哈爾濱工業大學

实验报告

实 验（二）

题　　目 Binary Bomb

二进制炸弹

专　　业 人工智能

学　　号 2022113416

班　　级 2203601

学　　生 刘子康

指 导 教 师 吴锐

实 验 地 点 G715

实 验 日 期 2024.3.31

计算学部

目 录

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；3.2GHz；16G RAM；512G SSD

1.2.2 软件环境

Windows11 64 位；Vmware 17；Ubuntu 22.04 LTS 64 位

1.2.3 开发工具

Visual Studio 2022 64 位；CodeBlocks；gcc、g++；vi/vim/gedit

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 hello.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。

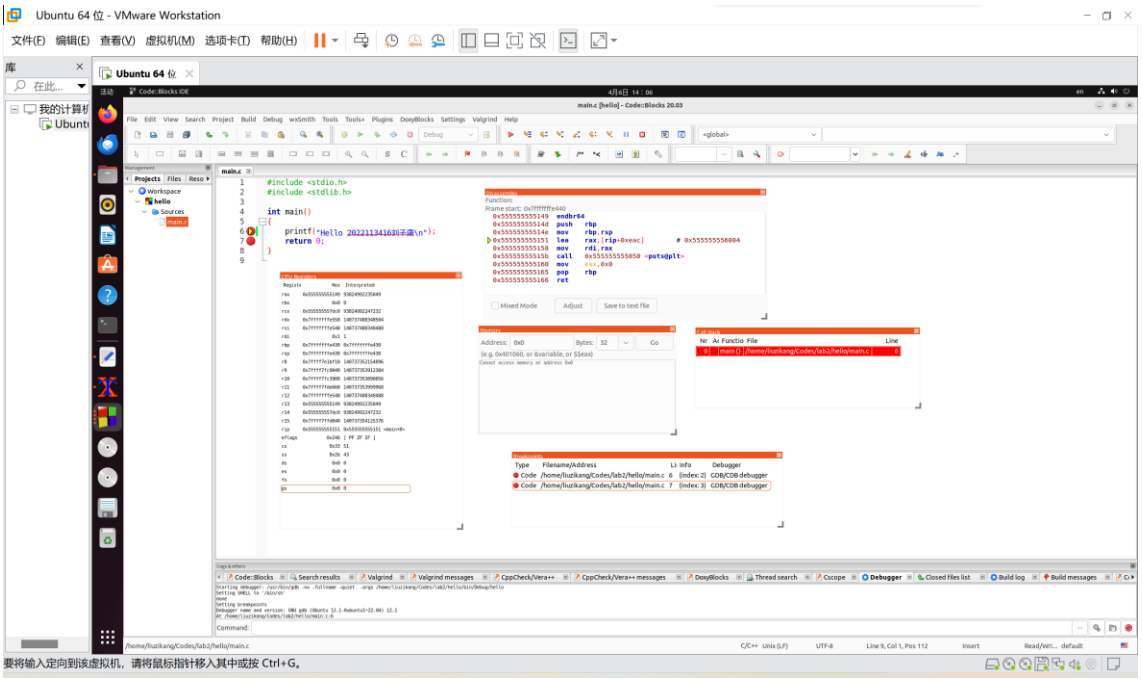

图 2-1 Ubuntu 下 CodeBlocks 反汇编截图

2.2 Ubuntu 下 EDB 运行环境建立（10 分）

用 EDB 调试 hello.c 的执行文件，截图，要求同 2.1

