

# 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 120L020701

班 级 2003005

学 生 董琦

指 导 教 师 吴锐

实 验 地 点 宿舍

实 验 日 期 2022/4/1

## 计算学部

## 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 -
第 2 章 实验环境建立 .....	- 4 -
2.1 UBUNTU 下 CODEBLOCKS 反汇编（10 分） .....	- 4 -
2.2 UBUNTU 下 EDB 运行环境建立（10 分） .....	- 4 -
第 3 章 各阶段炸弹破解与分析 .....	- 5 -
3.1 阶段 1 的破解与分析 .....	- 5 -
3.2 阶段 2 的破解与分析 .....	- 5 -
3.3 阶段 3 的破解与分析 .....	- 6 -
3.4 阶段 4 的破解与分析 .....	- 7 -
3.5 阶段 5 的破解与分析 .....	- 8 -
3.6 阶段 6 的破解与分析 .....	- 9 -
3.7 阶段 7 的破解与分析(隐藏阶段) .....	- 11 -
第 4 章 总结 .....	- 14 -
4.1 请总结本次实验的收获 .....	- 14 -
4.2 请给出对本次实验内容的建议 .....	- 14 -
参考文献 .....	- 15 -

## 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

处理器 Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz 8 核  
机带 RAM 16.0 GB (15.8 GB 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:

版本 Windows 11 家庭中文版

版本 21H2

安装日期 2022/2/4

操作系统版本 22000.556

体验 Windows 功能体验包 1000.22000.556.0

Ubuntu:

版本 Ubuntu 20.04.3 LTS

类型 64 位

#### 1.2.3 开发工具

Windows: Visual Studio 2019

Ubuntu: VScode gcc version 9.4.0 (Ubuntu 9.4.0-1ubuntu1~20.04), edb,gdb

## 2.1 Ubuntu 下 CodeBlocks 反汇编 (10 分)

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。

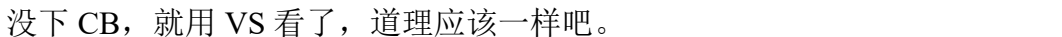


图 2-1 Windows 下 VisualCode 反汇编截图

[illegible]

- 4 -

## 第 3 章 各阶段炸弹破解与分析

每阶段 30 分，密码 10 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：I am the mayor. I can do anything I want.

破解过程：

由于主函数里面只有读取输入和函数调用，所以我们只需要看调用的函数即可。

00000000:004013ec	48 83 ec 08	subq \$8, %rsp	begin of phase1
00000000:004013f0	be 50 31 40 00	movl \$0x403150, %esi	ASCII "I am the mayor. I can do anything I want."
00000000:004013f5	e8 cd 04 00 00	callq bomb!strings_not_equal	
00000000:004013fa	85 c0	testl %eax, %eax	
00000000:004013fc	75 05	jne 0x401403	
00000000:004013fe	48 83 c4 08	addq \$8, %rsp	
00000000:00401402	c3	retq	
00000000:00401403	e8 a1 05 00 00	callq bomb!explode_bomb	
00000000:00401408	eb f4	jmp 0x4013fe	end of phase1

该段为 phase1 的函数，可以看到里面调用了比较字符串是否相等的函数，然后检测返回值是否为 0，若不为 0 则炸弹爆炸，另外一个参数为右面的那个字符串地址，内容即为密码，由此第一题解。

