6 个数字后，首先将读入的第一个数字与 0 进行比较，若小于 0，则跳转到炸弹爆炸函数，所以读入的第一个数字需要大于或等于 0。

00000000:00401426	83 7d d0 00		cmpl \$0, -0x30(%rbp)	
00000000:0040142a	78 07		js 0x401433	
00000000:0040142c	bb 01 00 00 00		movl \$1, %ebx	
00000000:00401431	eb 0f		jmp 0x401442	
00000000:00401433	e8 bb 04 00 00		callq bomb!explode_bomb	

接下来进入循环，发现这段汇编代码表示一个循环的程序。

00000000:0040143f	83 c3 01		addl \$1, %ebx	
00000000:00401442	83 fb 05		cmpl \$5, %ebx	
00000000:00401445	7f 17		jb 0x40145e	
00000000:00401447	48 63 c3		movslq %ebx, %rax	
00000000:0040144a	8d 53 ff		leal -1(%rbx), %edx	
00000000:0040144d	48 63 d2		movslq %edx, %rdx	
00000000:00401450	89 d9		movl %ebx, %ecx	
00000000:00401452	03 4c 95 d0		addl -0x30(%rbp, %rdx, 4), %ecx	
00000000:00401456	39 4c 85 d0		cmpl %ecx, -0x30(%rbp, %rax, 4)	
00000000:0040145a	74 e3		je 0x40143f	
00000000:0040145c	eb dc		jmp 0x40143a	

设我们输入的第 n 个数字为 A_n ，通过分析可知，这段循环程序表示：

$A_{n+1} = A_n + 1$ 这个通项公式。若输入的数字不满足这个公式则立即跳转到炸弹爆炸函数。若输入的每个数字之间都满足这个公式则拆除炸弹成功。退出 `phrase_2`。

00000000:0040131f	e8 f0 00 00 00		callq bomb!phase_2	
00000000:00401324	e8 53 07 00 00		callq bomb!phase_defused	
00000000:00401329	bf 3d 30 40 00		movl \$0x40303d, %edi	ASCII "That's number 2. Keep going!"
00000000:0040132e	e8 2d fd ff ff		callq bomb!putsplt	

3.3 阶段 3 的破解与分析

密码如下：0 152

```

fhy-1190201816@fhy1190201816-virtual-machine: ~/桌面/HITI...
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/bomb240$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
The future will be better tomorrow.
Phase 1 defused. How about the next one?
1 2 4 7 11 16
That's number 2. Keep going!
0 152
Halfway there!
  
```

破解过程：进入 `phrase_3` 函数，发现程序将两个 `%d` 传给寄存器，猜想阶段 3 需要输入两个数字。

00000000:00401465	55	pushq %rbp	
00000000:00401466	48 89 e5	movq %rsp, %rbp	
00000000:00401469	48 83 ec 10	subq \$0x10, %rsp	
00000000:0040146d	48 8d 4d f8	leaq -8(%rbp), %rcx	
00000000:00401471	48 8d 55 fc	leaq -4(%rbp), %rdx	
00000000:00401475	be 2f 33 40 00	movl \$0x40332f, %esi	ASCII "%d %d"
00000000:0040147a	b8 00 00 00 00	movl \$0, %eax	

`%eax` 中先存放用户输入的数字的个数，将它与 1 比较，如果小于或等于 1，则炸弹爆炸，所以输入的数字个数需要大于 1。

00000000:00401484	83 f8 01	cmpl \$1, %eax	
00000000:00401487	7e 11	jle 0x40149a	
00000000:00401489	8b 45 fc	movl -4(%rbp), %eax	
00000000:0040148c	83 f8 07	cmpl \$7, %eax	
00000000:0040148f	77 46	ja 0x4014d7	
00000000:00401491	89 c0	movl %eax, %eax	
00000000:00401493	ff 24 c5 a0 31 40 00	jmpq *0x4031a0(, %rax, 8)	
00000000:0040149a	e8 54 04 00 00	callq bombexplode_bomb	

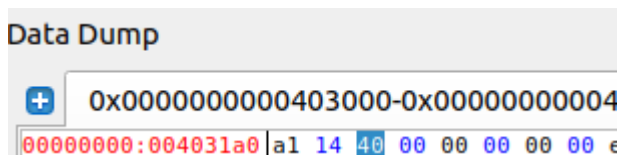
随后，`%eax` 中存放用户输入的的第一个数字，将它与 7 比较，如果大于 7，则炸弹爆炸，所以输入的的第一个数字需要小于 7。