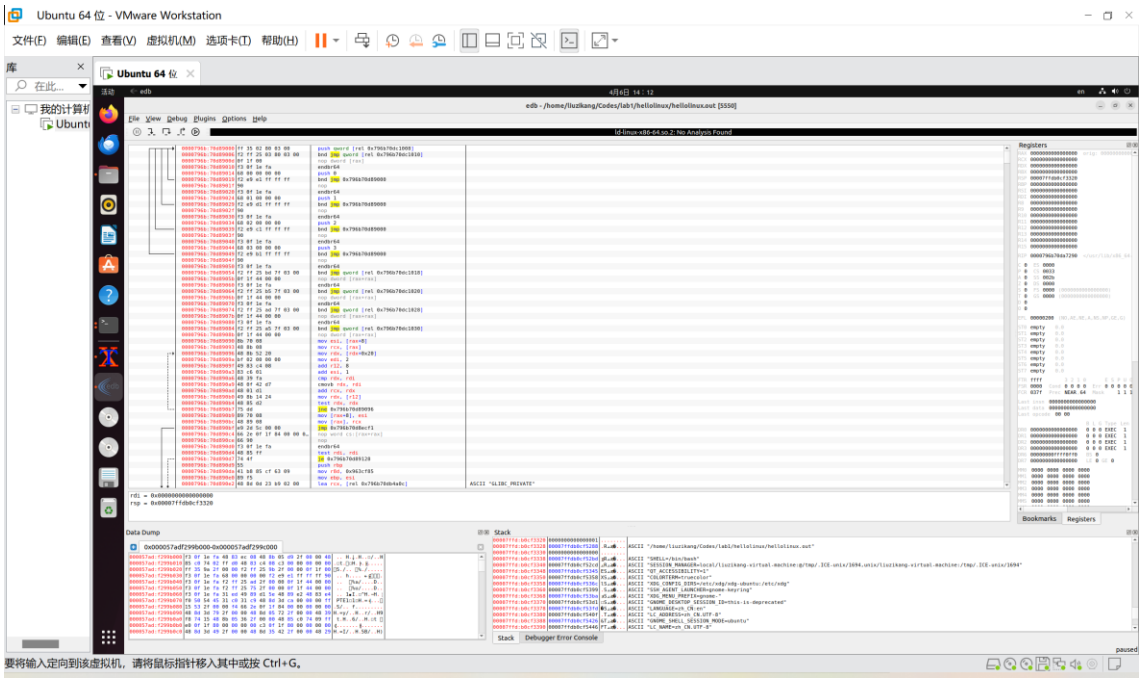

图 2-2　Ubuntu 下 EDB 截图

第 3 章 各阶段炸弹破解与分析

每阶段 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：Houses will begat jobs, jobs will begat houses.

破解过程：

（1）用户输入的字符串经过 readline 函数存储在%edi，309 行将立即数 0x403150 赋给%esi， 310 行调用 strings_not_equal 字符串比较函数，参数为%edi 和%esi，若返回值%eax 为 0（即不相等）则炸弹爆炸。

```
309   4013fd:      be 50 31 40 00          mov    $0x403150,%esi
310   401402:      e8 2d 04 00 00          call   401834 <strings_not_equal>
311   401407:      85 c0                   test   %eax,%eax
312   401409:      75 02                   jne    40140d <phase_1+0x14>
```