### 3.2 阶段 2 的破解与分析

密码如下：1 2 4 8 16 32

破解过程：

00000000:0040140a	53	pushq %rbx	begin of phase2
00000000:0040140b	48 83 ec 20	subq \$0x20, %rsp	
00000000:0040140f	48 89 e6	movq %rsp, %rsi	
00000000:00401412	e8 b6 05 00 00	callq bomb!read_six_numbers	
00000000:00401417	83 3c 24 01	cmpl \$1, 0(%rsp)	
00000000:0040141b	75 07	jne 0x401424	
00000000:0040141d	bb 01 00 00 00	movl \$1, %ebx	
00000000:00401422	eb 0f	jmp 0x401433	
00000000:00401424	e8 80 05 00 00	callq bomb!explode_bomb	
00000000:00401429	eb f2	jmp 0x40141d	
00000000:0040142b	e8 79 05 00 00	callq bomb!explode_bomb	
00000000:00401430	83 c3 01	addl \$1, %ebx	
00000000:00401433	83 fb 05	cmpl \$5, %ebx	
00000000:00401436	7f 14	jg 0x40144c	
00000000:00401438	48 63 d3	movslq %ebx, %rdx	
00000000:0040143b	8d 43 ff	leal -1(%rbx), %eax	
00000000:0040143e	48 98	cltq	
00000000:00401440	8b 04 84	movl 0(%rsp, %rax, 4), %eax	
00000000:00401443	01 c0	addl %eax, %eax	
00000000:00401445	39 04 94	cmpl %eax, 0(%rsp, %rdx, 4)	
00000000:00401448	74 e6	jg 0x401430	
00000000:0040144a	eb df	jmp 0x40142b	
00000000:0040144c	48 83 c4 20	addq \$0x20, %rsp	
00000000:00401450	5b	popq %rbx	
00000000:00401451	c3	retq	end of phase2

如图为 phase2 的函数。

首先看到他调用了 `read_six_numbers` 函数，将之保存在一个数组里，，因此将栈指针作为参数传入，可知我们应该输入六个数字。

然后把数组第一个元素和 1 比较，若不相等则爆炸，可知第一个数字应该为 1。

之后函数进行一个循环，检测后一个元素是否为前一个元素的二倍，可知输入数列应该为一个公比为 2 的等比数列。

### 3.3 阶段 3 的破解与分析

密码如下：0 p 480

破解过程：

00000000:00401452	48 83 ec 18	subq \$0x18, %rsp	begin of phase3
00000000:00401456	4c 8d 44 24 08	leaq 8(%rsp), %r8	
00000000:0040145b	48 8d 4c 24 07	leaq 7(%rsp), %rcx	
00000000:00401460	48 8d 54 24 0c	leaq 0xc(%rsp), %rdx	
00000000:00401465	be a6 31 40 00	movl \$0x4031a6, %esi	ASCII "%d %c %d"
00000000:0040146a	b8 00 00 00 00	movl \$0, %eax	
00000000:0040146f	e8 9c fc ff ff	callq bomb!_isoc99_sscanf@plt	
00000000:00401474	83 f8 02	cmpl \$2, %eax	
00000000:00401477	7e 16	jle 0x40148f	
00000000:00401479	8b 44 24 0c	movl 0xc(%rsp), %eax	
00000000:0040147d	83 f8 07	cmpl \$7, %eax	
00000000:00401480	0f 87 0a 01 00 00	ja 0x401590	
00000000:00401486	89 c0	movl %eax, %eax	
00000000:00401488	ff 24 c5 c0 31 40 00	jmpq *0x4031c0(, %rax, 8)	00401496
00000000:0040148f	e8 15 05 00 00	callq bomb!explode_bomb	
00000000:00401494	eb e3	jmp 0x401479	
00000000:00401496	81 7c 24 08 e0 01 00 00	cmpl \$0x1e0, 8(%rsp)	0
00000000:0040149e	75 0a	jne 0x4014aa	
00000000:004014a0	b8 70 00 00 00	movl \$0x70, %eax	
00000000:004014a5	e9 f0 00 00 00	jmp 0x40159a	
00000000:004014a8	e8 fa 00 00 00	callq bomb!explode_bomb	
00000000:004014af	b8 70 00 00 00	movl \$0x70, %eax	
00000000:004014b4	e9 e1 00 00 00	jmp 0x40159a	
00000000:004014b9	81 7c 24 08 b3 02 00 00	cmpl \$0x2b3, 8(%rsp)	1
00000000:004014c1	75 0a	jne 0x4014cd	
00000000:004014c3	b8 75 00 00 00	movl \$0x75, %eax	
00000000:004014c8	e9 cd 00 00 00	jmp 0x40159a	
00000000:004014cd	e8 d7 00 00 00	callq bomb!explode_bomb	
00000000:004014d2	b8 75 00 00 00	movl \$0x75, %eax	
00000000:004014d7	e9 be 00 00 00	jmp 0x40159a	
00000000:004014dc	81 7c 24 08 98 03 00 00	cmpl \$0x398, 8(%rsp)	2

