

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 1190200526

班 级 1903002

学 生 沈城有

指 导 教 师 郑贵滨

实 验 地 点 G709

实 验 日 期 2021.04.23

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	- 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	- 6 -
3.3 阶段 3 的破解与分析	- 7 -
3.4 阶段 4 的破解与分析	- 8 -
3.5 阶段 5 的破解与分析	- 10 -
3.6 阶段 6 的破解与分析	- 11 -
3.7 阶段 7 的破解与分析(隐藏阶段).....	- 14 -
第 4 章 总结	- 15 -
4.1 请总结本次实验的收获	- 15 -
4.2 请给出对本次实验内容的建议	- 15 -
参考文献.....	- 16 -

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、O2、O3，-m32/m64。再次查看生成的汇编语言与原来的区别。

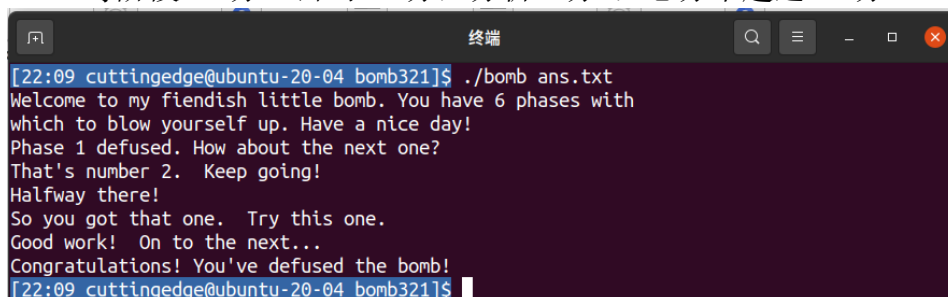
注意 O1 之后无栈帧，EBP 做别的用途。-fno-omit-frame-pointer 加上栈指针。

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 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



```

[22:09 cuttingedge@ubuntu-20-04 bomb321]$ ./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...
Congratulations! You've defused the bomb!
[22:09 cuttingedge@ubuntu-20-04 bomb321]$

```

3.1 阶段 1 的破解与分析

密码：I am for medical liability at the federal level.

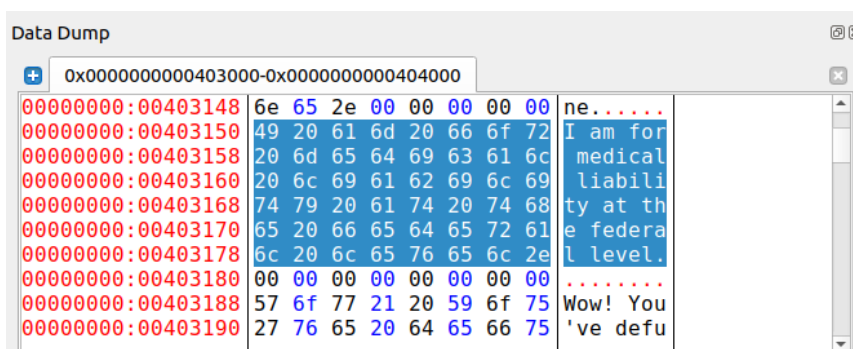
破解过程：

下图为 phase_1 函数的汇编指令：

306	0000000000004013f9	<phase_1>:		
307	4013f9:	55	push	%rbp
308	4013fa:	48 89 e5	mov	%rsp,%rbp
309	4013fd:	be 50 31 40 00	mov	\$0x403150,%esi
310	401402:	e8 2e 04 00 00	callq	401835 <strings_not_equal>
311	401407:	85 c0	test	%eax,%eax
312	401409:	75 02	jne	40140d <phase_1+0x14>
313	40140b:	5d	pop	%rbp
314	40140c:	c3	retq	
315	40140d:	e8 1f 05 00 00	callq	401931 <explode_bomb>
316	401412:	eb f7	jmp	40140b <phase_1+0x12>

可知此函数将输入字符串（寄存器 RDI 指向的字符串）与寄存器 RSI 指向的字符串进行比较（利用 string_not_equal 函数），不相同（返回值不为 0）则调用 explode_bomb 函数，导致炸弹爆炸。

