

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 1170301027

班 级 1703010

学 生 冯帅

指 导 教 师 史先俊

实 验 地 点 G712

实 验 日 期 20181021

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.....	错误! 未定义书签。
1.2.2 软件环境.....	错误! 未定义书签。
1.2.3 开发工具.....	错误! 未定义书签。
1.3 实验预习.....	- 3 -
第 2 章 实验环境建立.....	- 12 -
2.1 UBUNTU 下 CODEBLOCKS 反汇编（10 分）	- 12 -
2.2 UBUNTU 下 EDB 运行环境建立（10 分）	- 12 -
第 3 章 各阶段炸弹破解与分析.....	- 14 -
3.1 阶段 1 的破解与分析.....	- 14 -
3.2 阶段 2 的破解与分析.....	- 15 -
3.3 阶段 3 的破解与分析.....	- 17 -
3.4 阶段 4 的破解与分析.....	- 20 -
3.5 阶段 5 的破解与分析.....	- 23 -
3.6 阶段 6 的破解与分析.....	- 25 -
3.7 阶段 7 的破解与分析(隐藏阶段).....	- 28 -
第 4 章 总结.....	- 30 -
4.1 请总结本次实验的收获.....	- 30 -
4.2 请给出对本次实验内容的建议.....	- 30 -
参考文献.....	- 31 -

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

Intel Core i7-7700HQ 2.81GHz, 8GB RAM, 128GB SSD

1.2.2 软件环境

Windows10 64 位以上; Vmware 11 以上; Ubuntu 16.04 LTS 64 位/优麒麟 64 位

1.2.3 开发工具

Visual Studio 2010 64 位; CodeBlocks 64 位; vi/vim/gedit+gcc

1.3 实验预习

(a)sample.c 文件生成.out 文件:

gcc 操作时修改编译选项, -O -O1 -O2 -O3 -m32 -m64 分别生成汇编语言

Ubuntu - VMware Workstation

文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)

Ubuntu

活动 终端 Tuesday 03 : 56 英

fs1170301027@ubuntu: ~/hitics

文件(E) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
-rwxrwxrwx 1 root root 17950 Sep 18 08:49 hello.i
-rwxrwxrwx 1 root root 10760 Sep 29 19:43 hellolinux
-rwxrwxrwx 1 root root 8296 Sep 16 05:41 hellolinux2
-rwxrwxrwx 1 root root 70 Sep 16 05:39 hellolinux.c
-rwxrwxrwx 1 root root 5840 Sep 29 19:43 hellolinux.o
-rwxrwxrwx 1 root root 1552 Sep 18 08:50 hello.o
-rwxrwxrwx 1 root root 8296 Sep 18 08:50 hello.out
-rwxrwxrwx 1 root root 496 Sep 18 08:49 hello.s
-rwxrwxrwx 1 root root 69 Sep 23 01:55 hollowin.c
-rwxrwxrwx 1 root root 2035 Oct 16 03:14 sample.c
-rwxrwxrwx 1 root root 0 Sep 18 07:55 新建文本文档.txt
```

```
fs1170301027@ubuntu:~/hitics$ gcc -S sample.c -o sample.s
sample.c: In function 'CopyString':
sample.c:13:2: warning: implicit declaration of function 'strcpy' [-Wimplicit-f
unction-declaration]
  strcpy(buf, s);
sample.c:13:2: warning: incompatible implicit declaration of built-in function
'strcpy'
sample.c:13:2: note: include '<string.h>' or provide a declaration of 'strcpy'
fs1170301027@ubuntu:~/hitics$ gedit sample.c
^C
fs1170301027@ubuntu:~/hitics$ gedit sample.c
^C
fs1170301027@ubuntu:~/hitics$ gcc -S sample.c -o sample.s
fs1170301027@ubuntu:~/hitics$ gcc -c sample.s -o sample.o
fs1170301027@ubuntu:~/hitics$ gcc sample.o -o sample.out
fs1170301027@ubuntu:~/hitics$
```

要将输入定向到该虚拟机，请将鼠标指针移入其中或按 Ctrl+G。

```
fs1170301027@ubuntu:~/hitics$ gcc -S -O1 sample.c -o sample01.s
fs1170301027@ubuntu:~/hitics$ gcc -S -O2 sample.c -o sample02.s
fs1170301027@ubuntu:~/hitics$ gcc -S -O3 sample.c -o sample03.s
fs1170301027@ubuntu:~/hitics$ gcc -S -m32 sample.c -o samplem32.s
fs1170301027@ubuntu:~/hitics$ gcc -S -m64 sample.c -o samplem64.s
```

```
-rwxrwxrwx 1 root root 2054 10月 16 03:54 sample.c
-rwxrwxrwx 1 root root 11983 10月 25 07:00 samplem32.s
-rwxrwxrwx 1 root root 9874 10月 25 07:00 samplem64.s
-rwxrwxrwx 1 root root 5496 10月 16 03:55 sample.o
-rwxrwxrwx 1 root root 5074 10月 25 06:59 sample01.s
-rwxrwxrwx 1 root root 5610 10月 25 06:59 sample02.s
-rwxrwxrwx 1 root root 8105 10月 25 06:59 sample03.s
-rwxrwxrwx 1 root root 12936 10月 16 03:56 sample.out
-rwxrwxrwx 1 root root 9868 10月 16 03:54 sample.s
```

比较不同编译选项下生成的汇编语言代码，发现从-O1 开始相比-O 有优化，而后依次优化程度加深，就 `strcpy` 段代码来说，下面两幅图是有优化和没优化的区别，当然这种优化并不仅仅表现在代码行数上，不同的优化选项其代码的执行效率上也有很大不同，同时可读性也不一样；而且并不是每一个优化都会缩短代码

行数,

```
CopyString:
.LFB5:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $48, %rsp
    movq    %rdi, -40(%rbp)
    movq    %fs:40, %rax
    movq    %rax, -8(%rbp)
    xorl    %eax, %eax
    movq    -40(%rbp), %rdx
    leaq    -32(%rbp), %rax
    movq    %rdx, %rsi
    movq    %rax, %rdi
    call    strcpy@PLT
    nop
    movq    -8(%rbp), %rax
    xorq    %fs:40, %rax
    je      .L2
    call    __stack_chk_fail@PLT

CopyString:
.LFB52:
    .cfi_startproc
    subq    $40, %rsp
    .cfi_def_cfa_offset 48
    movq    %rdi, %rsi
    movq    %fs:40, %rax
    movq    %rax, 24(%rsp)
    xorl    %eax, %eax
    movq    %rsp, %rdi
    movl    $20, %edx
    call    __strcpy_chk@PLT
    movq    24(%rsp), %rax
    xorq    %fs:40, %rax
    jne     .L7
    addq    $40, %rsp
    .cfi_remember_state
    .cfi_def_cfa_offset 8
    ret
```

而-m32 和-m64 有明显的不同，从精简角度来看的话-m64 生成的代码更简洁，醒目，但是其最大的不同还是在寻址上，对于寄存器的间接寻址，好像二者方式相同，定位的寄存器却不一样，似乎看上去-m64 生成的汇编语言代码可读性更强。。以上是我个人的见解。。

CopyString:

.LFB5:

```

.cfi_startproc
pushl   %ebp
.cfi_def_cfa_offset 8
.cfi_offset 5, -8
movl    %esp, %ebp
.cfi_def_cfa_register 5
pushl   %ebx
subl    $52, %esp
.cfi_offset 3, -12
call    __x86.get_pc_thunk.ax
addl    $GLOBAL_OFFSET_TABLE_, %eax
movl    8(%ebp), %edx
movl    %edx, -44(%ebp)
movl    %gs:20, %ecx
movl    %ecx, -12(%ebp)
xorl    %ecx, %ecx
subl    $8, %esp
pushl   -44(%ebp)
leal    -32(%ebp), %edx
pushl   %edx
movl    %eax, %ebx
call    strcpy@PLT
addl    $16, %esp
nop
movl    -12(%ebp), %eax
xorl    %gs:20, %eax
je      .L2
call    __stack_chk_fail_local

```

CopyString:

.LFB5:

```

.cfi_startproc
pushq   %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq    %rsp, %rbp
.cfi_def_cfa_register 6
subq    $48, %rsp
movq    %rdi, -40(%rbp)
movq    %fs:40, %rax
movq    %rax, -8(%rbp)
xorl    %eax, %eax
movq    -40(%rbp), %rdx
leaq    -32(%rbp), %rax
movq    %rdx, %rsi
movq    %rax, %rdi
call    strcpy@PLT
nop
movq    -8(%rbp), %rax
xorq    %fs:40, %rax
je      .L2
call    __stack_chk_fail@PLT

```

(b) 汇编语言代码及各段对应关系:

如下是我用 objdump 将 sample.out 反汇编重定向到文本文件中,之后打开的 txt 文件中, 各个函数之间的对应关系十分明朗。

计算机系统实验报告