（2）推测拆弹密码存放在地址 0x403150，使用 gdb 指令 x/s 0x403150 查看。

```
liuzikang@liuzikang-virtual-machine:~/Codes/lab2$ chmod 744 bomb
liuzikang@liuzikang-virtual-machine:~/Codes/lab2$ gdb bomb
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...
(gdb) b main
Breakpoint 1 at 0x4012a6: file bomb.c, line 37.
(gdb) r
Starting program: /home/liuzikang/Codes/lab2/bomb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0x7fffffffdf28) at bomb.c:37
warning: Source file is more recent than executable.
37      {
(gdb) n
45          if (argc == 1) {
(gdb) n
46              infile = stdin;
(gdb) n
67          initialize_bomb();
(gdb) n
69          printf("Welcome to my fiendish little bomb. You have 6 phases with\n");
(gdb) n
Welcome to my fiendish little bomb. You have 6 phases with
70          printf("which to blow yourself up. Have a nice day!\n");
(gdb) n
which to blow yourself up. Have a nice day!
```

```
73          input = read_line();             /* Get input
(gdb) n
hello world
74          phase_1(input);                  /* Run the phase
(gdb) x/10i $rip
=> 0x401303 <main+93>:  call   0x4013f9 <phase_1>
   0x401308 <main+98>:  call   0x401ab9 <phase_defused>
   0x40130d <main+103>: mov    $0x4030f8,%edi
   0x401312 <main+108>: call   0x401060 <puts@plt>
   0x401317 <main+113>: call   0x40198e <read_line>
   0x40131c <main+118>: mov    %rax,%rdi
   0x40131f <main+121>: call   0x401414 <phase_2>
   0x401324 <main+126>: call   0x401ab9 <phase_defused>
   0x401329 <main+131>: mov    $0x40303d,%edi
   0x40132e <main+136>: call   0x401060 <puts@plt>
(gdb) si
phase_1 (input=0x405780 <input_strings> "hello world") at phases.c:20
20      phases.c: 没有那个文件或目录.
(gdb) si
0x00000000004013fa      20      in phases.c
(gdb) x/10i $rip
=> 0x4013fa <phase_1+1>:        mov    %rsp,%rbp
   0x4013fd <phase_1+4>:        mov    $0x403150,%esi
   0x401402 <phase_1+9>:        call   0x401834 <strings_not_equal>
   0x401407 <phase_1+14>:       test   %eax,%eax
   0x401409 <phase_1+16>:       jne    0x40140d <phase_1+20>
   0x40140b <phase_1+18>:       pop    %rbp
   0x40140c <phase_1+19>:       ret
   0x40140d <phase_1+20>:       call   0x401930 <explode_bomb>
   0x401412 <phase_1+25>:       jmp    0x40140b <phase_1+18>
   0x401414 <phase_2>:  push   %rbp
(gdb) x/s0x403150
0x403150:       "Houses will begat jobs, jobs will begat houses."
(gdb) q
A debugging session is active.

        Inferior 1 [process 19829] will be killed.

Quit anyway? (y or n) y
```

3.2 阶段 2 的破解与分析

密码如下：0 1 3 6 10 15

破解过程：

（1）由 322-324 行申请了 48 字节的内存空间，并调用 read_six_numbers 函数，推测阶段 2 应输入 6 个数字，并且首个数字应非负，否则炸弹爆炸。

```
322   401419:       48 83 ec 28             sub    $0x28,%rsp
323   40141d:       48 8d 75 d0             lea    -0x30(%rbp),%rsi
324   401421:       e8 2c 05 00 00          call   401952 <read_six_numbers>
325   401426:       83 7d d0 00             cmpl   $0x0,-0x30(%rbp)
326   40142a:       78 07                   js     401433 <phase_2+0x1f>
```

（2）327-342 行是循环体，%ebx 是初值为 1 的计数器，每次循环结束后加一，大于 5 时跳出循环（333-334 行）；在一次循环中，%ecx 存储计算结果（%ecx= %ebx+(%rbp-0x30)+4*(%rbx-1)），输入的数字（(%rbp-0x30)+4*%rax）应等于%ecx；因此输入的 6 个数字应为 n, n+1, n+1+2, n+1+2+3, n+1+2+3+4, n+1+2+3+4+5，其中 n 非负且可由用户决定。

```
327   40142c:       bb 01 00 00 00          mov    $0x1,%ebx
328   401431:       eb 0f                   jmp    401442 <phase_2+0x2e>
329   401433:       e8 f8 04 00 00          call   401930 <explode_bomb>
330   401438:       eb f2                   jmp    40142c <phase_2+0x18>
331   40143a:       e8 f1 04 00 00          call   401930 <explode_bomb>
332   40143f:       83 c3 01                add    $0x1,%ebx
333   401442:       83 fb 05                cmp    $0x5,%ebx
334   401445:       7f 17                   jg     40145e <phase_2+0x4a>
335   401447:       48 63 c3                movslq %ebx,%rax
336   40144a:       8d 53 ff                lea    -0x1(%rbx),%edx
337   40144d:       48 63 d2                movslq %edx,%rdx
338   401450:       89 d9                   mov    %ebx,%ecx
339   401452:       03 4c 95 d0             add    -0x30(%rbp,%rdx,4),%ecx
340   401456:       39 4c 85 d0             cmp    %ecx,-0x30(%rbp,%rax,4)
341   40145a:       74 e3                   je     40143f <phase_2+0x2b>
342   40145c:       eb dc                   jmp    40143a <phase_2+0x26>
```