通过 EDB 查看对应内存位置答案字符串如下：



Address	Hex	ASCII
00000000:00403148	6e 65 2e 00 00 00 00 00	ne.....
00000000:00403150	49 20 61 6d 20 66 6f 72	I am for
00000000:00403158	20 6d 65 64 69 63 61 6c	medical
00000000:00403160	20 6c 69 61 62 69 6c 69	liabili
00000000:00403168	74 79 20 61 74 20 74 68	ty at th
00000000:00403170	65 20 66 65 64 65 72 61	e federa
00000000:00403178	6c 20 6c 65 76 65 6c 2e	l level.
00000000:00403180	00 00 00 00 00 00 00 00
00000000:00403188	57 6f 77 21 20 59 6f 75	Wow! You
00000000:00403190	27 76 65 20 64 65 66 75	've defu

3.2 阶段 2 的破解与分析

密码：0 1 1 2 3 5

破解过程：

下图为 phase_2 函数的汇编指令及分析：

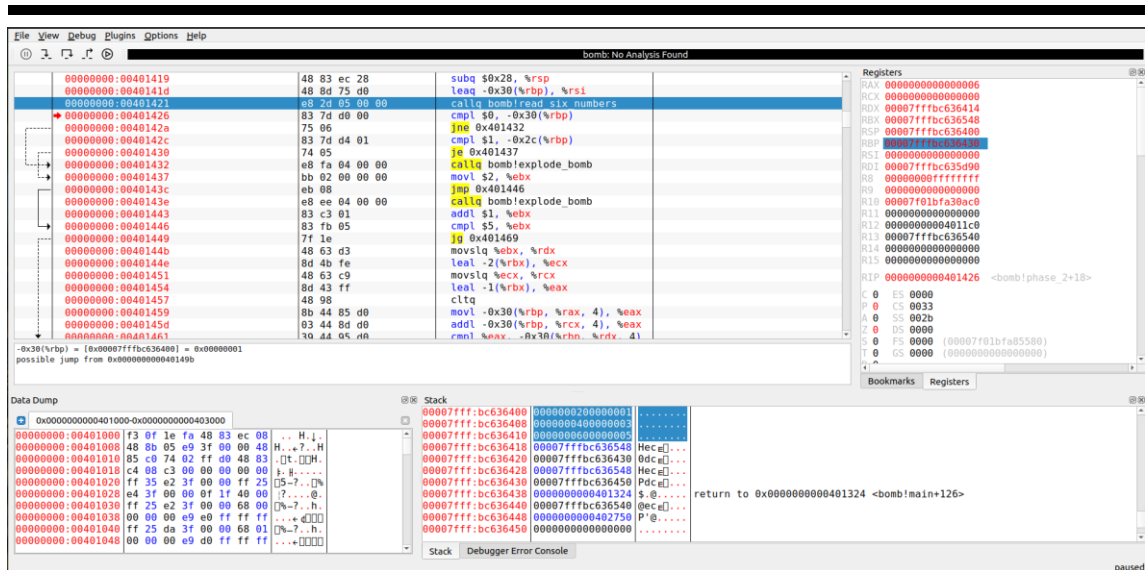
```
0000000000401414 <phase_2>:
401414: 55                push    %rbp
401415: 48 89 e5          mov     %rsp,%rbp
401418: 53                push    %rbx
401419: 48 83 ec 28       sub     $0x28,%rsp
40141d: 48 8d 75 d0       lea     -0x30(%rbp),%rsi
401421: e8 2d 05 00 00    callq   401953 <read_six_numbers> #读入6个数字，读入个数少于6直接引爆
401426: 83 7d d0 00       cmpl    $0x0,-0x30(%rbp)          #第一个数字为0
40142a: 75 06             jne     401432 <phase_2+0x1e>
40142c: 83 7d d4 01       cmpl    $0x1,-0x2c(%rbp)          #第二个数字为1
401430: 74 05             je      401437 <phase_2+0x23>
401432: e8 fa 04 00 00    callq   401931 <explode_bomb>
401437: bb 02 00 00 00    mov     $0x2,%ebx                  #%ebx = 2
40143c: eb 08             jmp     401446 <phase_2+0x32>
40143e: e8 ee 04 00 00    callq   401931 <explode_bomb>
401443: 83 c3 01          add     $0x1,%ebx
401446: 83 fb 05          cmp     $0x5,%ebx                  #循环终止条件
401449: 7f 1e             jg      401469 <phase_2+0x55>      #循环结束前未爆炸则正常返回
40144b: 48 63 d3          movslq  %ebx,%rdx
40144e: 8d 4b fe          lea     -0x2(%rbx),%ecx             #%ebx - 2 -> %ecx
401451: 48 63 c9          movslq  %ecx,%rcx
401454: 8d 43 ff          lea     -0x1(%rbx),%eax             #%ebx - 1 -> %eax
401457: 48 98             cltq
401459: 8b 44 85 d0       mov     -0x30(%rbp,%rax,4),%eax     #取数
40145d: 03 44 8d d0       add     -0x30(%rbp,%rcx,4),%eax     #将取的数与其前一个相加
401461: 39 44 95 d0       cmp     %eax,-0x30(%rbp,%rdx,4)     #结果与后一个数比较
401465: 74 dc             je      401443 <phase_2+0x2f>      #相等继续循环
401467: eb d5             jmp     40143e <phase_2+0x2a>      #不相等则爆炸
401469: 48 83 c4 28       add     $0x28,%rsp
40146d: 5b                pop     %rbx
40146e: 5d                pop     %rbp
40146f: c3                retq
```