上图为 phase3 部分代码。

首先可以看到函数从字符串中提取三个参数，俩个数字和一个字符，然后检测第一个数字是否大于 7，可知第一个参数需要小于等于 7。然后跟着一个跳转指令，跳转目标以一个数据内存地址中所存的数为基地址的数组。

然后我们找到该内存位置，可以看到所存的为 0X00401496，为上图标 0 位置，后续还有七个元素，分别为代码的某个地址，而且各代码段结构相同。可以看出这是一个 switch 表，一共有 0 到 7 八个入口，由第一个参数决定。

在 switch 中，比较了第三个参数和给定数字，若不相等则炸弹爆炸，由此可以得到第二个数字的值，其取决于第一个参数。然后又在 `eax` 中保存了一个一字

节的数字，不过扩展为 4 字节。在 switch 结束后，看到将第二个参数与这个数做比较，若不相等则炸弹爆炸，由此可知第二个参数的 ascii 编码应该等于这个数，由此得到第二个参数的值。

由于该关卡为一个 switch 表，因此有 8 种可能结果，我这里只写了第一种。

### 3.4 阶段 4 的破解与分析

密码如下：7 0 DrEvil

破解过程：

	00000000:004015e9	48 83 ec 18	subq \$0x18, %rsp	begin of phase 4
	00000000:004015ed	48 8d 4c 24 08	leaq 8(%rsp), %rcx	
	00000000:004015f2	48 8d 54 24 0c	leaq 0xc(%rsp), %rdx	
	00000000:004015f7	be 1f 33 40 00	movl \$0x40331f, %esi	ASCII "%d %d"
	00000000:004015fc	b8 00 00 00 00	movl \$0, %eax	
	00000000:00401601	e8 0a fb ff ff	callq bomb!_isoc99_sscanf@plt	
	00000000:00401606	83 f8 02	cmpl \$2, %eax	
	00000000:00401609	75 0d	jne 0x401618	
	00000000:0040160b	8b 44 24 0c	movl 0xc(%rsp), %eax	
	00000000:0040160f	85 c0	testl %eax, %eax	
	00000000:00401611	78 05	js 0x401618	
	00000000:00401613	83 f8 0e	cmpl \$0xe, %eax	
	00000000:00401616	7e 05	jle 0x40161d	
	00000000:00401618	e8 8c 03 00 00	callq bomb!explode_bomb	
	00000000:0040161d	ba 0e 00 00 00	movl \$0xe, %edx	
	00000000:00401622	be 00 00 00 00	movl \$0, %esi	
	00000000:00401627	8b 7c 24 0c	movl 0xc(%rsp), %edi	
	00000000:0040162b	e8 7c ff ff ff	callq bomb!func4	
	00000000:00401630	85 c0	testl %eax, %eax	
	00000000:00401632	75 07	jne 0x40163b	
	00000000:00401634	83 7c 24 08 00	cmpl \$0, 8(%rsp)	
	00000000:00401639	74 05	je 0x401640	
	00000000:0040163b	e8 69 03 00 00	callq bomb!explode_bomb	
	00000000:00401640	48 83 c4 18	addq \$0x18, %rsp	
	00000000:00401644	c3	retq	end of phase 4

上图为 phase4 函数代码，可知需要读入俩个整数，存在临时变量中，然后将第一个整数和 0, 14 作为参数传入调用 func4 函数，检测返回值是否为 0，不为 0 则爆炸。然后检测第二个整数是否为 0，可知第二个整数应该是 0。

	00000000:004015ac	48 83 ec 08	subq \$8, %rsp	begin of fun4
	00000000:004015b0	89 d1	movl %edx, %ecx	
	00000000:004015b2	29 f1	subl %esi, %ecx	
	00000000:004015b4	89 c8	movl %ecx, %eax	
	00000000:004015b6	c1 e8 1f	shrl \$0x1f, %eax	
	00000000:004015b9	01 c8	addl %ecx, %eax	
	00000000:004015bb	d1 f8	sarl \$1, %eax	
	00000000:004015bd	01 f0	addl %esi, %eax	
	00000000:004015bf	39 f8	cmpl %edi, %eax	
	00000000:004015c1	7f 0c	jb 0x4015cf	
	00000000:004015c3	7c 16	jl 0x4015db	
	00000000:004015c5	b8 00 00 00 00	movl \$0, %eax	
	00000000:004015ca	48 83 c4 08	addq \$8, %rsp	
	00000000:004015cc	c3	retq	
	00000000:004015cf	8d 50 ff	leal -1(%rax), %edx	
	00000000:004015d2	e8 d5 ff ff ff	callq bomb!func4	
	00000000:004015d7	01 c0	addl %eax, %eax	
	00000000:004015d9	eb ef	jmp 0x4015ca	
	00000000:004015db	8d 70 01	leal 1(%rax), %esi	
	00000000:004015de	e8 c9 ff ff ff	callq bomb!func4	
	00000000:004015e3	8d 44 00 01	leal 1(%rax, %rax), %eax	
	00000000:004015e7	eb e1	jmp 0x4015ca	end of fun4