```

00000000000007e1 <main>:
7e1: 55                push  %rbp
7e2: 48 89 e5          mov   %rsp,%rbp
7e5: 48 83 ec 60       sub   $0x60,%rsp
7e9: 64 48 8b 04 25 28 00 mov   %fs:0x28,%rax
7f0: 00 00
7f2: 48 89 45 f8       mov   %rax,-0x8(%rbp)
7f6: 31 c0             xor   %eax,%eax
7f8: 48 b8 30 30 30 30 31 movabs $0x313131313130303030,%rax
7ff: 31 31 31
802: 48 ba 32 32 32 32 33 movabs $0x333333333323232,%rdx
809: 33 33 33
80c: 48 89 45 b0       mov   %rax,-0x50(%rbp)
810: 48 89 55 b8       mov   %rdx,-0x48(%rbp)
814: 48 b8 34 34 34 34 35 movabs $0x3535353534343434,%rax
81b: 35 35 35
81e: 48 ba 36 36 36 36 37 movabs $0x3737373736363636,%rdx
825: 37 37 37
828: 48 89 45 c0       mov   %rax,-0x40(%rbp)
82c: 48 89 55 c8       mov   %rdx,-0x38(%rbp)
830: 48 b8 38 38 38 38 39 movabs $0x3939393938383838,%rax
837: 39 39 39
83a: 48 ba 61 61 61 61 62 movabs $0x6262626261616161,%rdx
841: 62 62 62
844: 48 89 45 d0       mov   %rax,-0x30(%rbp)
848: 48 89 55 d8       mov   %rdx,-0x28(%rbp)
84c: 48 b8 63 63 63 63 64 movabs $0x6464646463636363,%rax
853: 64 64 64
856: 48 ba 65 65 65 65 66 movabs $0x6666666665656565,%rdx
85d: 66 66 66
860: 48 89 45 e0       mov   %rax,-0x20(%rbp)
864: 48 89 55 e8       mov   %rdx,-0x18(%rbp)
868: c6 45 f0 00       movb  $0x0,-0x10(%rbp)
86c: 48 8d 45 b0       lea   -0x50(%rbp),%rax
870: 48 83 c0 28       add   $0x28,%rax
874: 48 89 45 a8       mov   %rax,-0x58(%rbp)
878: 48 8d 15 50 ff ff ff lea   -0xb0(%rip),%rdx      # 7cf <hacked>
87f: 48 8b 45 a8       mov   -0x58(%rbp),%rax
883: 48 89 10          mov   %rdx,(%rax)
886: 48 8d 45 b0       lea   -0x50(%rbp),%rax
88a: 48 89 c7          mov   %rax,%rdi
88d: e8 f8 fe ff ff    callq 78a <CopyString>
892: b8 00 00 00 00    mov   $0x0,%eax
897: 48 8b 4d f8       mov   -0x8(%rbp),%rcx
89b: 64 48 33 0c 25 28 00 xor   %fs:0x28,%rcx
8a2: 00 00
8a4: 74 05            je     8ab <main+0xca>
8a6: e8 95 fd ff ff    callq 640 <__stack_chk_fail@plt>
8ab: c9              leaveq
8ac: c3              retq

```

000000000000078a <CopyString>:

```

78a: 55          push    %rbp
78b: 48 89 e5    mov     %rsp,%rbp
78e: 48 83 ec 30 sub     $0x30,%rsp
792: 48 89 7d d8 mov     %rdi,-0x28(%rbp)
796: 64 48 8b 04 25 28 00 mov     %fs:0x28,%rax
79d: 00 00
79f: 48 89 45 f8 mov     %rax,-0x8(%rbp)
7a3: 31 c0       xor     %eax,%eax
7a5: 48 8b 55 d8 mov     -0x28(%rbp),%rdx
7a9: 48 8d 45 e0 lea     -0x20(%rbp),%rax
7ad: 48 89 d6    mov     %rdx,%rsi
7b0: 48 89 c7    mov     %rax,%rdi
7b3: e8 68 fe ff ff callq   620 <strcpy@plt>
7b8: 90          nop
7b9: 48 8b 45 f8 mov     -0x8(%rbp),%rax
7bd: 64 48 33 04 25 28 00 xor     %fs:0x28,%rax
7c4: 00 00
7c6: 74 05       je      7cd <CopyString+0x43>
7c8: e8 73 fe ff ff callq   640 <__stack_chk_fail@plt>
7cd: c9          leaveq  %rax
7ce: c3          retq

```

00000000000007cf <hacked>:

```

7cf: 55          push    %rbp
7d0: 48 89 e5    mov     %rsp,%rbp
7d3: 48 8d 3d 2a 06 00 00 lea     0x62a(%rip),%rdi      # e04 <_IO_stdin_used+0x4>
7da: e8 51 fe ff ff callq   630 <puts@plt>
7df: eb f2       jmp     7d3 <hacked+0x4>

```