00000000:0040148c	83 f8 07	cmpl \$7, %eax	
00000000:0040148f	77 46	ja 0x4014d7	

程序会根据输入的的第一个数字跳转到不同的地址。

	00000000:00401493	ff 24 c5 a0 31 40 00	jmpq *0x4031a0(, %rax, 8)
	00000000:0040149a	e8 54 04 00 00	callq bomb!explode_bomb
	00000000:0040149f	eb e8	jmp 0x401489
	00000000:004014a1	b8 98 00 00 00	movl \$0x98, %eax
	00000000:004014a6	39 45 f8	cmpl %eax, -8(%rbp)
	00000000:004014a9	75 3f	jne 0x4014ea
	00000000:004014ab	c9	leave
	00000000:004014ac	c3	retq
	00000000:004014ad	b8 84 01 00 00	movl \$0x184, %eax
	00000000:004014b2	eb f2	jmp 0x4014a6
	00000000:004014b4	b8 36 01 00 00	movl \$0x136, %eax
	00000000:004014b9	eb eb	jmp 0x4014a6
	00000000:004014bb	b8 15 01 00 00	movl \$0x115, %eax
	00000000:004014c0	eb e4	jmp 0x4014a6
	00000000:004014c2	b8 82 02 00 00	movl \$0x282, %eax
	00000000:004014c7	eb dd	jmp 0x4014a6
	00000000:004014c9	b8 61 03 00 00	movl \$0x361, %eax
	00000000:004014ce	eb d6	jmp 0x4014a6
	00000000:004014d0	b8 05 02 00 00	movl \$0x205, %eax
	00000000:004014d5	eb cf	jmp 0x4014a6
	00000000:004014d7	e8 17 04 00 00	callq bomb!explode_bomb
	00000000:004014dc	b8 00 00 00 00	movl \$0, %eax
	00000000:004014e1	eb c3	jmp 0x4014a6
	00000000:004014e3	b8 47 01 00 00	movl \$0x147, %eax
	00000000:004014e8	eb bc	jmp 0x4014a6

通过内存查询地址 0x4031a0 保存的内容为：a1 14 40 00.由于该计算机的存储方式为小段法，正常的顺序应为：0x004014a1，发现是反汇编某一行的地址。分析可知该段程序为一个分支语句，通过输入的第一个数字进行跳转。并且不同的跳转语句会赋给寄存器不同的常数。



即用户输入的第二个数字为 0 时，跳转到 0x004014a1 语句。该语句将 0x98 赋值给寄存器，并且将该数字与-8(%rbp)进行比较，-8(%rbp)即为用户输入的第二个数字。若这两个数字相同，则退出 phrase_3，炸弹拆除成功。若不相同，则炸弹爆炸。

00000000:004014a1	b8 98 00 00 00	movl \$0x98, %eax
00000000:004014a6	39 45 f8	cmpl %eax, -8(%rbp)
00000000:004014a9	75 3f	jne 0x4014ea
00000000:004014ab	c9	leave
00000000:004014ac	c3	retq

0x98 转化为十进制数字为：152。所以输入的一组数字可以为 0 152。

还有其他的组合，如 1 327 也可以拆除炸弹。

```
edb output

Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
The future will be better tomorrow.
Phase 1 defused. How about the next one?
1 2 4 7 11 16
That's number 2. Keep going!
0 152
Halfway there!
```

```
fhy-1190201816@fhy1190201816-virtual-machine: ~/桌面/HITI...
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/bomb240$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
The future will be better tomorrow.
Phase 1 defused. How about the next one?
1 2 4 7 11 16
That's number 2. Keep going!
1 327
Halfway there!
```

3.4 阶段 4 的破解与分析

密码如下：66 2

```
fhy-1190201816@fhy1190201816-virtual-machine: ~/桌面/HITI...
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/bomb240$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
The future will be better tomorrow.
Phase 1 defused. How about the next one?
1 2 4 7 11 16
That's number 2. Keep going!
0 152
Halfway there!
66 2
So you got that one. Try this one.
```