然后我们看 fun4 内容。此时%edi 中为我们的第一个数字，%esi 中为 0，%edx 中为 14。开头几句的处理是为了得到俩个数的中间值，其中向右移 31 位是得到了差值的符号位，然后做一个偏置处理（由于他这个只除以 2，相当于向右移一位，因此直接把符号位的值重复利用了，所以一开始没咋看懂。）这样保证了这个算差值一半是向 0 舍入的。

然后比较我们输入的第一个数字是否等于这个平均值。若小于，则将该中间值减 1 作为第三个参数递归调用 `fun4`，若大于，则炸弹爆炸。若等于，则拆弹成功。可知这是一个类似于折半查找的函数，可能的解有 7, 3, 1, 0.

我这里直接将 7 做为答案输入。

### 3.5 阶段 5 的破解与分析

密码如下：IONEFG

破解过程：

	00000000:00401645	53	pushq %rbx	begin of phase5
	00000000:00401646	48 83 ec 10	subq \$0x10, %rsp	
	00000000:0040164a	48 89 fb	movq %rdi, %rbx	
	00000000:0040164d	e8 61 02 00 00	callq bombstring_length	
	00000000:00401652	83 f8 06	cmpl \$6, %eax	
	00000000:00401655	75 24	jng 0x40167b	
	00000000:00401657	b8 00 00 00 00	movl \$0, %eax	
	00000000:0040165c	83 f8 05	cmpl \$5, %eax	
	00000000:0040165f	7f 21	jg 0x401682	
	00000000:00401661	48 63 c8	movslq %eax, %rcx	
	00000000:00401664	0f b6 14 0b	movzbl 0(%rbx, %rcx), %edx	
	00000000:00401668	83 e2 0f	andl \$0xf, %edx	
	00000000:0040166b	0f b6 92 00 32 40 00	movzbl 0x403200(%rdx), %edx	
	00000000:00401672	88 54 0c 09	movb %dl, 9(%rsp, %rcx)	
	00000000:00401676	83 c0 01	addl \$1, %eax	
	00000000:00401679	eb e1	jmp 0x40165c	
	00000000:0040167b	e8 29 03 00 00	callq bombexplode_bomb	
	00000000:00401680	eb d5	jmp 0x401657	
	00000000:00401682	c6 44 24 0f 00	movb \$0, 0xf(%rsp)	
	00000000:00401687	be af 31 40 00	movl \$0x4031af, %esi	ASCII "flyers"
	00000000:0040168c	48 8d 7c 24 09	leaq 9(%rsp), %rdi	
	00000000:00401691	e8 31 02 00 00	callq bombstrings_not_equal	
	00000000:00401696	85 c0	testl %eax, %eax	
	00000000:00401698	75 06	jng 0x4016a0	
	00000000:0040169a	48 83 c4 10	addq \$0x10, %rsp	
	00000000:0040169e	5b	popq %rbx	
	00000000:0040169f	c3	retq	
	00000000:004016a0	e8 04 03 00 00	callq bombexplode_bomb	
	00000000:004016a5	eb f3	jmp 0x40169a	end of phase5