```

000000000000008fa <sumn>:
8fa: 55                                push    %rbp
8fb: 48 89 e5                          mov     %rsp,%rbp
8fe: 48 81 ec d0 0f 00 00             sub     $0xfd0,%rsp
905: 89 bd 3c f0 ff ff               mov     %edi,-0xfc4(%rbp)
90b: 64 48 8b 04 25 28 00           mov     %fs:0x28,%rax
912: 00 00
914: 48 89 45 f8                      mov     %rax,-0x8(%rbp)
918: 31 c0                            xor     %eax,%eax
91a: c7 85 4c f0 ff ff 00          movl    $0x0,-0xfb4(%rbp)
921: 00 00 00
924: c7 85 48 f0 ff ff 00          movl    $0x0,-0xfb8(%rbp)
92b: 00 00 00
92e: eb 1c                            jmp     94c <sumn+0x52>
930: 8b 85 48 f0 ff ff             mov     -0xfb8(%rbp),%eax
936: 48 98                          cltq
938: 8b 95 48 f0 ff ff             mov     -0xfb8(%rbp),%edx
93e: 89 94 85 50 f0 ff ff          mov     %edx,-0xfb0(%rbp,%rax,4)
945: 83 85 48 f0 ff ff 01          addl    $0x1,-0xfb8(%rbp)
94c: 8b 85 48 f0 ff ff             mov     -0xfb8(%rbp),%eax
952: 3b 85 3c f0 ff ff             cmp     -0xfc4(%rbp),%eax
958: 7c d6                          jl      930 <sumn+0x36>
95a: c7 85 48 f0 ff ff 00          movl    $0x0,-0xfb8(%rbp)
961: 00 00 00
964: eb 1c                            jmp     982 <sumn+0x88>
966: 8b 85 48 f0 ff ff             mov     -0xfb8(%rbp),%eax
96c: 48 98                          cltq
96e: 8b 84 85 50 f0 ff ff          mov     -0xfb0(%rbp,%rax,4),%eax
975: 01 85 4c f0 ff ff             add     %eax,-0xfb4(%rbp)
97b: 83 85 48 f0 ff ff 01          addl    $0x1,-0xfb8(%rbp)
982: 8b 85 48 f0 ff ff             mov     -0xfb8(%rbp),%eax
988: 3b 85 3c f0 ff ff             cmp     -0xfc4(%rbp),%eax
98e: 7c d6                          jl      966 <sumn+0x6c>
990: 8b 85 4c f0 ff ff             mov     -0xfb4(%rbp),%eax
996: 48 8b 4d f8                      mov     -0x8(%rbp),%rcx
99a: 64 48 33 0c 25 28 00          xor     %fs:0x28,%rcx
9a1: 00 00
9a3: 74 05                          je      9aa <sumn+0xb0>
9a5: e8 96 fc ff ff               callq   640 <__stack_chk_fail@plt>
9aa: c9                            leaveq
9ab: c3                            retq

```

00000000000009ac <f>:

9ac:	55	push	%rbp
9ad:	48 89 e5	mov	%rsp,%rbp
9b0:	48 83 ec 10	sub	\$0x10,%rsp
9b4:	89 7d fc	mov	%edi,-0x4(%rbp)
9b7:	83 7d fc 00	cmpl	\$0x0,-0x4(%rbp)
9bb:	7e 16	jle	9d3 <f+0x27>
9bd:	8b 45 fc	mov	-0x4(%rbp),%eax
9c0:	83 e8 01	sub	\$0x1,%eax
9c3:	89 c7	mov	%eax,%edi
9c5:	e8 e2 ff ff ff	callq	9ac <f>
9ca:	89 c2	mov	%eax,%edx
9cc:	8b 45 fc	mov	-0x4(%rbp),%eax
9cf:	01 d0	add	%edx,%eax
9d1:	eb 05	jmp	9d8 <f+0x2c>
9d3:	b8 00 00 00 00	mov	\$0x0,%eax
9d8:	c9	leaveq	
9d9:	c3	retq	

00000000000009da <sub>:

9da:	55	push	%rbp
9db:	48 89 e5	mov	%rsp,%rbp
9de:	89 7d fc	mov	%edi,-0x4(%rbp)
9e1:	89 75 f8	mov	%esi,-0x8(%rbp)
9e4:	89 55 f4	mov	%edx,-0xc(%rbp)
9e7:	48 89 4d e8	mov	%rcx,-0x18(%rbp)
9eb:	4c 89 45 e0	mov	%r8,-0x20(%rbp)
9ef:	4c 89 4d d8	mov	%r9,-0x28(%rbp)
9f3:	8b 45 10	mov	0x10(%rbp),%eax
9f6:	66 89 45 f0	mov	%ax,-0x10(%rbp)
9fa:	b8 01 00 00 00	mov	\$0x1,%eax
9ff:	2b 45 fc	sub	-0x4(%rbp),%eax
a02:	2b 45 f8	sub	-0x8(%rbp),%eax
a05:	2b 45 f4	sub	-0xc(%rbp),%eax
a08:	89 c2	mov	%eax,%edx
a0a:	48 8b 45 e8	mov	-0x18(%rbp),%rax
a0e:	29 c2	sub	%eax,%edx
a10:	48 8b 45 e0	mov	-0x20(%rbp),%rax
a14:	29 c2	sub	%eax,%edx
a16:	48 8b 45 d8	mov	-0x28(%rbp),%rax
a1a:	29 c2	sub	%eax,%edx
a1c:	0f bf 45 f0	movswl	-0x10(%rbp),%eax
a20:	29 c2	sub	%eax,%edx
a22:	89 d0	mov	%edx,%eax
a24:	5d	pop	%rbp
a25:	c3	retq	

000000000000ae4 <swap>:

ae4:	55	push	%rbp
ae5:	48 89 e5	mov	%rsp,%rbp
ae8:	48 89 7d e8	mov	%rdi,-0x18(%rbp)
aec:	48 89 75 e0	mov	%rsi,-0x20(%rbp)
af0:	48 8b 45 e8	mov	-0x18(%rbp),%rax
af4:	8b 00	mov	(%rax),%eax
af6:	89 45 fc	mov	%eax,-0x4(%rbp)
af9:	48 8b 45 e0	mov	-0x20(%rbp),%rax
afd:	8b 10	mov	(%rax),%edx
aff:	48 8b 45 e8	mov	-0x18(%rbp),%rax
b03:	89 10	mov	%edx,(%rax)
b05:	48 8b 45 e0	mov	-0x20(%rbp),%rax
b09:	8b 55 fc	mov	-0x4(%rbp),%edx
b0c:	89 10	mov	%edx,(%rax)
b0e:	90	nop	
b0f:	5d	pop	%rbp
b10:	c3	retq	

000000000000b11 <main1>:

b11:	55	push	%rbp
b12:	48 89 e5	mov	%rsp,%rbp
b15:	48 83 ec 30	sub	\$0x30,%rsp
b19:	64 48 8b 04 25 28 00	mov	%fs:0x28,%rax
b20:	00 00		
b22:	48 89 45 f8	mov	%rax,-0x8(%rbp)
b26:	31 c0	xor	%eax,%eax
b28:	c7 45 d8 64 00 00 00	movl	\$0x64,-0x28(%rbp)
b2f:	c7 45 dc c8 00 00 00	movl	\$0xc8,-0x24(%rbp)
b36:	c7 45 e4 00 04 00 00	movl	\$0x400,-0x1c(%rbp)
b3d:	6a 07	pushq	\$0x7
b3f:	41 b9 06 00 00 00	mov	\$0x6,%r9d
b45:	41 b8 05 00 00 00	mov	\$0x5,%r8d
b4b:	b9 04 00 00 00	mov	\$0x4,%ecx
b50:	ba 03 00 00 00	mov	\$0x3,%edx
b55:	be 02 00 00 00	mov	\$0x2,%esi
b5a:	bf 01 00 00 00	mov	\$0x1,%edi
b5f:	e8 37 ff ff ff	callq	a9b <add>
b64:	48 83 c4 08	add	\$0x8,%rsp
b68:	89 45 e8	mov	%eax,-0x18(%rbp)
b6b:	6a 11	pushq	\$0x11
b6d:	41 b9 10 00 00 00	mov	\$0x10,%r9d
b73:	41 b8 0f 00 00 00	mov	\$0xf,%r8d
b79:	b9 0e 00 00 00	mov	\$0xe,%ecx
b7e:	ba 0d 00 00 00	mov	\$0xd,%edx
b83:	be 0c 00 00 00	mov	\$0xc,%esi
b88:	bf 0b 00 00 00	mov	\$0xb,%edi
b8d:	e8 94 fe ff ff	callq	a26 <add1>
b92:	48 83 c4 08	add	\$0x8,%rsp
b96:	89 45 ec	mov	%eax,-0x14(%rbp)

第 2 章 实验环境建立

2.1 Ubuntu 下 CodeBlocks 反汇编（10 分）

CodeBlocks 运行 hellolinux.c。反汇编查看 printf 函数的实现。

要求：C、ASM、内存(显示 hello 等内容)、堆栈（call printf 前）、寄存器同时在一个窗口。

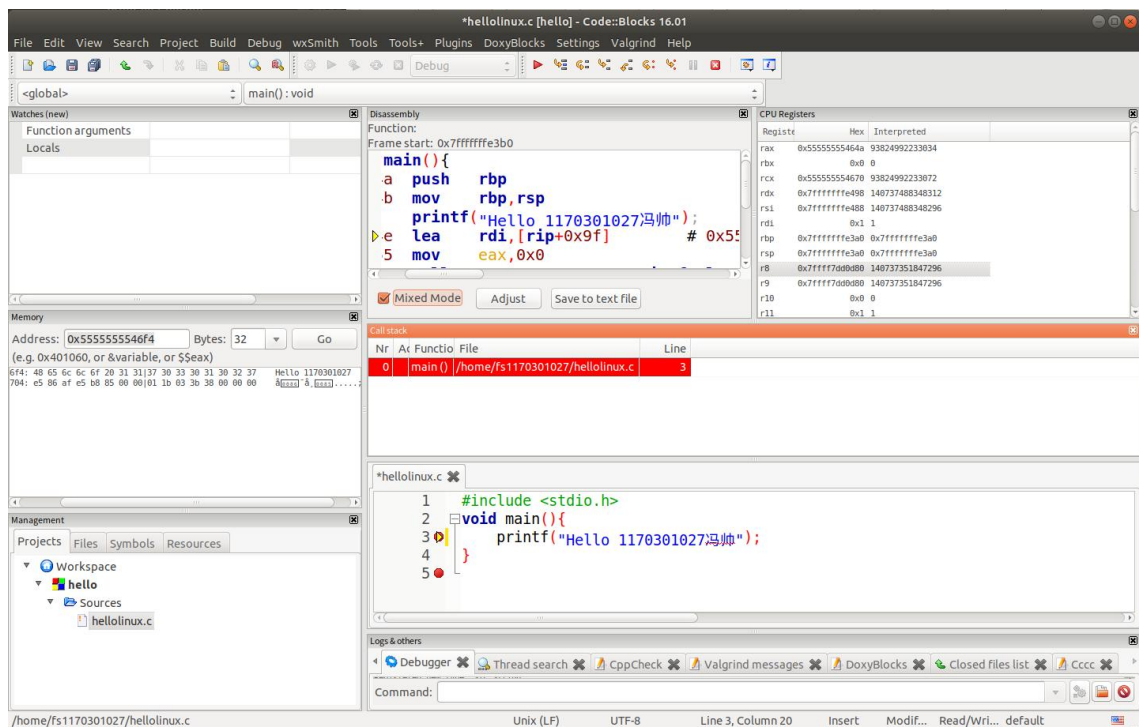


图 2-1 Ubuntu 下 CodeBlocks 反汇编截图

2.2 Ubuntu 下 EDB 运行环境建立（10 分）

用 EDB 调试 hellolinux.c 的执行文件，截图，要求同 2.1

计算机系统实验报告

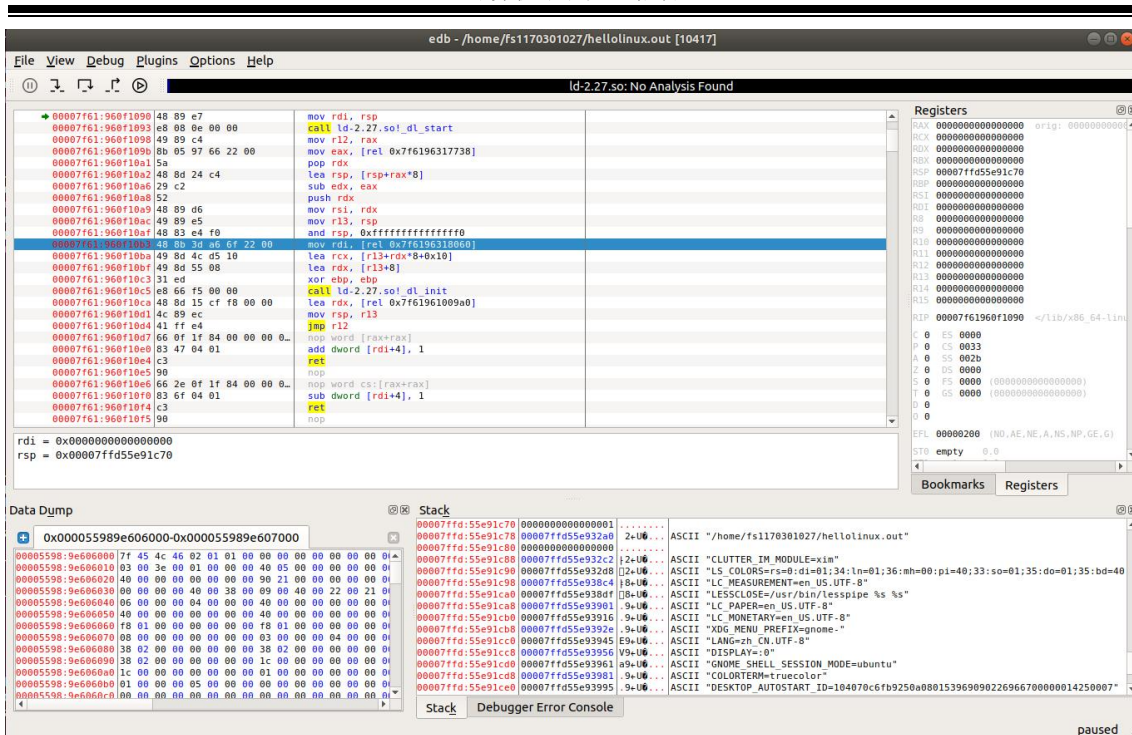


图 2-2 Ubuntu 下 EDB 截图

第 3 章 各阶段炸弹破解与分析

每阶段 15 分，密码 10 分，分析 5 分，总分不超过 80 分

3.1 阶段 1 的破解与分析

密码如下：Wow! Brazil is big.

破解过程：

```
106d: e8 d2 06 00 00    callq 1744 <read_line>
1072: 48 89 c7          mov  %rax,%rdi
1075: e8 fa 00 00 00    callq 1174 <phase_1>
107a: e8 09 08 00 00    callq 1888 <phase_defused>
107f: 48 8d 3d 92 15 00 00    lea  0x1592(%rip),%rdi    # 2618 <_IO_stdin_used+0xf8>
1086: 48 8b 15 92 15 00 00    mov  0x1592(%rip),%rdi    # 2618 <_IO_stdin_used+0xf8>

0000000000001174 <phase_1>:
1174: 48 83 ec 08      sub  $0x8,%rsp
1178: 48 8d 35 ed 14 00 00    lea  0x14ed(%rip),%rsi    # 266c <_IO_stdin_used+0x14c> //rsi全局变量地址
117f: e8 4d 04 00 00    callq 15d1 <strings_not_equal>
1184: 85 c0           test  %eax,%eax
1186: 75 05          jne  118d <phase_1+0x19>
1188: 48 83 c4 08      add  $0x8,%rsp
118c: c3            retq
118d: e8 4b 05 00 00    callq 16dd <explode_bomb>
1192: eb f4          jmp  1188 <phase_1+0x14>
```

第一阶段破解一个字符串

分析了 bomb.c 程序构成之后，到反汇编中查找到 main 函数，对应先找到 phase1 的地址，对应可以看到在 117f 调用 strings_not_equal 函数之前，将寄存器 rsi（第二个参数）的地址做了偏移，也就是指向了 0x14ed（%rip），因而判断，这个地址指向了需要破解的字符串，因而进入到 gdb 中实际操作了一遍并定位 1178 行偏移后地址得到破解结果

```

(gdb) b 73
Breakpoint 1 at 0x106d: file bomb.c, line 73.
(gdb) r
Starting program: /home/fs1170301027/bomb129/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!