3.3 阶段 3 的破解与分析

密码如下：0 -231

破解过程：

（1）%eax 为 isoc99_sscanf 的返回参数，代表输入参数的个数，则行可知应至少输入两个参数，否则炸弹爆炸。

```
352   40146d:      48 8d 4d f8             lea    -0x8(%rbp),%rcx
353   401471:      48 8d 55 fc             lea    -0x4(%rbp),%rdx
354   401475:      be 4f 33 40 00          mov    $0x40334f,%esi
355   40147a:      b8 00 00 00 00          mov    $0x0,%eax
356   40147f:      e8 8c fc ff ff          call   401110 <__isoc99_sscanf@plt>
357   401484:      83 f8 01                cmp    $0x1,%eax
358   401487:      7e 11                   jle    40149a <phase_3+0x35>
```

```
(gdb) x/s 0x40334f
0x40334f:       "%d %d"
```

（2）359-361 行说明第一个参数应小于等于 7，否则炸弹爆炸；

```
359   401489:      8b 45 fc                mov    -0x4(%rbp),%eax
360   40148c:      83 f8 07                cmp    $0x7,%eax
361   40148f:      77 7b                   ja     40150c <phase_3+0xa7>
362   401491:      89 c0                   mov    %eax,%eax
363   401493:      ff 24 c5 c0 31 40 00    jmp    *0x4031c0(,%rax,8)
```

362-363 行根据第一个参数的值跳转到 0x4031c0+8*%rax 处存储的地址，不同%rax 对应的跳转地址如下图所示。

```
(gdb) x/s *0x4031c0
0x4014db <phase_3+118>: "\270S\001"
(gdb) x/s *(0x4031c0+8)
0x4014a1 <phase_3+60>:  "\270"
(gdb) x/s *(0x4031c0+16)
0x4014e2 <phase_3+125>: "\270"
(gdb) x/s *(0x4031c0+24)
0x4014e9 <phase_3+132>: "\270"
(gdb) x/s *(0x4031c0+32)
0x4014f0 <phase_3+139>: "\270"
(gdb) x/s *(0x4031c0+40)
0x4014f7 <phase_3+146>: "\270"
(gdb) x/s *(0x4031c0+48)
0x4014fe <phase_3+153>: "\270"
```

（3）366-380 行说明第一个参数的值应小于等于 5，同时第二个参数的值应与通过一系列加减得出的%eax 的值相等，否则炸弹爆炸。

```
366   4014a1:       b8 00 00 00 00          mov    $0x0,%eax
367   4014a6:       2d 37 02 00 00          sub    $0x237,%eax
368   4014ab:       05 e0 03 00 00          add    $0x3e0,%eax
369   4014b0:       2d e3 03 00 00          sub    $0x3e3,%eax
370   4014b5:       05 e3 03 00 00          add    $0x3e3,%eax
371   4014ba:       2d e3 03 00 00          sub    $0x3e3,%eax
372   4014bf:       05 e3 03 00 00          add    $0x3e3,%eax
373   4014c4:       2d e3 03 00 00          sub    $0x3e3,%eax
374   4014c9:       83 7d fc 05             cmpl   $0x5,-0x4(%rbp)
375   4014cd:       7f 05                   jg     4014d4 <phase_3+0x6f>
376   4014cf:       39 45 f8                cmp    %eax,-0x8(%rbp)
377   4014d2:       74 05                   je     4014d9 <phase_3+0x74>
378   4014d4:       e8 57 04 00 00          call   401930 <explode_bomb>
379   4014d9:       c9                      leave
380   4014da:       c3                      ret
```