然后来到第 5 阶段，首先看到了 `flyers` 这个字段，不过尝试后不是答案，那么就好好看看函数了。

首先检查参数读取，可以看到要求输入一个长度为 6 的字符串。然后对字符串各字符进行一个循环。然后我们看循环内部。

首先他对我们输入的字符做了一个掩码处理，只取了第四位，然后将得到的数字作为索引从 `0X403200` 这个地址中取值，存到一个临时数组中。那我们看看这个地址里存着什么。

```
0000:00403200 | 6d 61 64 75 69 65 72 73 6e 66 6f 74 76 62 79 6c | madiersnfotvbyl
0000:00403210 | 53 6f 70 70 6f 75 70 74 68 60 60 6b 70 70 6f 75 | So you think you
```

如图，可以看到这个地址中存了一些乱序的字母，仔细观察，包含了 `flyers` 的所有字母。然后我们看到循环后将得到的字符数组与 `flyers` 做比较，可知我们应该按顺序取出 `flyers` 这六个字母，这要求我们输入的字符串低四位对应的值是这六个字母对应的位置下标，由此就可以得到应该输入的答案，此答案有多种可能，因为高四位是任意的，我这里只取了一种，即 `IONEFG`。



### 3.6 阶段 6 的破解与分析

密码如下：6 5 3 1 4 2

破解过程：

这个阶段的代码很长，我们慢慢分析。

00000000:004016a7	41 54	pushq %r12	begin of phase6
00000000:004016a9	55	pushq %rbp	
00000000:004016aa	53	pushq %rbx	
00000000:004016ab	48 83 ec 50	subq \$0x50, %rsp	
00000000:004016af	48 8d 74 24 30	leaq 0x30(%rsp), %rsi	
00000000:004016b4	e8 14 03 00 00	callq bomb!read_six_numbers	

首先从这部分可以看到我们需要读入六个数字，存在一个临时数组当中。

00000000:004016b9	bd 00 00 00 00	movl \$0, %ebp	
00000000:004016be	eb 29	jmp 0x4016e9	
00000000:004016c0	e8 e4 02 00 00	callq bomb!explode_bomb	
00000000:004016c5	eb 36	jmp 0x4016fd	
00000000:004016c7	e8 dd 02 00 00	callq bomb!explode_bomb	
00000000:004016cc	83 c3 01	addl \$1, %ebx	
00000000:004016cf	83 fb 05	cmpl \$5, %ebx	
00000000:004016d2	7f 12	jg 0x4016e6	
00000000:004016d4	48 63 c5	movslq %ebp, %rax	
00000000:004016d7	48 63 d3	movslq %ebx, %rdx	
00000000:004016da	8b 7c 94 30	movl 0x30(%rsp, %rdx, 4), %edi	
00000000:004016de	39 7c 84 30	cmpl %edi, 0x30(%rsp, %rax, 4)	
00000000:004016e2	75 e8	jne 0x4016cc	
00000000:004016e4	eb e1	jmp 0x4016c7	
00000000:004016e6	44 89 e5	movl %r12d, %ebp	
00000000:004016e9	83 fd 05	cmpl \$5, %ebp	
00000000:004016ec	7f 18	jg 0x401706	
00000000:004016ee	48 63 c5	movslq %ebp, %rax	
00000000:004016f1	8b 44 84 30	movl 0x30(%rsp, %rax, 4), %eax	
00000000:004016f5	83 e8 01	subl \$1, %eax	
00000000:004016f8	83 f8 05	cmpl \$5, %eax	
00000000:004016fb	77 c3	ja 0x4016c0	
00000000:004016fd	44 8d 65 01	leal 1(%rbp), %r12d	
00000000:00401701	44 89 e3	movl %r12d, %ebx	
00000000:00401704	eb c9	jmp 0x4016cf	

这段代码对我们输入的参数进行了检测，流程不仔细分析了，其作用就是依次检查这六个数字是否不大于 6 并且互不相等。（不过这里有点漏洞感觉，从后面我可以看出他是输入 1 到 6 的数字，这里没有限制大于 0.其次他也没有检测出一定读入了六个数字，可能是在 read\_six\_numbers 里检测了？）

00000000:0040170b	eb 13	jmp 0x401720	
00000000:0040170d	48 63 c8	movslq %eax, %rcx	
00000000:00401710	ba 07 00 00 00	movl \$7, %edx	
00000000:00401715	2b 54 8c 30	subl 0x30(%rsp, %rcx, 4), %edx	
00000000:00401719	89 54 8c 30	movl %edx, 0x30(%rsp, %rcx, 4)	
00000000:0040171d	83 c0 01	addl \$1, %eax	
00000000:00401720	83 f8 05	cmpl \$5, %eax	
00000000:00401723	7e e8	jle 0x40170d	