下图为 read_six_numbers 函数的汇编指令及分析：

```
0000000000401953 <read_six_numbers>:
401953: 55                push    %rbp
401954: 48 89 e5          mov     %rsp,%rbp
401957: 48 89 f2          mov     %rsi,%rdx                  #0
40195a: 48 8d 4e 04       lea     0x4(%rsi),%rcx              #+4
40195e: 48 8d 46 14       lea     0x14(%rsi),%rax             #+20
401962: 50                push    %rax
401963: 48 8d 46 10       lea     0x10(%rsi),%rax             #+16
401967: 50                push    %rax
401968: 4c 8d 4e 0c       lea     0xc(%rsi),%r9               #+12
40196c: 4c 8d 46 08       lea     0x8(%rsi),%r8               #+8
401970: be 13 33 40 00    mov     $0x403313,%esi              #格式字符串: "%d %d %d %d %d %d"
401975: b8 00 00 00 00    mov     $0x0,%eax
40197a: e8 91 f7 ff ff    callq   401110 <__isoc99_sscanf@plt> #格式读入
40197f: 48 83 c4 10       add     $0x10,%rsp
401983: 83 f8 05          cmp     $0x5,%eax
401986: 7e 02             jle     40198a <read_six_numbers+0x37> #个数小于等于5, 引爆
401988: c9                leaveq  %rsi,%rdi
401989: c3                retq
40198a: e8 a2 ff ff ff    callq   401931 <explode_bomb>
```

如下图，通过简单的输入测试可确定读入后的具体保存位置：

计算机系统实验报告



第 1 个数→第 6 个数：(%rbp-0x30)→(%rbp)

由以上分析可得，程序中有一个类似于检测斐波那契数列的循环体，且可知前两个数字分别为 0、1，之后四个数字即可推出，分别为 0+1=1，1+1=2，1+2=3，2+3=5。