（4）故第 3 阶段通过输入的第一个参数（0~5）确定跳转地址，从而确定%eax 的初始值以及从何处开始进行一系列加减，并且输入的第二个参数应与计算结果相等。以第一个参数是 0 为例，%eax 初始值为 0x153，进行 367-373 行运算，结果为 0xffffff19(-231)，因此输入字符串应为“0 -231”。

```
(gdb) x/10i $rip
=> 0x4014cd <phase_3+104>:      jg     0x4014d4 <phase_3+111>
   0x4014cf <phase_3+106>:      cmp    %eax,-0x8(%rbp)
   0x4014d2 <phase_3+109>:      je     0x4014d9 <phase_3+116>
   0x4014d4 <phase_3+111>:      call   0x401930 <explode_bomb>
   0x4014d9 <phase_3+116>:      leave
   0x4014da <phase_3+117>:      ret
   0x4014db <phase_3+118>:      mov    $0x153,%eax
   0x4014e0 <phase_3+123>:      jmp    0x4014a6 <phase_3+65>
   0x4014e2 <phase_3+125>:      mov    $0x0,%eax
   0x4014e7 <phase_3+130>:      jmp    0x4014ab <phase_3+70>
(gdb) si
0x00000000004014cf      96      in phases.c
(gdb) si
0x00000000004014d2      96      in phases.c
(gdb) info reg
rax            0xffffff19          4294967065
```

3.4 阶段 4 的破解与分析

密码如下：14 7

破解过程：只需分析汇编指令即可。

（1）由 428-434 行可知应输入两个参数，否则炸弹爆炸。

```
428   40155a:      48 8d 4d f8             lea    -0x8(%rbp),%rcx
429   40155e:      48 8d 55 fc             lea    -0x4(%rbp),%rdx
430   401562:      be 4f 33 40 00          mov    $0x40334f,%esi
431   401567:      b8 00 00 00 00          mov    $0x0,%eax
432   40156c:      e8 9f fb ff ff          call   401110 <__isoc99_sscanf@plt>
433   401571:      83 f8 02                cmp    $0x2,%eax
434   401574:      75 0c                   jne    401582 <phase_4+0x30>
```

（2）由 435-440 行可知第一个参数应大于等于 0 且小于等于 14。

```
435   401576:      8b 45 fc                mov    -0x4(%rbp),%eax
436   401579:      85 c0                   test   %eax,%eax
437   40157b:      78 05                   js     401582 <phase_4+0x30>
438   40157d:      83 f8 0e                cmp    $0xe,%eax
439   401580:      7e 05                   jle    401587 <phase_4+0x35>
440   401582:      e8 a9 03 00 00          call   401930 <explode_bomb>
```

（3）441-446 行调用 func4 函数，参数为%edx（初值 14）、%esi（初值 0）和%edi（第一个参数），且返回值必须为 7，否则炸弹爆炸，通过穷举 0~14 并查看返回值可找到返回值 7 对应的输入参数应为 14。

```
441   401587:      ba 0e 00 00 00          mov    $0xe,%edx
442   40158c:      be 00 00 00 00          mov    $0x0,%esi
443   401591:      8b 7d fc                mov    -0x4(%rbp),%edi
444   401594:      e8 7f ff ff ff          call   401518 <func4>
445   401599:      83 f8 07                cmp    $0x7,%eax
446   40159c:      75 06                   jne    4015a4 <phase_4+0x52>
```

（4）func4 是一个递归函数，有三个形参，416 和 420 行进行了递归调用；函数返回值为 1/2*(signal(edx-esi)+edx- esi)+esi，等于 edi 时退出，大于时 esi =1/2*(signal(edx - esi)+ edx - esi)+ esi +1，再次调用 func4 函数并返回结果的 2 倍+1，小于时 edx =(signal(edx - esi)+ edx - esi)+ esi -1，调用 fun4 返回结果的 2 倍；通过返回值 7 逆推可得到输入的第一个参数应为 14。

```
399 0000000000401518 <func4>:
400   401518:       55                      push   %rbp
401   401519:       48 89 e5                mov    %rsp,%rbp
402   40151c:       89 d1                   mov    %edx,%ecx
403   40151e:       29 f1                   sub    %esi,%ecx
404   401520:       89 c8                   mov    %ecx,%eax
405   401522:       c1 e8 1f                shr    $0x1f,%eax
406   401525:       01 c8                   add    %ecx,%eax
407   401527:       d1 f8                   sar    %eax
408   401529:       01 f0                   add    %esi,%eax
409   40152b:       39 f8                   cmp    %edi,%eax
410   40152d:       7f 09                   jg     401538 <func4+0x20>
411   40152f:       7c 13                   jl     401544 <func4+0x2c>
412   401531:       b8 00 00 00 00          mov    $0x0,%eax
413   401536:       5d                      pop    %rbp
414   401537:       c3                      ret
415   401538:       8d 50 ff                lea    -0x1(%rax),%edx
416   40153b:       e8 d8 ff ff ff          call   401518 <func4>
417   401540:       01 c0                   add    %eax,%eax
418   401542:       eb f2                   jmp    401536 <func4+0x1e>
419   401544:       8d 70 01                lea    0x1(%rax),%esi
420   401547:       e8 cc ff ff ff          call   401518 <func4>
421   40154c:       8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
422   401550:       eb e4                   jmp    401536 <func4+0x1e>
```