参数检测完毕后，又将各个数字对 7 取了一个补。

00000000:00401725	be 00 00 00 00	movl \$0, %esi	
00000000:0040172a	eb 17	jmp 0x401743	
00000000:0040172c	48 8b 52 08	movq 8(%rdx), %rdx	
00000000:00401730	83 c0 01	addl \$1, %eax	
00000000:00401733	48 63 ce	movslq %esi, %rcx	
00000000:00401736	39 44 8c 30	cmpl %eax, 0x30(%rsp, %rcx, 4)	
00000000:0040173a	7f f0	jg 0x40172c	
00000000:0040173c	48 89 14 cc	movq %rdx, 0(%rsp, %rcx, 8)	
00000000:00401740	83 c6 01	addl \$1, %esi	
00000000:00401743	83 fe 05	cmpl \$5, %esi	
00000000:00401746	7f 0c	jg 0x401754	
00000000:00401748	b8 01 00 00 00	movl \$1, %eax	
00000000:0040174d	ba d0 52 40 00	movl \$0x4052d0, %edx	
00000000:00401752	eb df	jmp 0x401733	list head7

这段代码读了好久才读懂。他首先用到了一个地址中的内容，去看看是什么：

j:004052d0	c5 03 00 00 01 00 00 00 e0 52 40 00 00 00 00 00	H.....R@.
j:004052e0	6f 03 00 00 02 00 00 00 f0 52 40 00 00 00 00 00	o.....R@.
j:004052f0	26 02 00 00 03 00 00 00 00 53 40 00 00 00 00 00	&.....S@.
j:00405300	53 03 00 00 04 00 00 00 10 53 40 00 00 00 00 00	S.....S@.
j:00405310	b4 00 00 00 05 00 00 00 20 53 40 00 00 00 00 00	H.....S@.
j:00405320	2f 02 00 00 06 00 00 00 00 00 00 00 00 00 00 00	/.....

观察后发现，这是一个长度为六的链表，并且很贴心的标注了 123456 这个序号。然后再看代码，发现他是将这链表的六个节点的地址存到了一个临时数组中，顺序即为我们输入六个数字所对应的顺序。（比如第 1 个数字为 6，那么就把第 6 个节点地址作为数组第一个元素，然后昂，他这个链表节点顺序也不是按图上那个顺序，是反着标的，比如标 1 的是第六个节点，这样就对了）

00000000:00401754	48 8b 1c 24	movq 0(%rsp), %rbx
00000000:00401758	48 89 d9	movq %rbx, %rcx
00000000:0040175b	b8 01 00 00 00	movl \$1, %eax
00000000:00401760	eb 11	jmp 0x401773
00000000:00401762	48 63 d0	movslq %eax, %rdx
00000000:00401765	48 8b 14 d4	movq 0(%rsp, %rdx, 8), %rdx
00000000:00401769	48 89 51 08	movq %rdx, 8(%rcx)
00000000:0040176d	83 c0 01	addl \$1, %eax
00000000:00401770	48 89 d1	movq %rdx, %rcx
00000000:00401773	83 f8 05	cmpl \$5, %eax
00000000:00401776	7e ea	jle 0x401762
00000000:00401778	48 c7 41 08 00 00 00 00	movq \$0, 8(%rcx)

然后这段代码就是把这个链表按照我们那个地址的顺序重排了一下，然后末节点指向 null。

00000000:00401780	bd 00 00 00 00	movl \$0, %ebp
00000000:00401785	eb 07	jmp 0x40178e
00000000:00401787	48 8b 5b 08	movq 8(%rbx), %rbx
00000000:0040178b	83 c5 01	addl \$1, %ebp
00000000:0040178e	83 fd 04	cmpl \$4, %ebp
00000000:00401791	7f 11	jg 0x4017a4
00000000:00401793	48 8b 43 08	movq 8(%rbx), %rax
00000000:00401797	8b 00	movl 0(%rax), %eax
00000000:00401799	39 03	cmpl %eax, 0(%rbx)
00000000:0040179b	7d ea	jge 0x401787
00000000:0040179d	e8 07 02 00 00	callq bomb!explode_bomb
00000000:004017a2	eb e3	jmp 0x401787
00000000:004017a4	48 83 c4 50	addq \$0x50, %rsp