3.3 阶段 3 的破解与分析

密码：40（注：其中一种可行数字组合）

破解过程：

下图为 phase_3 函数部分汇编代码及分析：

401480:	be 1f 33 40 00	mov \$0x40331f,%esi	#格式字符串"%d %d"，说明应该输入两个数
401485:	b8 00 00 00 00	mov \$0x0,%eax	#设第一个数为input1，第二个数为input2
40148a:	e8 81 fc ff ff	callq 401110 <_isoc99_sscanf@plt>	
40148f:	83 f8 01	cmpl \$0x1,%eax	
401492:	7e 11	jle 4014a5 <phase_3+0x35>	#输入少于2个爆炸
401494:	8b 45 fc	mov -0x4(%rbp),%eax	#取input1至%eax
401497:	83 f8 07	cmpl \$0x7,%eax	
40149a:	77 7b	ja 401517 <phase_3+0xa7>	#input1大于7则爆炸
40149c:	89 c0	mov %eax,%eax	
40149e:	ff 24 c5 c0 31 40 00	jmpq *0x4031c0(%rax,8)	#基址-变址法寻址，0x4031c0应该是一个跳转表的起始位置
4014a5:	e8 87 04 00 00	callq 401931 <explode_bomb>	
4014aa:	e8 b8 00 00 00	jmp 401494 <phase_3+0x24>	
4014ac:	b8 00 00 00 00	mov \$0x0,%eax	#input1 = 1, %eax = 0 - 0x1d7 + 0x22e - 0x256
4014b1:	2d d7 01 00 00	sub \$0x1d7,%eax	#input1 = 0, %eax = 0x127 - 0x1d7 + 0x22e - 0x256
4014b6:	05 2e 02 00 00	add \$0x22e,%eax	#input1 = 2, %eax = 0 + 0x22e - 0x256
4014bb:	2d 56 02 00 00	sub \$0x256,%eax	#input1 = 3, %eax = 0 - 0x256
4014c0:	05 56 02 00 00	add \$0x256,%eax	#input1 = 4, %eax = 0
4014c5:	2d 56 02 00 00	sub \$0x256,%eax	#input1 = 5, %eax = 0 - 0x256
4014ca:	05 56 02 00 00	add \$0x256,%eax	#input1 = 6, %eax = 0
4014cf:	2d 56 02 00 00	sub \$0x256,%eax	#input1 = 7, %eax = 0 - 0x256
4014d4:	83 7d fc 05	cmpl \$0x5,-0x4(%rbp)	#要求input1 <= 5
4014d8:	7f 05	jg 4014df <phase_3+0x6f>	
4014da:	39 45 f8	cmpl %eax,-0x8(%rbp)	#要求input2 = %eax
4014dd:	74 05	je 4014e4 <phase_3+0x74>	
4014df:	e8 4d 04 00 00	callq 401931 <explode_bomb>	
4014e4:	c9	leaveq	
4014e5:	c3	retq	#以下是跳转表跳转到的一些目标位置，均为对%eax赋值再跳至上方，此处略

如下图，输入 1 2 进行测试，可知阶段 3 读入两个数，第一个数保存于

计算机系统实验报告

```

401572:    be 1f 33 40 00      mov     $0x40331f,%esi      #省略部分为与phase_3同理的数据读入
401577:    b8 00 00 00 00      mov     $0x0,%eax          #设第一个数为input1, 第二个数为input2
40157c:    e8 8f fb ff ff      callq   401110 <__isoc99_sscanf@plt>
401581:    83 f8 02             cmp     $0x2,%eax
401584:    75 0c               jne     401592 <phase_4+0x30>    #依然是要求输入两个数
401586:    8b 45 fc             mov     -0x4(%rbp),%eax
401589:    85 c0             test    %eax,%eax
40158b:    78 05             js      401592 <phase_4+0x30>    #input1如果是负数则引爆
40158d:    83 f8 0e             cmp     $0xe,%eax          #input1 <= 14
401590:    7e 05             jle     401597 <phase_4+0x35>
401592:    e8 9a 03 00 00      callq   401931 <explode_bomb>
401597:    ba 0e 00 00 00      mov     $0xe,%edx          #%edx = 14
40159c:    be 00 00 00 00      mov     $0x0,%esi          #%esi = 0
4015a1:    8b 7d fc             mov     -0x4(%rbp),%edi    #%edi = input1
4015a4:    e8 7a ff ff ff      callq   401523 <func4>        #调用func4函数, 以上几个寄存器可视为传递参数
4015a9:    83 f8 23             cmp     $0x23,%eax          #func4函数返回值应为35
4015ac:    75 06             jne     4015b4 <phase_4+0x52>
4015ae:    83 7d f8 23             cmpl    $0x23,-0x8(%rbp)
4015b2:    74 05             je      4015b9 <phase_4+0x57>    #input2 = 35
4015b4:    e8 78 03 00 00      callq   401931 <explode_bomb>
4015b9:    c9                 leaveq  %eax,%edi
4015ba:    c3                 retq