Breakpoint 1, main (argc=<optimized out>, argv=<optimized out>) at bomb.c:73
73      input = read_line();          /* Get input          */
(gdb) n
74      phase_1(input);               /* Run the phase      */
(gdb) si
0x000055555555075      74      phase_1(input);               /* Run the phase      */
(gdb) si
0x000055555555174 in phase_1 ()
(gdb) x/10i $rip
=> 0x55555555174 <phase_1>:  sub    rsp,0x8
0x55555555178 <phase_1+4>:  lea     rsi,[rip+0x14ed]    # 0x55555555666c
0x5555555517f <phase_1+11>: call   0x555555555d1 <strings_not_equal>
0x55555555184 <phase_1+16>: test    eax,eax
0x55555555186 <phase_1+18>: jne     0x5555555518d <phase_1+25>
0x55555555188 <phase_1+20>: add     rsp,0x8
0x5555555518c <phase_1+24>: ret
0x5555555518d <phase_1+25>: call   0x555555556dd <explode_bomb>
0x55555555192 <phase_1+30>: jmp     0x55555555188 <phase_1+20>
0x55555555194 <phase_2>:  push    rbp
(gdb) x/s 0x55555555666c
0x55555555666c: "Wow! Brazil is big."

```

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：

```

asm - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
00000000000001194 <phase_2>:
1194: 55      push    %rbp
1195: 53      push    %rbx
1196: 48 83 ec 28  sub    $0x28,%rsp          //rsp-40
119a: 48 89 e6      mov     %rsp,%rsi          //rsp给rsi
119d: e8 61 05 00 00  callq   1703 <read_six_numbers> //调用读数
11a2: 83 3c 24 00    cmpl    $0x0,(%rsp)        //比较0和当前rsp
11a6: 75 07      jne     11af <phase_2+0x1b> //不等于时候炸
11a8: 83 7c 24 04 01  cmpl    $0x1,0x4(%rsp)     //比较1和rsp+4
11ad: 74 05      je      11b4 <phase_2+0x20> //等于时候跳转至11b4
11af: e8 29 05 00 00  callq   16dd <explode_bomb> //不等于时候炸
11b4: 48 89 e3      mov     %rsp,%rbx          //当前rsp给rbx
11b7: 48 8d 6b 10    lea     0x10(%rbx),%rbp     //rbx(rsp)+16给rbp
11bb: eb 09      jmp     11c6 <phase_2+0x32> //跳转11c6
11bd: 48 83 c3 04    add     $0x4,%rbx          //rbx = rbx+4
11c1: 48 39 eb      cmp     %rbp,%rbx          //比较rbp和rbx
11c4: 74 11      je      11d7 <phase_2+0x43> //等于时候跳转11d7
11c6: 8b 43 04      mov     0x4(%rbx),%eax      //rbx+4 给eax
11c9: 03 03      add     (%rbx),%eax         //rbx内容加给eax
11cb: 39 43 08      cmp     %eax,0x8(%rbx)      //比较eax和rbx+8
11ce: 74 ed      je      11bd <phase_2+0x29> //等于时候跳转11bd
11d0: e8 08 05 00 00  callq   16dd <explode_bomb> //不等于时候炸
11d5: eb e6      jmp     11bd <phase_2+0x29>
11d7: 48 83 c4 28    add     $0x28,%rsp          //rsp+40
11db: 5b      pop     %rbx

```

刚开始看的时候没什么感觉，只感觉冗长，读不懂，后来查完了汇编指令之

后就一个一个把注释都写上了，可以看到在 phase2 处理过程中有一个 read_six_numbers 的调用过程，所以猜想，可能 sercet2 是要输入六个数，当老师实验课上演示的时候细看了一下没想到真的是六个整数。

后来分析，可以看到在 1196 行将寄存器 rsp 的地址减了四十，想到应该是留出来位置存放数据，那么在 11a2 行比较立即数 0 和 rsp 就相当于在比较录得第一个数据了，再往后又比较立即数 1 和 rsp+4 【一个 int 占 4 个】，正好代表比较 1 和第二个录的数据，由此确定了前两个数是 0,1

可以看到在 11ad 行上边是判断，此句代表等于时跳转到 11b4（跳转 1），跟随之后发现 11b4 行代表把 rsp 给了 rbx，而后把 rbx+16 给了 rbp，先看 11c1 中的比较可知此为循环结束的判断标志，因为 rbp=rbx+16 指向的是录入的最后一个数也就是第六个数；而中间的跳转至 11c6 代表进入循环，跟踪红色部分发现这是一个算法：

eax = rbx+4（后一个数）

eax = eax + rbx（后一个数加前一个数）

eax ? = rbx + 8（后两个数）

等于的时候继续循环（跳转 2，使得 rbx 指向下一个数直到最后一个数使得条件成立退出循环进入跳转 3），不等于时候退出循环并爆炸

因而由前两个数 0, 1 可以推出

前一个数 当前数 后一个数

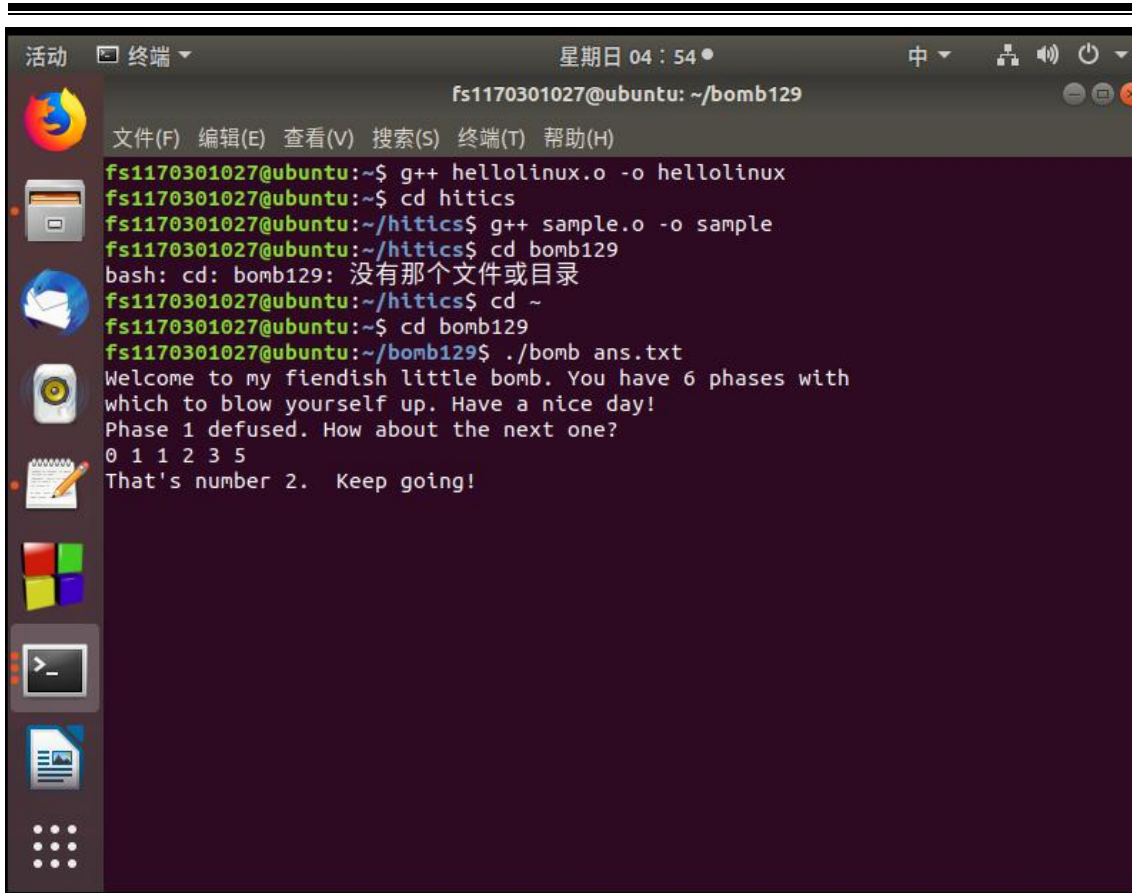
0 + 1 = 1

1 + 1 = 2

1 + 2 = 3

2 + 3 = 5

得出结果 0 1 1 2 3 5



The screenshot shows a terminal window titled "fs1170301027@ubuntu: ~/bomb129". The user has executed the following commands and received the following output:

```
fs1170301027@ubuntu:~$ g++ hellolinux.o -o hellolinux
fs1170301027@ubuntu:~$ cd hitics
fs1170301027@ubuntu:~/hitics$ g++ sample.o -o sample
fs1170301027@ubuntu:~/hitics$ cd bomb129
bash: cd: bomb129: 没有那个文件或目录
fs1170301027@ubuntu:~/hitics$ cd ~
fs1170301027@ubuntu:~$ cd bomb129
fs1170301027@ubuntu:~/bomb129$ ./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?
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：4 0（5 -473 或 3 -473 或 2 221 或 1 -219）

破解过程：

```

asm - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
11ec: 48 8d 35 24 16 00 00    lea     0x1624(%rip),%rsi    # 2817 <array.3415+0x177>
                                   //("%d %d")
11f3: b8 00 00 00 00 00    mov     $0x0,%eax          //eax = 0
11f8: e8 63 fc ff ff    callq   e60 <_isoc99_sscanf@plt>
11fd: 83 f8 01    cmp     $0x1,%eax          //比较eax和1 (条件)
1200: 7e 1f    jle     1221 <phase_3+0x43> //eax<=1时候跳转1221炸弹爆炸
1202: 83 7c 24 0c 07    cmpl    $0x7,0xc(%rsp)     //比较rsp+12 (第二个数) 和 7
1207: 0f 87 8b 00 00 00    ja      1298 <phase_3+0xba> //rsp+12>7时候跳转1298炸弹爆炸
120d: 8b 44 24 0c    mov     0xc(%rsp),%eax     //否则eax = rsp+12 (第二个数)
1211: 48 8d 15 68 14 00 00    lea     0x1468(%rip),%rdx  # 2680 <_IO_stdin_used+0x160>
                                   //rdx = rip+0x1468
1218: 48 63 04 82    movslq  (%rdx,%rax,4),%rax  //rax = rdx+rax*4
121c: 48 01 d0    add     %rdx,%rax          //rax = rax + rdx
121f: ff e0    jmpq    *%rax              //跳转到rax里面的地址处
1221: e8 b7 04 00 00    callq   16dd <explode_bomb>
1226: eb da    jmp     1202 <phase_3+0x24> //跳转1202 (上)
1228: b8 36 01 00 00    mov     $0x136,%eax        //0x136给eax(310)
122d: eb 05    jmp     1234 <phase_3+0x56> //跳转1234 (下)
122f: b8 00 00 00 00 00    mov     $0x0,%eax          //eax = 0
1234: 2d b8 01 00 00    sub     $0x1b8,%eax         //eax-0x1b8(440)
1239: 05 b6 02 00 00    add     $0x2b6,%eax         //eax+0x2b6(694)
123e: 2d d9 01 00 00    sub     $0x1d9,%eax         //eax-0x1d9(473)
1243: 05 d9 01 00 00    add     $0x1d9,%eax         //eax+0x1d9
1248: 2d d9 01 00 00    sub     $0x1d9,%eax         //eax-0x1d9
124d: 05 d9 01 00 00    add     $0x1d9,%eax         //eax+0x1d9
1252: 2d d9 01 00 00    sub     $0x1d9,%eax         //eax-0x1d9
1257: 83 7c 24 0c 05    cmpl    $0x5,0xc(%rsp)     //比较rsp+12 (第一个数) 和 5
125c: 7f 06    jg      1264 <phase_3+0x86> //大于时跳转1264炸弹爆炸
125e: 39 44 24 08    cmp     %eax,0x8(%rsp)     //否则比较rsp+8 (第二个数) 和eax
1262: 74 05    je      1269 <phase_3+0x8b> //相等时跳转1269
1264: e8 74 04 00 00    callq   16dd <explode_bomb>

```

因为之前怎么都没想法就把注释都打上了，，然而没什么用，接下来是在 edb 中调试过程

```

RAX 0000000000000000
RCX 0000000000000000
RDX 00007ffd03629318
RBX 0000000000000000
RSP 00007ffd03629310
RBP 00005600c3acb4a0 ASCII "AWAVI"
RSI 0000000000000000
RDI 00007ffd03628ca0
R8 0000000000000000

00007ffd:03629310 00005600c3acb4a0  ASCII "AWAVI"
00007ffd:03629318 0000000040000000  .....
00007ffd:03629320 00007ffd03629410  ..b.0...

```

这两个截图是我输入之后，汇编程序中调用 scanf 函数之后的结果，我的数据是 4、0，可见在十六进制表示时 rsp(310)+8 存储第二个数据 0，【这个点我想了半天，之前相反了，后来反应过来栈顶减 18 分配的空间，应该倒过来放】，rsp+12 存储第一个数据

而且后来看到，cmp dword[rsp = 0xc], 7(a202) 行的时候，意识到是在比较第一个数，只有比 7 小的时候炸弹才不会被引爆

计算机系统实验报告

00005600:c3aca1de	48 83 ec 18	sub rsp, 0x18	
00005600:c3aca1e2	48 8d 4c 24 08	lea rcx, [rsp+8]	
00005600:c3aca1e7	48 8d 54 24 0c	lea rdx, [rsp+0xc]	
00005600:c3aca1ec	48 8d 35 24 16 00 00	lea rsi, [rel 0x5600c3acb817]	ASCII "%d %d"
00005600:c3aca1f3	b8 00 00 00 00	mov eax, 0	
00005600:c3aca1f8	e8 63 fc ff ff	call bomb!_isoc99_sscanf@plt	
00005600:c3aca1fd	83 7c 01	cmp eax, 1	
00005600:c3aca200	7e 1f	jle 0x5600c3aca221	
00005600:c3aca202	83 7c 24 0c 07	cmp dword [rsp+0xc], 7	
00005600:c3aca207	0f 87 8b 00 00 00	ja 0x5600c3aca298	
00005600:c3aca20d	8b 44 24 0c	mov eax, [rsp+0xc]	
00005600:c3aca211	48 8d 15 68 14 00 00	lea rdx, [rel 0x5600c3acb680]	
00005600:c3aca218	48 63 04 82	movsxd rax, [rdx+rax*4]	
00005600:c3aca21c	48 01 d0	add rax, rdx	
00005600:c3aca21f	ff e0	jmp rax	
00005600:c3aca221	e8 b7 04 00 00	call bomb!explode_bomb	
00005600:c3aca226	eb da	jmp 0x5600c3aca202	
00005600:c3aca228	b8 36 01 00 00	mov eax, 0x136	
00005600:c3aca22d	eb 05	jmp 0x5600c3aca234	
00005600:c3aca22f	b8 00 00 00 00	mov eax, 0	
00005600:c3aca234	2d b8 01 00 00	sub eax, 0x1b8	
00005600:c3aca239	05 b6 02 00 00	add eax, 0x2b6	
00005600:c3aca23e	2d d9 01 00 00	sub eax, 0x1d9	
00005600:c3aca243	05 d9 01 00 00	add eax, 0x1d9	
00005600:c3aca248	2d d9 01 00 00	sub eax, 0x1d9	
00005600:c3aca24d	05 d9 01 00 00	add eax, 0x1d9	
00005600:c3aca252	2d d9 01 00 00	sub eax, 0x1d9	
00005600:c3aca257	83 7c 24 0c 05	cmp dword [rsp+0xc], 5	
00005600:c3aca25c	7f 06	jg 0x5600c3aca264	
00005600:c3aca25e	39 44 24 08	cmp [rsp+8], eax	
00005600:c3aca262	74 05	je 0x5600c3aca269	
00005600:c3aca264	e8 74 04 00 00	call bomb!explode_bomb	
00005600:c3aca269	48 83 c4 18	add rsp, 0x18	
00005600:c3aca26d	c3	ret	
00005600:c3aca26e	b8 00 00 00 00	mov eax, 0	
00005600:c3aca273	eb c4	jmp 0x5600c3aca239	
00005600:c3aca275	b8 00 00 00 00	mov eax, 0	
00005600:c3aca27a	eb c2	jmp 0x5600c3aca23e	
00005600:c3aca27c	b8 00 00 00 00	mov eax, 0	
00005600:c3aca281	eb c0	jmp 0x5600c3aca243	
00005600:c3aca283	b8 00 00 00 00	mov eax, 0	

继续运行，运行至此行时候，可以看到接下来要跳转到 rax 所在行

00005600:c3aca221	b8 00 00 00 00	mov eax, 0	
00005600:c3aca234	2d b8 01 00 00	sub eax, 0x1b8	
00005600:c3aca239	05 b6 02 00 00	add eax, 0x2b6	
00005600:c3aca23e	2d d9 01 00 00	sub eax, 0x1d9	
00005600:c3aca243	05 d9 01 00 00	add eax, 0x1d9	
00005600:c3aca248	2d d9 01 00 00	sub eax, 0x1d9	
00005600:c3aca24d	05 d9 01 00 00	add eax, 0x1d9	
00005600:c3aca252	2d d9 01 00 00	sub eax, 0x1d9	
00005600:c3aca257	83 7c 24 0c 05	cmp dword [rsp+0xc], 5	
00005600:c3aca25c	7f 06	jg 0x5600c3aca264	
00005600:c3aca25e	39 44 24 08	cmp [rsp+8], eax	
00005600:c3aca262	74 05	je 0x5600c3aca269	
00005600:c3aca264	e8 74 04 00 00	call bomb!explode_bomb	
00005600:c3aca269	48 83 c4 18	add rsp, 0x18	
00005600:c3aca26d	c3	ret	
00005600:c3aca26e	b8 00 00 00 00	mov eax, 0	
00005600:c3aca273	eb c4	jmp 0x5600c3aca239	
00005600:c3aca275	b8 00 00 00 00	mov eax, 0	
00005600:c3aca27a	eb c2	jmp 0x5600c3aca23e	