这段代码是遍历了一遍列表，检测前一个节点元素是否大于后一个节点元素，若不大于则炸弹爆炸，由此可知这个阶段是让我们输入一个顺序，将这个链表从大到小重新排序。不过这里有一个需要注意的是，比较的数字是四字节的，因此需要忽略后面的四字节，只比较前四个字节，并且这是一个小端法机器，要注意高低位。故到这里我们便可以得到答案了，仔细对一下数字可以得

到 6 5 3 1 4 2。

### 3.7 阶段 7 的破解与分析(隐藏阶段)

密码如下：22

破解过程：

首先看汇编代码可以找到 `secret_phase` 这个函数，但是不知道怎么进去。

00000000:004017ea	53	pushq %rbx	begin of secret phase
00000000:004017eb	e8 1c 02 00 00	callq bomb!read_line	
00000000:004017f0	48 89 c7	movq %rax, %rdi	
00000000:004017f3	e8 48 f9 ff ff	callq bomb!atoi@plt	
00000000:004017f9	89 c3	movl %eax, %ebx	
00000000:004017fa	8d 40 ff	leal -1(%rax), %eax	
00000000:004017fd	3d e8 03 00 00	cmpl \$0x3e8, %eax	
00000000:00401802	77 22	ja 0x401826	
00000000:00401804	89 de	movl %ebx, %esi	
00000000:00401806	bf f0 50 40 00	movl \$0x4050f0, %edi	
00000000:0040180b	e8 9d ff ff ff	callq bomb!fun7	
00000000:00401810	83 f8 02	cmpl \$2, %eax	
00000000:00401813	75 18	jne 0x40182d	
00000000:00401815	bf 00 31 40 00	movl \$0x403100, %edi	ASCII "Wow! You've defused the secret stage!"
00000000:0040181a	e8 41 f8 ff ff	callq bomb!puts@plt	
00000000:0040181f	e8 16 03 00 00	callq bomb!phase_defused	
00000000:00401824	5b	popq %rbx	
00000000:00401825	c3	retq	
00000000:00401826	e8 7e 01 00 00	callq bomb!explode_bomb	
00000000:0040182b	eb d7	jmp 0x401804	
00000000:0040182d	e8 77 01 00 00	callq bomb!explode_bomb	
00000000:00401832	eb e1	jmp 0x401815	end

然后我看了看源代码，发现只有 `phase_defused` 这个函数我没仔细看过了，而其他 1 到 6 阶段的代码我都仔细看过，没有调用这个函数的，由此我去瞅瞅 `phase_defused` 的源代码。

00000000:00401b3a	83 3d 2b 3c 00 00 06	cmpl \$6, 0x3c2b(%rip)	begin of phase_defused
00000000:00401b41	74 01	jg 0x401b44	
00000000:00401b43	c3	retq	
00000000:00401b44	48 83 ec 68	subq \$0x68, %rsp	
00000000:00401b48	4c 8d 44 24 10	leaq 0x10(%rsp), %r8	
00000000:00401b4d	48 8d 4c 24 08	leaq 8(%rsp), %rcx	
00000000:00401b52	48 8d 54 24 0c	leaq 0xc(%rsp), %rdx	
00000000:00401b57	be 69 33 40 00	movl \$0x403369, %esi	ASCII "%d %d %s"
00000000:00401b5c	bf 70 58 40 00	movl \$0x405870, %edi	
00000000:00401b61	b8 00 00 00 00	movl \$0, %eax	
00000000:00401b66	e8 a5 f5 ff ff	callq bomb!_isoc99_sscanf@plt	
00000000:00401b6b	83 f8 03	cmpl \$3, %eax	
00000000:00401b6e	74 0f	jg 0x401b7f	
00000000:00401b70	bf a8 32 40 00	movl \$0x4032a8, %edi	ASCII "Congratulations! You've defused the bomb!"
00000000:00401b75	e8 e6 f4 ff ff	callq bomb!puts@plt	
00000000:00401b7a	48 83 c4 68	addq \$0x68, %rsp	
00000000:00401b7e	c3	retq	
00000000:00401b7f	be 72 33 40 00	movl \$0x403372, %esi	ASCII "DrEvil"
00000000:00401b84	48 8d 7c 24 10	leaq 0x10(%rsp), %rdi	
00000000:00401b89	e8 39 fd ff ff	callq bomb!strings_not_equal	
00000000:00401b8e	85 c0	testl %eax, %eax	
00000000:00401b90	75 de	jne 0x401b70	
00000000:00401b92	bf 48 32 40 00	movl \$0x403248, %edi	ASCII "Curses, you've found the secret phase!"
00000000:00401b97	e8 c4 f4 ff ff	callq bomb!puts@plt	
00000000:00401b9c	bf 70 32 40 00	movl \$0x403270, %edi	ASCII "But finding it and solving it are quite different..."
00000000:00401ba1	e8 ba f4 ff ff	callq bomb!puts@plt	
00000000:00401ba6	b8 00 00 00 00	movl \$0, %eax	
00000000:00401bab	e8 3a fc ff ff	callq bomb!secret_phase	
00000000:00401bb0	eb be	jmp 0x401b70	
00000000:00401bb3	eb 77	jmp 0x401b70	end