```

下图为递归调用的 func4 函数汇编代码及分析：

```

000000000401523 <func4>:                                #由phase_4函数调用
401523:    55                 push    %rbp
401524:    48 89 e5           mov     %rsp,%rbp
401527:    53                 push    %rbx
401528:    48 83 ec 08        sub     $0x8,%rsp
40152c:    89 d0             mov     %edx,%eax          #%eax = %edx
40152e:    29 f0             sub     %esi,%eax          #%eax -= %esi
401530:    89 c3             mov     %eax,%ebx          #%ebx = %eax
401532:    c1 eb 1f           shr     $0x1f,%ebx         #%ebx >= 31 (逻辑右移, 相当于取符号位)
401535:    01 c3             add     %eax,%ebx          #%ebx += %eax
401537:    d1 fb           sar     %ebx               #%ebx >= 1 (算术右移)
401539:    01 f3             add     %esi,%ebx          #%ebx += %esi
40153b:    39 fb             cmp     %edi,%ebx          #比较%ebx与%edi
40153d:    7f 0b             jg      40154a <func4+0x27>    #%ebx > %edi跳转
40153f:    7c 15             jl      401556 <func4+0x33>    #%ebx < %edi跳转
401541:    89 d8             mov     %ebx,%eax          #%eax = %ebx
401543:    48 83 c4 08        add     $0x8,%rsp
401547:    5b                 pop     %rbx
401548:    5d                 pop     %rbp
401549:    c3                 retq
40154a:    8d 53 ff           lea     -0x1(%rbx),%edx     #%edx = %ebx - 1
40154d:    e8 d1 ff ff ff      callq   401523 <func4>        #递归调用自身
401552:    01 c3             add     %eax,%ebx          #%ebx += %eax
401554:    eb eb             jmp     401541 <func4+0x1e>    #跳转
401556:    8d 73 01           lea     0x1(%rbx),%esi     #%esi = %ebx + 1
401559:    e8 c5 ff ff ff      callq   401523 <func4>        #递归调用自身
40155e:    01 c3             add     %eax,%ebx          #%ebx += %eax
401560:    eb df             jmp     401541 <func4+0x1e>    #跳转

```

根据分析，我编写了 C 语言程序模拟这一过程并最终得到密码，模拟程序部分代码及运行结果见下图：

```

C sim_phase_4.c U X
VSCode > HITCS > HITCS\ICS_CSAPP > HIT_Labs > Lab3 > bomb321 > C sim_phase_4.c > func4(int, int, int)

1 //模拟phase_4的处理过程
2 #include <stdio.h>
3
4 int func4(int edx, int edi, int esi)
5 {
6     int eax, ebx;
7     eax = edx - esi;
8     ebx = eax;
9     ebx >>= 1;
10    ebx += esi;
11    if (ebx > edi)
12    {
13        edx = ebx - 1;
14        ebx += func4(edx, edi, esi);
15    }
16    else if (ebx < edi)
17    {
18        esi = ebx + 1;
19        ebx += func4(edx, edi, esi);
20    }
21    eax = ebx;
22    return eax;
23 }

[18:59 cuttingedge@ubuntu-20-04 bomb321]$ gcc sim_phase_4.c -o sim_phase_4
[18:59 cuttingedge@ubuntu-20-04 bomb321]$ ./sim_phase_4
Wrong:[0 35]
Wrong:[1 35]
Wrong:[2 35]
Wrong:[3 35]
Wrong:[4 35]
Wrong:[5 35]
Wrong:[6 35]
Wrong:[7 35]
Password:[8 35]
Wrong:[9 35]
Wrong:[10 35]
Wrong:[11 35]
Wrong:[12 35]
Wrong:[13 35]
Wrong:[14 35]
[18:59 cuttingedge@ubuntu-20-04 bomb321]$

```

3.5 阶段 5 的破解与分析

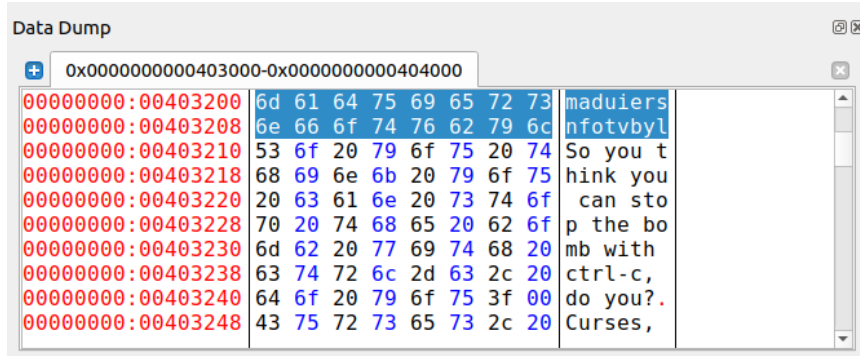
密码: BELDOG

破解过程:

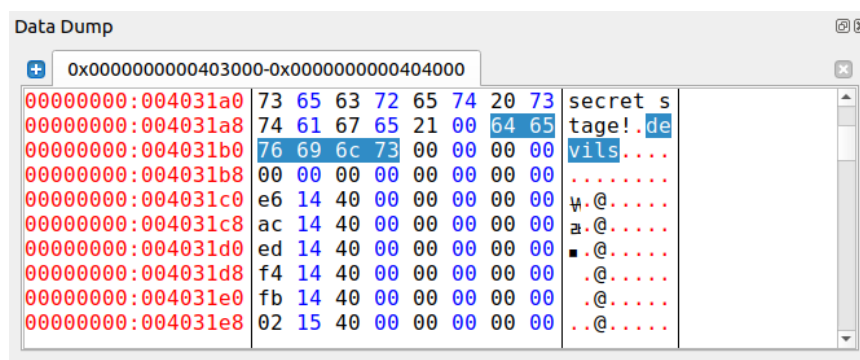
下图为 phase_5 函数部分汇编代码及分析:

4015c4:	48 89 fb	mov %rdi,%rbx	#将输入字符串的地址传给string_length函数
4015c7:	e8 55 02 00 00	callq 401821 <string_length>	
4015cc:	83 f8 06	cmp \$0x6,%eax	#输入字符串的长度要求为6
4015cf:	75 24	jne 4015f5 <phase_5+0x3a>	
4015d1:	b8 00 00 00 00	mov \$0x0,%eax	#循环变量初始化
4015d6:	83 f8 05	cmp \$0x5,%eax	#循环终止条件 (遍历完整个字符串)
4015d9:	7f 21	jg 4015fc <phase_5+0x41>	
4015db:	48 63 c8	movslq %eax,%rcx	
4015de:	0f b6 14 0b	movzbl (%rbx,%rcx,1),%edx	#依次取字符串中每个字符, 以ascii数值保存于%edx
4015e2:	83 e2 0f	and \$0xf,%edx	##edx = %edx & 0xf (低四位)
4015e5:	0f b6 92 00 32 40 00	movzbl 0x403200(%rdx),%edx	#将处理后的值作为偏移量, 访问一个字符数组
4015ec:	88 54 0d e9	mov %dl,-0x17(%rbp,%rcx,1)	#将取出的字符逐个放置
4015f0:	83 c0 01	add \$0x1,%eax	#循环变量增一
4015f3:	eb e1	jmp 4015d6 <phase_5+0x1b>	#检查循环条件
4015f5:	e8 37 03 00 00	callq 401931 <explode_bomb>	
4015fa:	eb d5	jmp 4015d1 <phase_5+0x16>	
4015fc:	c6 45 ef 00	movb \$0x0,-0x11(%rbp)	#向末尾放置一个'\0'
401600:	be ae 31 40 00	mov \$0x4031ae,%esi	#指向一个字符串: "devils"
401605:	48 8d 7d e9	lea -0x17(%rbp),%rdi	
401609:	e8 27 02 00 00	callq 401835 <strings_not_equal>	#检查新组合成的字符串与要求的是否相同
40160e:	85 c0	test %eax,%eax	
401610:	75 07	jne 401619 <phase_5+0x5e>	
401612:	48 83 c4 18	add \$0x18,%rsp	
401616:	5b	pop %rbx	
401617:	5d	pop %rbp	
401618:	c3	retq	
401619:	e8 13 03 00 00	callq 401931 <explode_bomb>	
40161e:	eb f2	jmp 401612 <phase_5+0x57>	

下图为用于组合要求字符串所用到的字符数组:



下图为要求组合得到的字符串：



由以上可知，我们需要构造一个 6 个字符的字符串，用其 ascii 码与 0xf 做与运算后获得的偏移量取出字符数组里的字符构成目标字符串。

过程如下：

目标字符串字符	d	e	v	i	l	s
偏移量	2	5	12	4	15	7
输入字符串字符编码低四位	0010	0101	1100	0100	1111	0111
一种可行的编码	64+2	64+5	64+12	64+4	64+15	64+7
对应字符	B	E	L	D	O	G