跳转过上一步之后，返回到，当前所在行，之后就是加减 0x1d9，直至 a257 行，相当于未增未减，eax 还是在跳转 rax 行时候赋给的 0x0

00005600:c3aca257	83 7c 24 0c 05	cmp dword [rsp+0xc], 5	
00005600:c3aca25c	7f 06	jg 0x5600c3aca264	
00005600:c3aca25e	39 44 24 08	cmp [rsp+8], eax	
00005600:c3aca262	74 05	je 0x5600c3aca269	
00005600:c3aca264	e8 74 04 00 00	call bomb!explode_bomb	
00005600:c3aca269	48 83 c4 18	add rsp, 0x18	
00005600:c3aca26d	c3	ret	

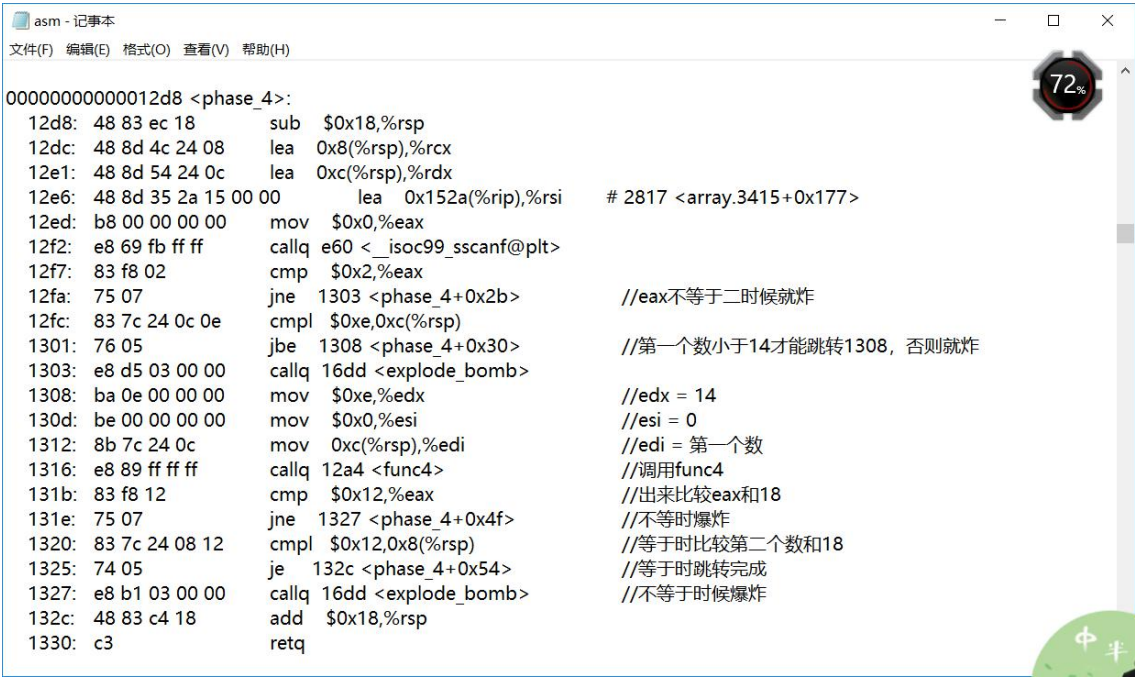
最后这几行其实是在判断第一个数小于五不，第二个数等于 eax 不，由此看出 4 和 0 正好通过了，然而其实分析跳转 rax 行其实和当时 rax 中存储的地址有

关，而当时的 `rdx` 是由第二个数决定的，所以我判断答案可能不唯一，所以如果我能把炸弹全破解我想回来把第一个数为 0~3 的情况试出来。如果不对也算是一个验证。

3.4 阶段 4 的破解与分析

密码如下：11 18

破解过程：



```

asm - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

00000000000012d8 <phase_4>:
12d8: 48 83 ec 18      sub    $0x18,%rsp
12dc: 48 8d 4c 24 08    lea    0x8(%rsp),%rcx
12e1: 48 8d 54 24 0c    lea    0xc(%rsp),%rdx
12e6: 48 8d 35 2a 15 00 00    lea    0x152a(%rip),%rsi    # 2817 <array.3415+0x177>
12ed: b8 00 00 00 00    mov    $0x0,%eax
12f2: e8 69 fb ff ff    callq  e60 <_isoc99_sscanf@plt>
12f7: 83 f8 02         cmp    $0x2,%eax
12fa: 75 07           jne     1303 <phase_4+0x2b>    //eax不等于二时候就炸
12fc: 83 7c 24 0c 0e    cmpl   $0xe,0xc(%rsp)
1301: 76 05           jbe     1308 <phase_4+0x30>    //第一个数小于14才能跳转1308，否则就炸
1303: e8 d5 03 00 00    callq  16dd <explode_bomb>
1308: ba 0e 00 00 00    mov    $0xe,%edx            //edx = 14
130d: be 00 00 00 00    mov    $0x0,%esi            //esi = 0
1312: 8b 7c 24 0c      mov    0xc(%rsp),%edi        //edi = 第一个数
1316: e8 89 ff ff ff    callq  12a4 <func4>           //调用func4
131b: 83 f8 12         cmp    $0x12,%eax            //出来比较eax和18
131e: 75 07           jne     1327 <phase_4+0x4f>    //不等时爆炸
1320: 83 7c 24 08 12    cmpl   $0x12,0x8(%rsp)       //等于时比较第二个数和18
1325: 74 05           je      132c <phase_4+0x54>    //等于时跳转完成
1327: e8 b1 03 00 00    callq  16dd <explode_bomb>    //不等于时候爆炸
132c: 48 83 c4 18      add    $0x18,%rsp
1330: c3             retq
  
```

首先分析阶段四，写完了注释之后我发现，1301 行要求第一个数要小于 14 才能进入程序中，并且在该阶段程序最后 1320 行处比较了第二个数和 18，要求两者相等，所以我初步判断第一个数不超过 14，第二个数等于 18。

```

asm - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
00000000000012a4 <func4>:
12a4: 53          push  %rbx
12a5: 89 d0       mov   %edx,%eax          //edx给eax
12a7: 29 f0       sub   %esi,%eax          //eax = eax-esi
12a9: 89 c3       mov   %eax,%ebx          //ebx = eax
12ab: c1 eb 1f    shr   $0x1f,%ebx         //ebx左移0x1f位
12ae: 01 c3       add   %eax,%ebx          //ebx = eax+ebx
12b0: d1 fb       sar   %ebx               //ebx算术右移1位
12b2: 01 f3       add   %esi,%ebx          //ebx = ebx + esi
12b4: 39 fb       cmp   %edi,%ebx          //比较ebx和edi
12b6: 7f 08       jg    12c0 <func4+0x1c>   //ebx>edi 跳转12c0
12b8: 39 fb       cmp   %edi,%ebx
12ba: 7c 10       jl    12cc <func4+0x28>   //ebx<edi 跳转12cc
12bc: 89 d8       mov   %ebx,%eax          //如若相等, 则将ebx给eax
12be: 5b         pop   %rbx
12bf: c3         retq                    //返回phase4去

12c0: 8d 53 ff    lea   -0x1(%rbx),%edx     //edx = rbx -1
12c3: e8 dc ff ff callq 12a4 <func4>         //重新调用
12c8: 01 c3       add   %eax,%ebx          //ebx = ebx + eax
12ca: eb f0       jmp   12bc <func4+0x18>
12cc: 8d 73 01    lea   0x1(%rbx),%esi      //esi = rbx + 1
12cf: e8 d0 ff ff callq 12a4 <func4>         //重新调用
12d4: 01 c3       add   %eax,%ebx

```

func (ebx)

```

{
    //数值ebx = rbx
    //数值edi = 第一个数
    //初始edx = 14,esi = 0, eax = 2
    eax = edx - esi;
    ebx = edx - esi;
    ebx = ebx >> 31; //()
    ebx + = edx;
    ebx = ebx >> 1;
    ebx + = esi;
    //上三句总结 ebx = edx/2 + esi
    if (ebx > edi)
        edx = ebx - 1;
        func (ebx) ;
        ebx = ebx + eax;
    elseif (ebx < edi)
        esi = ebx + 1;
        func (ebx) ;
        ebx = ebx + eax;
    else
        return ;
}

```

上述是我把 func4 分析了一下递归的过程，但是我确实看不懂，所以就按着那个大致写了一下 C 语言代码，因为我有第一个数小于 14 的结论，所以下面几幅图就是我自己用 codeblocks 对比汇编语言调试好多遍最终的出来的相对差不多的 C 语言代码实现，结果跟密码一样、、

```

5   int edx = 14;
6   int esi = 0;
7   int eax = 2;
8   int edi;
9   int func(int ebx)
10  {
11      //数值ebx = rbx
12      //数值edi = 第一个数
13      //初始edx = 14, esi = 0, eax = 2
14      eax = edx;
15      eax = eax - esi;
16      ebx = eax;
17      ebx = ebx >> 31; //()
18      ebx += eax;
19      ebx = ebx >> 1;
20      ebx += esi;
21      //上三句总结 ebx = edx/2 + esi
22      if (ebx > edi) {
23          edx = ebx - 1;
24          eax = func(ebx);
25          ebx = eax + ebx;
26          return ebx;
27      }
28      if (ebx < edi) {
29          esi = ebx + 1;
30          eax = func(ebx);
31          ebx = eax + ebx;
32          return ebx;
33      }
34      return ebx;

```

```

int main()
{
    int i = 11;
    edi = i;
    while (i < 14) {
        edi = i;
        if (func(i) == 18)
            printf("%d", i);
        i++;
    }
    return 0;
}

```

```

11
Process returned -1073741571 (0xC00000FD)   execution time : 4.619 s
Press any key to continue.