这里可以看到他一些字符串提示，显然这里是秘密阶段的入口。然后看一下代码，首先他把 6 和某一个值做了比较，应该是一个全局变量，检测我们通过的阶段数。如果通过六关，则进行一系列检测，分析略，总之我们应该在第四阶段附带一个 DrEvil 的字符串。

分析隐藏阶段的代码。首先可以看出我们需要输入一个数字，然后他将这个数字和一个地址作为参数传入到 `fun7`，然后检测返回值是否为 2，若不为 2 则爆炸。我们先看一下这个地址 `0X4050F0` 的内容。



```

graph TD
    Root((5040  
24)) --> Node1((5110  
8))
    Root --> Node2((5130  
32))
    Node1 --> Node3((5190  
6))
    Node1 --> Node4((5150  
16))
    Node2 --> Node5((5170  
20))
    Node2 --> Node6((5160  
66))
    Node3 --> Node7((5140  
1))
    Node3 --> Node8((5250  
7))
    Node4 --> Node9((5270  
14))
    Node4 --> Node10((5230  
23))
    Node5 --> Node11((5180  
28))
    Node5 --> Node12((5240  
24))
    Node6 --> Node13((5210  
63))
    Node6 --> Node14((5260  
29))
  
```

然后他要求最后结果是 2,那么只有 0 1 2 这个序列符合结果,因此应该在 5150 这个节点里找到,这样跳到 5110 节点得到 1,再跳到根得到 2.然后昂他这里有一

个注意点是节点里的数字是十六进制数字，因此应该进行一个到十进制的转换， $0x16=0d22$ ，所以我们应该输入 22.

至此 炸弹全部拆除~ 🎉🎉

```
dqv587@ubuntu:~/Documents/Code/LAB2/bomb315$ ./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!
```

另外，我还看了看其他没有看到的代码。发现了一个 `sig_handler` 函数，看了看内容，发现可以用 `ctrl+c` 来发送中断异常，从而中止这个炸弹。

```
dqv587@ubuntu:~/Documents/Code/LAB2/bomb315$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
^CSo you think you can stop the bomb with ctrl-c, do you?
Well...OK. :-)
```

此外还发现一个 `submit` 的函数，他链接一个服务器，不过 IP 地址已经被改掉了，应该是原版实验有一个汇报学生情况做统计的函数。

## 第 4 章 总结

### 4.1 请总结本次实验的收获

首先熟悉了汇编语言，能够从汇编代码中反推出函数的操作。然后我仔细看了一下。Plt 部分，也对动态链接有了更好的认识，另外也对程序的内存分布有了认识，比如代码区、只读区、可写区等。

### 4.2 请给出对本次实验内容的建议

没啥建议。

注：本章为酌情加分项。

## 参考文献

- [1] Kernighan B W, Ritchie D M. The C Programming Language[M]. 2. Dennis Ritchie & Bell Labs, / June 2018.
- [2] Bryant R E, David R. O'Hallaron. Computer Systems a Programmer's Perspective[M]. 3rd. Carnegie Mellon University, 2016.