3.6 阶段 6 的破解与分析

密码：2 3 5 6 1 4

破解过程：

由 phase_6 函数汇编代码开头部分调用 read_six_numbers 函数可以看出，输入格式类似于 phase_2，即 6 个整数。

分析 phase_6 函数的汇编代码，发现其主要由四个循环体构成，下图为第一个循环体的汇编代码及分析：

```

#循环体1开始
401636: 41 bc 00 00 00 00    mov     $0x0,%r12d      #外层循环变量初始化
40163c: eb 29                jmp     401667 <phase_6+0x47> #跳转到外层循环条件检测
40163e: e8 ee 02 00 00      callq   401931 <explode_bomb>
401643: eb 37                jmp     40167c <phase_6+0x5c>
401645: 83 c3 01            add     $0x1,%ebx        #内层循环变量更新
401648: 83 fb 05            cmp     $0x5,%ebx        #内层循环终止条件
40164b: 7f 17              jg      401664 <phase_6+0x44>
40164d: 49 63 c4            movslq  %r12d,%rax
401650: 48 63 d3            movslq  %ebx,%rdx
401653: 8b 7c 95 c0        mov     -0x40(%rbp,%rdx,4),%edi #内层循环逐个取数
401657: 39 7c 85 c0        cmp     %edi,-0x40(%rbp,%rax,4) #要求任两个数字不相等
40165b: 75 e8              jne     401645 <phase_6+0x25>
40165d: e8 cf 02 00 00      callq   401931 <explode_bomb>
401662: eb e1              jmp     401645 <phase_6+0x25>
401664: 45 89 ec            mov     %r13d,%r12d      #外层循环变量更新
401667: 41 83 fc 05        cmp     $0x5,%r12d      #外层循环终止条件（遍历完6个数字）
40166b: 7f 19              jg      401686 <phase_6+0x66>
40166d: 49 63 c4            movslq  %r12d,%rax
401670: 8b 44 85 c0        mov     -0x40(%rbp,%rax,4),%eax #外层循环逐个取数
401674: 83 e8 01            sub     $0x1,%eax
401677: 83 f8 05            cmp     $0x5,%eax        #取的数减一与5比较
40167a: 77 c2              ja      40163e <phase_6+0x1e> #说明数字大于6或小于1会爆炸（ja无符号比较）
40167c: 45 8d 6c 24 01      lea     0x1(%r12),%r13d
401681: 44 89 eb            mov     %r13d,%ebx        #%ebx = 外层循环变量 + 1（内层循环变量初始化）
401684: eb c2              jmp     401648 <phase_6+0x28>
#循环体1结束

```

类似于以下的 C 代码功能：

```

for (int i = 0; i < 6; ++i)
{
    if (num[i] > 6 || num[i] < 0)
        return -1; //炸弹爆炸
    for (int j = i + 1; j < 6; ++j)
    {
        if (num[i] == num[j])
            return -1; //炸弹爆炸
    }
}

```

故这个循环体的作用为判断 6 个数是否互不相同，且取值范围是否为 $1 \leq x \leq 6$ ，若不是则爆炸。

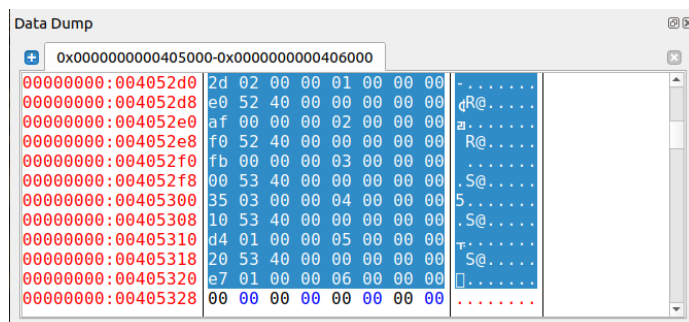
下图为 phase_6 函数的第二个循环体的汇编代码及分析：

```

#循环体2开始
401686: be 00 00 00 00    mov     $0x0,%esi        #循环变量初始化
40168b: eb 08            jmp     401695 <phase_6+0x75> #跳至循环条件检测
40168d: 48 89 54 cd 90    mov     %rdx,-0x70(%rbp,%rcx,8) #将地址放在对应数字序号位置
401692: 83 c6 01            add     $0x1,%esi        #循环变量更新
401695: 83 fe 05            cmp     $0x5,%esi        #说明此循环体依然是遍历6个数字
401698: 7f 1c              jg      4016b6 <phase_6+0x96> #跳出循环
40169a: b8 01 00 00 00    mov     $0x1,%eax        #%eax = 1
40169f: ba d0 52 40 00    mov     $0x4052d0,%edx    #一个地址，保存了一些有规律的数据（链表结构）
4016a4: 48 63 ce            movslq  %esi,%rcx
4016a7: 39 44 8d c0        cmp     %eax,-0x40(%rbp,%rcx,4) #逐个取数，与%eax比较
4016ab: 7e e0            jle     40168d <phase_6+0x6d> #<=eax跳转
4016ad: 48 8b 52 08        mov     0x8(%rdx),%rdx    #未跳转则取当前结点next指向结点的地址
4016b1: 83 c0 01            add     $0x1,%eax        #%eax += 1
4016b4: eb ee            jmp     4016a4 <phase_6+0x84>
#循环体2结束