```

3.5 阶段 5 的破解与分析

密码如下：abcdhn（答案不唯一）

破解过程：

```

asm - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

1330: c3                retq

00000000000001331 <phase_5>:
1331: 53                push %rbx
1332: 48 89 fb          mov %rdi,%rbx           //rbx = rdi (指向了录取的数据)
1335: e8 7a 02 00 00    callq 15b4 <string_length> //调用计算字符串长度存储rax中
133a: 83 f8 06          cmp $0x6,%eax
133d: 75 31             jne 1370 <phase_5+0x3f>   //字符串长度不等于6就炸
133f: 48 89 d8          mov %rbx,%rax           //rax = rbx (继续存储录入的数据)
1342: 48 8d 7b 06       lea 0x6(%rbx),%rdi       //地址rdi = rbx+6 (字符串末)
1346: b9 00 00 00 00    mov $0x0,%ecx           //ecx = 0 【初始化】
134b: 48 8d 35 4e 13 00 00 lea 0x134e(%rip),%rsi # 26a0 <array.3415>
                                     //地址rsi = rip + 0x134e
1352: 0f b6 10          movzbl (%rax),%edx       //edx = rax
1355: 83 e2 0f          and $0xf,%edx           //edx = edx & 1111 (决定edx<16,进一步决定下一步的偏
移量不超过60, )
1358: 03 0c 96          add (%rsi,%rdx,4),%ecx   //ecx = ecx + (rsi+4*edx)
135b: 48 83 c0 01       add $0x1,%rax           //rax++
135f: 48 39 f8          cmp %rdi,%rax           //比较rax和rdi (代表循环次数)
1362: 75 ee             jne 1352 <phase_5+0x21>   //不等于向上跳转至1352
1364: 83 f9 30          cmp $0x30,%ecx          //直至等于时比较ecx和0x30 (48)
1367: 74 05             je 136e <phase_5+0x3d>    //等于时候向下跳转结束, 不等于时候爆炸
1369: e8 6f 03 00 00    callq 16dd <explode_bomb>
136e: 5b                pop %rbx

```

刚编辑完注释时候可以看到第一个炸弹爆炸的点在字符串长度上，所以可以判断这时候后录取的数据是字符串，而且长度必须等于 6 才能往下进行。

分析汇编语言，可以从第 134b 行看到，rsi 指向了一个未知地址，跟踪查看发现，该寄存器变量存储了一系列数据，而分析到 1355 和 1358 行的时候，发现，这些数据是有用的，索引到的偏移量是不超过 60（ 4×15 ，由于 0xf 相与只能留下模十六的余数）的也就相当于 0x40，也就对应 a0 到 e0 之间的数，而偏移量是 4 的倍数也代表着这些数据正好对应了模 16 的 15 的余数，

000055be:1bd896a0	0000000a00000002
000055be:1bd896a8	0000000100000006
000055be:1bd896b0	000000010000000c
000055be:1bd896b8	0000000300000009
000055be:1bd896c0	0000000700000004
000055be:1bd896c8	000000050000000e
000055be:1bd896d0	000000080000000b
000055be:1bd896d8	0000000d0000000f

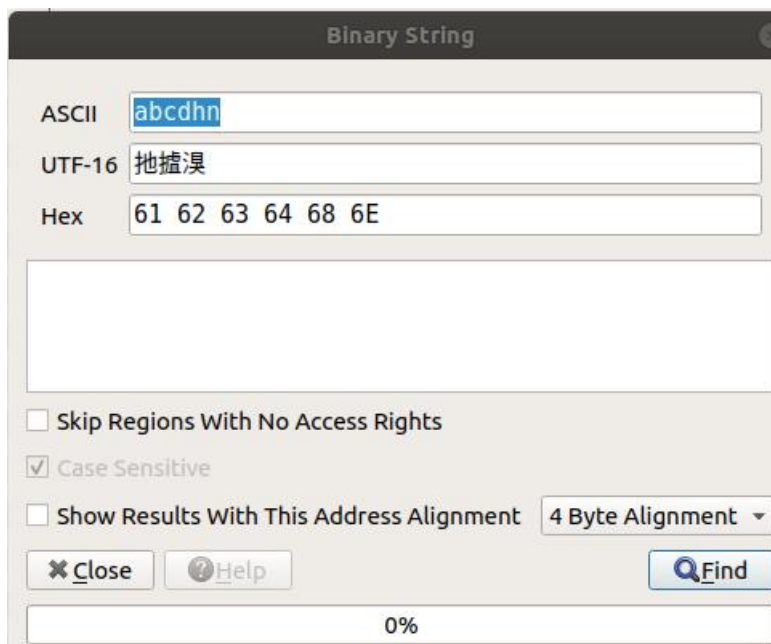
记录下来分别是

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

0x

2 a 6 1 c 10 9 3 4 7 5 e 8 b f d

而继续分析程序看到，在 135f 和 1362 行意思是用地址自增后的 `rax` 和 `rdi`（字符串末地址）比较，因而分析出，要将六个字符全部加一遍才算完，后来看到 1364 行要将 `ecx` 和 `0x30` 比较，所以六个数据分成三个十六来加是很自然的，因而也能判断出密码并不只有唯一一个，我选择 a 和 6,1 和 f, c 和 4，对应的余数是 1,2,3,4,8,e,如下在 edb 中直接表示出来了。。然后通过了




```

fs1170301027@ubuntu:~/bomb129$ ./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.

abcdhn
Good work! On to the next...

```

3.6 阶段 6 的破解与分析

密码如下：2 4 1 5 6 3

破解过程：

好吧还是和上边一样，我把九十九行代码都注释上了，然而完全看不懂，然后我在 edb 中调试时候才能看得懂点一块一块的，如下一幅图是判断输入的数据是否大于零小于等于六而且要互不相等，否则就炸，还有一幅图是判断之后所有的数据对 7 取补，也就是用 7 去一个一个减这些数据，然后代替原数据。

```

13b6: e8 22 03 00 00    callq 16dd <explode_bomb>
13bb: eb e5            jmp 13a2 <phase_6+0x2b>    //跳转回13a2
13bd: 49 83 c5 04      add $0x4,%r13              //r13 = r13+4
13c1: 4c 89 ed         mov %r13,%rbp              //rbp = r13
13c4: 41 8b 45 00      mov 0x0(%r13),%eax          //eax = 第一个数
13c8: 83 e8 01         sub $0x1,%eax
13cb: 83 f8 05         cmp $0x5,%eax
13ce: 77 cb           ja 139b <phase_6+0x24>      //第一个数减一大于五就炸,
13d0: 41 83 c6 01      add $0x1,%r14d
13d4: 41 83 fe 06      cmp $0x6,%r14d              //r14d初始化为零，自增之后与6比较可知此为计数阶段
13d8: 74 05           je 13df <phase_6+0x68>      //相等就跳转13df
13da: 44 89 f3         mov %r14d,%ebx              //不等时候把r14d给ebx
13dd: eb cb           jmp 13aa <phase_6+0x33>      //跳回 13aa行
(判断所有数据小于等于6且大于等于0，否则炸)

```

```

13df: 49 8d 4c 24 18    lea 0x18(%r12),%rcx          //rcx = r12+0x18(24)
13e4: ba 07 00 00 00    mov $0x7,%edx               //edx = 7
13e9: 89 d0            mov %edx,%eax               //eax = edx
13eb: 41 2b 04 24       sub (%r12),%eax              //eax = eax-r12
13ef: 41 89 04 24       mov %eax,(%r12)              //r12 = eax
13f3: 49 83 c4 04      add $0x4,%r12               //r12 = r12+4
13f7: 4c 39 e1         cmp %r12,%rcx               //比较rcx和r12
13fa: 75 ed           jne 13e9 <phase_6+0x72>      //不等时候跳转13e9
(所有数据用7减)

```

运行到底部的时候发现有些套路。。

000055d3:b4789210	00000001000000af	...
000055d3:b4789218	000055d3b4789220	.X...U..
000055d3:b4789220	00000002000000d9	...
000055d3:b4789228	000055d3b4789230	0.X...U..
000055d3:b4789230	0000000300000376	V.....
000055d3:b4789238	000055d3b4789240	@.X...U..
000055d3:b4789240	0000000400000076	V.....
000055d3:b4789248	000055d3b4789250	P.X...U..
000055d3:b4789250	00000005000003a5	...
00007ffe:235058f0	000055767dc03220	2[]}vU..
00007ffe:235058f8	000055767dc03210	.2[]}vU..
00007ffe:23505900	000055767dc03110	.1[]}vU..
00007ffe:23505908	000055767dc03250	P2[]}vU..
00007ffe:23505910	000055767dc03240	@2[]}vU..
00007ffe:23505918	000055767dc03230	02[]}vU..