破解过程：进入到 `phrase_4` 函数中，发现寄存器保存了两个“%d”，猜测该阶段需要输入两个数字。

00000000:0040153d	ed ed	jmp 0x401527	
00000000:0040153c	55	pushq %rbp	
00000000:0040153d	48 89 e5	movq %rsp, %rbp	
00000000:00401540	48 83 ec 10	subq \$0x10, %rsp	
00000000:00401544	48 8d 4d fc	leaq -4(%rbp), %rcx	
00000000:00401548	48 8d 55 f8	leaq -8(%rbp), %rdx	
00000000:0040154c	be 2f 33 40 00	movl \$0x40332f, %esi	ASCII "%d %d"

函数将 `%eax` 中的数据与 2 比较，若不等于 2，则炸弹爆炸。

说明该函数只接受两个数字。

同时，函数将传入的第二个数字与 1 比较，若小于等于 1，炸弹爆炸。

若大于 4，则炸弹爆炸。

说明密码的第二个数字为大于 1 且小于等于 4 的数字。

00000000:0040155b	83 f8 02	cmpl \$2, %eax	
00000000:0040155e	75 0d	jne 0x40156d	
00000000:00401560	8b 45 fc	movl -4(%rbp), %eax	
00000000:00401563	83 f8 01	cmpl \$1, %eax	
00000000:00401566	7e 05	jle 0x40156d	
00000000:00401568	83 f8 04	cmpl \$4, %eax	
00000000:0040156b	7e 05	jle 0x401572	
00000000:0040156d	e8 81 03 00 00	callq bomb!explode_bomb	

继续单步执行，发现函数将我们输入的第二个数字保存到寄存器 `%esi` 中，将常数 7 保存到寄存器 `%edi` 中，将 `%edi` 作为第一个参数，`%esi` 作为第二个参数，将他们传递给函数 `bomb! fun4`。

并且将该函数的返回值与 `-8(%rbp)` 比较，即与密码的第一个数字比较，若不相等则炸弹爆炸，若相等则炸弹解除。

00000000:00401572	8b 75 fc	movl -4(%rbp), %esi	
00000000:00401575	bf 07 00 00 00	movl \$7, %edi	
00000000:0040157a	e8 72 ff ff ff	callq bomb!func4	
00000000:0040157f	39 45 f8	cmpl %eax, -8(%rbp)	
00000000:00401582	75 02	jne 0x401586	
00000000:00401584	c9	leave	
00000000:00401585	c3	retq	
00000000:00401586	e8 68 03 00 00	callq bomb!explode_bomb	

接下来分析 `bomb! fun4` 函数。

从整体上看，该函数中又调用了两次该函数，可发现该函数为一个输入两个参数的递归函数。

00000000:004014f1	85 ff	testl %edi, %edi
00000000:004014f3	7e 3d	jle 0x401532
00000000:004014f5	55	pushq %rbp
00000000:004014f6	48 89 e5	movq %rsp, %rbp
00000000:004014f9	41 55	pushq %r13
00000000:004014fb	41 54	pushq %r12
00000000:004014fd	53	pushq %rbx
00000000:004014fe	48 83 ec 08	subq \$8, %rsp
00000000:00401502	41 89 fc	movl %edi, %r12d
00000000:00401505	89 f3	movl %esi, %ebx
00000000:00401507	83 ff 01	cmpl \$1, %edi
00000000:0040150a	74 2c	je 0x401538
00000000:0040150c	8d 7f ff	leal -1(%rdi), %edi
00000000:0040150f	e8 dd ff ff ff	callq bomb!func4
00000000:00401514	44 8d 2c 18	leal 0(%rax, %rbx), %r13d
00000000:00401518	41 8d 7c 24 fe	leal -2(%r12), %edi
00000000:0040151d	89 de	movl %ebx, %esi
00000000:0040151f	e8 cd ff ff ff	callq bomb!func4
00000000:00401524	44 01 e8	addl %r13d, %eax
00000000:00401527	48 83 c4 08	addq \$8, %rsp
00000000:0040152b	5b	popq %rbx
00000000:0040152c	41 5c	popq %r12
00000000:0040152e	41 5d	popq %r13
00000000:00401530	5d	popq %rbp
00000000:00401531	c3	retq

当传入的第一个参数等于 0 时，直接返回 0。

00000000:00401531	c3	retq
00000000:00401532	b8 00 00 00 00	movl \$0, %eax

当传入的第一个参数等于 1 时，直接返回传入的第二个参数。

00000000:00401524	44 01 e8	addl %r13d, %eax
00000000:00401527	48 83 c4 08	addq \$8, %rsp
00000000:0040152b	5b	popq %rbx
00000000:0040152c	41 5c	popq %r12
00000000:0040152e	41 5d	popq %r13
00000000:00401530	5d	popq %rbp
00000000:00401531	c3	retq
00000000:00401532	b8 00 00 00 00	movl \$0, %eax
00000000:00401537	c3	retq
00000000:00401538	89 f0	movl %esi, %eax
00000000:0040153a	eb eb	jmp 0x401527