```

下图为以上分析中所提到的链表结构：



可以看成 4 字节数值+4 字节序号(1~6)+8 字节地址(指向下一个结点)。
故此链表可如下表示：

$Head(0x4052d0) \rightarrow [0x22d\ 1] \rightarrow [0xaf\ 2] \rightarrow [0xfb\ 3] \rightarrow [0x335\ 4]$
 $\rightarrow [0x1d4\ 5] \rightarrow [0x1e7\ 6] \rightarrow NULL$

由此分析，第二个循环体的作用在于按照输入的 6 个数取对应标号的链表结点 next 地址，重新排列放置于栈的另一块连续空间里。

下图为 phase_6 函数的第三个循环体的汇编代码及分析：

```

#循环体3开始
4016b6: 48 8b 5d 90      mov     -0x70(%rbp),%rbx      #取放置的第一个结点地址
4016ba: 48 89 d9         mov     %rbx,%rcx
4016bd: b8 01 00 00 00   mov     $0x1,%eax             #循环变量初始化
4016c2: eb 12          jmp     4016d6 <phase_6+0xb6>
4016c4: 48 63 d0         movslq  %eax,%rdx
4016c7: 48 8b 54 d5 90   mov     -0x70(%rbp,%rdx,8),%rdx #链表访问next域指向的结点
4016cc: 48 89 51 08      mov     %rdx,0x8(%rcx)        #更新next域（重新连接链表）
4016d0: 83 c0 01         add     $0x1,%eax
4016d3: 48 89 d1         mov     %rdx,%rcx
4016d6: 83 f8 05        cmp     $0x5,%eax             #循环终止条件检测
4016d9: 7e e9          jle     4016c4 <phase_6+0xa4>
#循环体3结束

```

循环体 3 主要功能是重新链接整个链表。

下图为 phase_6 函数的第四个循环体的汇编代码及分析：

```

4016db: 48 c7 41 08 00 00 00  movq    $0x0,0x8(%rcx)      #设置新的链表尾部NULL
4016e2: 00
#循环体4开始
4016e3: 41 bc 00 00 00 00     mov     $0x0,%r12d          #循环变量初始化
4016e9: eb 08                jmp     4016f3 <phase_6+0xd3>
4016eb: 48 8b 5b 08          mov     0x8(%rbx),%rbx
4016ef: 41 83 c4 01          add     $0x1,%r12d
4016f3: 41 83 fc 04          cmp     $0x4,%r12d          #循环终止条件检测
4016f7: 7f 11                jg      40170a <phase_6+0xea>
4016f9: 48 8b 43 08          mov     0x8(%rbx),%rax      #取结点next域指向的结点地址
4016fd: 8b 00                mov     (%rax),%eax         #取结点中保存的数
4016ff: 39 03                cmp     %eax,(%rbx)         #与前一结点中数比较
401701: 7e e8                jle     4016eb <phase_6+0xcb> #要求新的链表数值递增
401703: e8 29 02 00 00      callq   401931 <explode_bomb>
401708: eb e1                jmp     4016eb <phase_6+0xcb>
#循环体4结束

```

循环体 4 的主要功能是检测新链接的链表数值部分是否递增，函数至此主要功能已经结束。

根据以上分析，将链表各点按数值升序得序号：2 3 5 6 1 4。

此数字序列即为密码。

3.7 阶段 7 的破解与分析(隐藏阶段)

密码：

破解过程：

第 4 章 总结

4.1 请总结本次实验的收获

1. 本次实验学会了 Code::Blocks、EDB 的反汇编调试方法；
2. 深化了对汇编指令的理解；
3. 锻炼了分析汇编指令和汇编程序过程的能力；
4. 理解了汇编语言中链表、数组递归、循环等结构的表示及实现。

4.2 请给出对本次实验内容的建议

建议课件补充 EDB 使用和配置的入门知识。例如在完成 EDB 的安装后，如何将 EDB 默认汇编指令格式从 Intel 修改至 AT&T，以及如何修改字号、如何查找函数名、如何设置断点及断点的类型与区别等等。

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>.