上边这两幅图是我跟踪栈顶指针附近发现的，我的测试数据是

5 6 1 2 3 4

用 7 处理之后

2 1 6 5 4 3

分别对应 `rsp`（58f0）以及接下来的 5 个指针，对应进去发现每一个指针所代表的地址都有一个数值与之对应

用三位尾数表示地址指针可表示为

2	1	6	5	4	3
220	210	110	250	240	230
d9	af	323	3a5	76	376

00005576:7da0042f	7f d2	jg 0x55767da00403
00005576:7da00431	eb db	jmp 0x55767da0040e
00005576:7da00433	48 8b 1c 24	mov rbx, [rsp]
00005576:7da00437	48 8b 44 24 08	mov rax, [rsp+8]
00005576:7da0043c	48 89 43 08	mov [rbx+8], rax
00005576:7da00440	48 8b 54 24 10	mov rdx, [rsp+0x10]
00005576:7da00445	48 89 50 08	mov [rax+8], rdx
00005576:7da00449	48 8b 44 24 18	mov rax, [rsp+0x18]
00005576:7da0044e	48 89 42 08	mov [rdx+8], rax
00005576:7da00452	48 8b 54 24 20	mov rdx, [rsp+0x20]
00005576:7da00457	48 89 50 08	mov [rax+8], rdx
00005576:7da0045b	48 8b 44 24 28	mov rax, [rsp+0x28]
00005576:7da00460	48 89 42 08	mov [rdx+8], rax
00005576:7da00464	48 c7 40 08 00 00 00 00	mov qword [rax+8], 0
00005576:7da0046c	bd 05 00 00 00	mov ebp, 5
00005576:7da00471	eb 09	jmp 0x55767da0047c
00005576:7da00473	48 8b 5b 08	mov rbx, [rbx+8]
00005576:7da00477	83 ed 01	sub ebp, 1
00005576:7da0047a	74 11	je 0x55767da0048d
00005576:7da0047c	48 8b 43 08	mov rax, [rbx+8]
00005576:7da00480	8b 00	mov eax, [rax]
00005576:7da00482	39 03	cmp [rbx], eax
00005576:7da00484	7d ed	jge 0x55767da00473
00005576:7da00486	e8 52 02 00 00	call bomb!explode_bomb
00005576:7da0048b	eb e6	jmp 0x55767da00473
00005576:7da0048d	48 83 c4 50	add rsp, 0x50
00005576:7da00491	5b	pop rbx
00005576:7da00492	5d	pop rbp
00005576:7da00493	41 5c	pop r12
00005576:7da00495	41 5d	pop r13
00005576:7da00497	41 5e	pop r14

dword ptr [rbx] = [0x000055767dc03220] = 0x000000d9
 eax = 0x000000af

然后如上这幅图试运行到该阶段底部的形式，可以看到鼠标指向的这条指令是在说比较两个指针指向的数据，必须大于才能接着比较，自然而然想到将指针指向的数据从大到小排序。而图中表示的正好是栈顶开始的依次比较，也就是从第一个数开始要求前一个数对应的数大于后一个数对应的数因而排序

d9	af	323	3a5	76	376（数值）
3a5	376	323	d9	af	76（排序）
5	3	6	2	1	4（对应 7-x）
2	4	1	5	6	3（对应 x）

即为密码

```
fs1170301027@ubuntu:~$ cd bomb129
fs1170301027@ubuntu:~/bomb129$ ./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...
2 4 1 5 6 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：1（DrEvil 触发，其余答案有 36,8,6）

破解过程：

00005561:95558808	83 3d fd 2d 20 00 06	cmp dword [rel.0x55619575b68c], 6	
00005561:9555880f	74 02	je 0x556195558893	
00005561:95558891	f3 c3	ret	
00005561:95558893	48 83 ec 68	sub rsp, 0x68	
00005561:95558897	48 8d 4c 24 00	lea rcx, [rsp+0]	
00005561:9555889c	48 8d 54 24 0c	lea rcx, [rsp+0xc]	
00005561:955588a1	4c 8d 44 24 10	lea r8, [rsp+0x10]	
00005561:955588a6	48 8d 35 b4 0f 00 00	lea rsi, [rel.0x556195559861]	ASCII "%d %d %s"
00005561:955588ad	48 8d 3d dc 2e 20 00	lea rdi, [rel.0x55619575b790]	ASCII "11 18 DrEvil"
00005561:955588b4	b8 00 00 00 00	mov eax, 0	
00005561:955588b9	e8 a2 f5 ff ff	call bomb!_isoc99_sscanf@plt	
00005561:955588be	83 f8 03	cmp eax, 3	
00005561:955588c1	74 11	je 0x5561955588d4	
00005561:955588c3	48 8d 3d d6 0e 00 00	lea rdi, [rel.0x5561955597a0]	ASCII "Congratulations! You've defused the bomb!"
00005561:955588ca	e8 e1 f4 ff ff	call bomb!puts@plt	
00005561:955588cf	48 83 c4 68	add rsp, 0x68	
00005561:955588d3	c3	ret	
00005561:955588d4	48 8d 7c 24 10	lea rdi, [rsp+0x10]	
00005561:955588d9	48 8d 35 8a 0f 00 00	lea rsi, [rel.0x55619555986a]	ASCII "DrEvil"
00005561:955588de	e8 ec fc ff ff	call bomb!strings_not_equal	
00005561:955588e5	b5 c0	test eax, eax	
00005561:955588e7	75 da	jne 0x5561955588c3	
00005561:955588e9	48 8d 3d 50 0e 00 00	lea rdi, [rel.0x556195559740]	ASCII "Curses, you've found the secret phase!"
00005561:955588ef	e8 bb f4 ff ff	call bomb!puts@plt	
00005561:955588f5	48 8d 3d 6c 0e 00 00	lea rdi, [rel.0x556195559768]	ASCII "But finding it and solving it are quite different..."
00005561:955588fc	e8 af f4 ff ff	call bomb!puts@plt	
00005561:95558901	b8 00 00 00 00	mov eax, 0	
00005561:95558906	e8 ce fb ff ff	call bomb!secret_phase	
00005561:9555890b	eb b6	jmp 0x5561955588c3	
00005561:9555890d	48 83 ec 08	sub rsp, 8	
00005561:95558911	b9 00 00 00 00	mov ecx, 0	
00005561:95558916	48 8d 15 a3 0f 00 00	lea rcx, [rel.0x5561955598c0]	ASCII "Program timed out after %d seconds\n"
00005561:9555891d	b1 01 00 00 00	mov esi, 1	
00005561:95558922	48 8b 3d 57 2d 20 00	mov rdi, [rel.0x55619575b680]	
00005561:95558929	b8 00 00 00 00	mov eax, 0	
00005561:9555892e	e8 7d f5 ff ff	call bomb!_fprintf_chk@plt	

没什么头绪，但是第一次运行的时候并不知道什么断点什么的，当时就运行到一处看到 DrEvil，然后抱着试一试的心态就输入了，发现从第六个破解之后出现了隐藏环节，所以我就在 edb 中第六 phase 的 defuse 按 F7 点进去了，好像印证猜想了，而且第一行的比较是相等的，说明了破解了所有阶段才能运行把隐藏阶段引出的代码，中间的

8d 7c 24 10	ret	
8d 35 8a 0f 00 00	lea rdi, [rsp+0x10]	
ec fc ff ff	lea rsi, [rel.0x55619555986a]	ASCII "DrEvil"
c0	call bomb!strings_not_equal	
	test eax, eax	

无疑是验证了隐藏字符串答案正确性

00005561:955588fc	e8 af f4 ff ff	call bomb!puts@plt
00005561:95558901	b8 00 00 00 00	mov eax, 0
00005561:95558906	e8 ce fb ff ff	call bomb!secret_phase
00005561:9555890b	eb b6	jmp 0x5561955588c3
00005561:9555890d	48 83 ec 08	sub rsp, 8
00005561:95558911	b9 00 00 00 00	mov ecx, 0

从此处进入隐藏环节，发现能够跳入到 func7 中，而此是个递归程序。

```

asm - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
1499: c3          retq

000000000000149a <fun7>:
149a: 48 85 ff      test %rdi,%rdi
149d: 74 34         je  14d3 <fun7+0x39>
149f: 48 83 ec 08   sub $0x8,%rsp
14a3: 8b 17         mov (%rdi),%edx //edx = rdi
14a5: 39 f2         cmp %esi,%edx //比较edx和esi
14a7: 7f 0e         jg  14b7 <fun7+0x1d> //大于则跳转14b7
14a9: b8 00 00 00 00 mov $0x0,%eax //否则eax = 0
14ae: 39 f2         cmp %esi,%edx //比较edx和esi
14b0: 75 12         jne 14c4 <fun7+0x2a> //不等于则跳转14c4
14b2: 48 83 c4 08   add $0x8,%rsp
14b6: c3          retq
14b7: 48 8b 7f 08   mov 0x8(%rdi),%rdi //rdi = rdi + 8
14bb: e8 da ff ff ff callq 149a <fun7> //递归调用
14c0: 01 c0         add %eax,%eax //eax*2
14c2: eb ee         jmp 14b2 <fun7+0x18> //跳转14b2 (关)
14c4: 48 8b 7f 10   mov 0x10(%rdi),%rdi //rdi = rdi + 16
14c8: e8 cd ff ff ff callq 149a <fun7> //递归调用
14cd: 8d 44 00 01   lea 0x1(%rax,%rax,1),%eax //eax = eax + eax + 1
14d1: eb df         jmp 14b2 <fun7+0x18>
14d3: b8 ff ff ff ff mov $0xffffffff,%eax
14d8: c3          retq

```

大致意思就是这样，具体各种寄存器的值还得进 edb 中看，最后直接运行发现 edi 的值更新为 0x24，0x8, 0x6, 0x1 分别和读取的数做比较，不等于进入递归等于向下运行，,等于则运行通过，因而答案不唯一，上述均为答案。有点侥幸

第 4 章 总结

4.1 请总结本次实验的收获

```
fs1170301027@ubuntu:~/bomb129$ ./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!
4 0
Halfway there!
11 18 DrEvil
So you got that one. Try this one.
abcdhn
Good work! On to the next...
2 4 1 5 6 3
Curses, you've found the secret phase!
But finding it and solving it are quite different...
1
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
```

感觉对汇编语言理解加深了，好多语句都能读懂了。。做的时候不停百度不停翻ppt，现在感觉一扫阴霾，注释写多了，我感觉现在短一点的 phase 应该可以不用 edb 和 gdb 分析了，感触颇深。

4.2 请给出对本次实验内容的建议

没什么更好的建议，感觉很棒了

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>.