发现两次递归之前一次将第一个参数-2，另一次将第一个参数-1。

00000000:00401502	41 89 fc	movl %edi, %r12d
00000000:00401505	89 f3	movl %esi, %ebx
00000000:00401507	83 ff 01	cmpl \$1, %edi
00000000:0040150a	74 2c	je 0x401538
00000000:0040150c	8d 7f ff	leal -1(%rdi), %edi
00000000:0040150f	e8 dd ff ff ff	callq bomb!func4
00000000:00401514	44 8d 2c 18	leal 0(%rax, %rbx), %r13d
00000000:00401518	41 8d 7c 24 fe	leal -2(%r12), %edi
00000000:0040151d	89 de	movl %ebx, %esi
00000000:0040151f	e8 cd ff ff ff	callq bomb!func4
00000000:00401524	44 01 e8	addl %r13d, %eax

最后将两次函数的返回值相加，再加上传入的第二个参数作为返回值。

可分析出该递归函数的结构由 C 语言编写后如下：

```
int fun4(int n, int m)
{
    if (n == 0)
        return 0;
    else if(n == 1)
        return m;
    else
        return fun4(n-2,m) + m + fun4(n-1,m);
}
```

则可计算出当密码的第二个数字为 2 时的递归过程：

$\text{fun4}(0,2)=0$

$\text{fun4}(1,2)=2$

$\text{fun4}(2,2)=4$

$\text{fun4}(3,2)=8$

$\text{fun4}(4,2)=14$

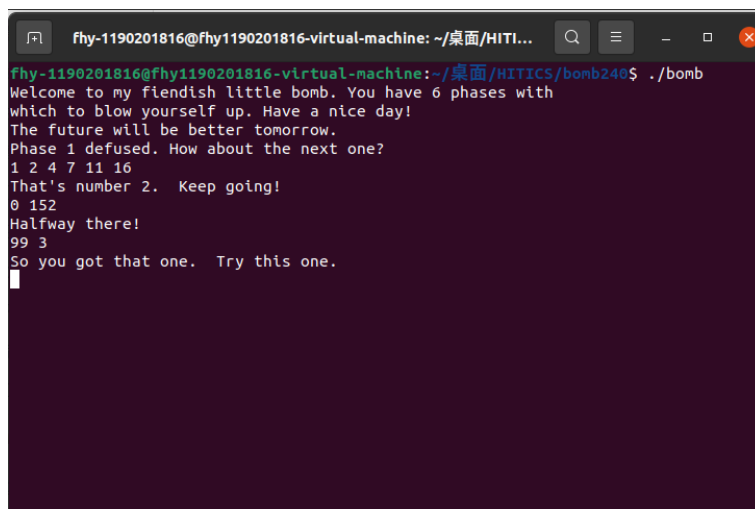
$\text{fun4}(5,2)=24$

$\text{fun4}(6,2)=40$

$\text{fun4}(7,2)=66$

并且还可得到其他答案：99 3

验证如下：



```
fhy-1190201816@fhy1190201816-virtual-machine: ~/桌面/HITI...
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/bomb240$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
The future will be better tomorrow.
Phase 1 defused. How about the next one?
1 2 4 7 11 16
That's number 2. Keep going!
0 152
Halfway there!
99 3
So you got that one. Try this one.
```

3.5 阶段 5 的破解与分析

密码如下：

破解过程：

3.6 阶段 6 的破解与分析

密码如下：

破解过程：

3.7 阶段 7 的破解与分析(隐藏阶段)

密码如下：

破解过程：

第 4 章 总结

4.1 请总结本次实验的收获

本实验学会了利用 gdb 来调试程序，查看程序的反汇编及其内存。学会了 edb 的使用，学会了利用 edb 调试程序，学会了通过分析程序的反汇编，通过分析寄存器保存的内容来分析程序的执行过程。学会了通过上述过程来破解密码，对反汇编语句的运用更加熟练。对于堆栈有了更深的理解。

4.2 请给出对本次实验内容的建议

老师可以多讲一下 edb 的各种使用方法，以及 edb 的各种快捷键的使用方法。

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>.