（5）由 447-449 行可知输入第二个参数应为 7，否则炸弹爆炸，故阶段 4 的拆弹密码为“14 7”。

```
447   40159e:       83 7d f8 07             cmpl   $0x7,-0x8(%rbp)
448   4015a2:       74 05                   je     4015a9 <phase_4+0x57>
449   4015a4:       e8 87 03 00 00          call   401930 <explode_bomb>
```

3.5 阶段 5 的破解与分析

密码如下：888889

破解过程：

（1）首先分析汇编指令，459-461 行说明输入的字符串长度应为 6，否则炸弹爆炸。

```
459   4015b7:       e8 64 02 00 00          call   401820 <string_length>
460   4015bc:       83 f8 06                cmp    $0x6,%eax
461   4015bf:       75 25                   jne    4015e6 <phase_5+0x3b>
```

（2）463-471 行是指针遍历字符串过程，%eax 计数器记录指针移位，467-468 行将%rbx+%rdx 处的值以 0 扩展形式赋给%edx 并保留二进制低四位；

```
462   4015c1:       b9 00 00 00 00          mov    $0x0,%ecx
463   4015c6:       b8 00 00 00 00          mov    $0x0,%eax
464   4015cb:       83 f8 05                cmp    $0x5,%eax
465   4015ce:       7f 1d                   jg     4015ed <phase_5+0x42>
466   4015d0:       48 63 d0                movslq %eax,%rdx
467   4015d3:       0f b6 14 13             movzbl (%rbx,%rdx,1),%edx
468   4015d7:       83 e2 0f                and    $0xf,%edx
469   4015da:       03 0c 95 00 32 40 00    add    0x403200(,%rdx,4),%ecx
470   4015e1:       83 c0 01                add    $0x1,%eax
471   4015e4:       eb e5                   jmp    4015cb <phase_5+0x20>
472   4015e6:       e8 45 03 00 00          call   401930 <explode_bomb>
473   4015eb:       eb d4                   jmp    4015c1 <phase_5+0x16>
474   4015ed:       83 f9 1b                cmp    $0x1b,%ecx
475   4015f0:       75 07                   jne    4015f9 <phase_5+0x4e>
```

%ecx 是初值为 0 的累加器，469 行将 0x403200+4*%rdx 处的值加到%ecx，字符串遍历结束后若%ecx 的值等于 27 则通过，其中 0x403204-0x403240 处的值如下图所示；

```
(gdb) x/s *0x403204
0xa:    <error: Cannot access memory at address 0xa>
(gdb) x/s 0x403204
0x403204 <array.3403+4>:        "\n"
(gdb) x/s *0x403208
0x6:    <error: Cannot access memory at address 0x6>
(gdb) x/s *0x403212
0x100000:       <error: Cannot access memory at address 0x100000>
(gdb) x/s *0x403216
0x90000:        <error: Cannot access memory at address 0x90000>
(gdb) x/s *0x403220
0x4:    <error: Cannot access memory at address 0x4>
(gdb) x/s *0x403224
0x7:    <error: Cannot access memory at address 0x7>
(gdb) x/s *0x403228
0xe:    <error: Cannot access memory at address 0xe>
(gdb) x/s *0x403232
0x80000:        <error: Cannot access memory at address 0x80000>
(gdb) x/s *0x403236
0xf0000:        <error: Cannot access memory at address 0xf0000>
(gdb) x/s *0x403240
0x79206f53:     <error: Cannot access memory at address 0x79206f53>
```

（3）可知第 5 阶段需要通过 6 次累加使%ecx 的值从 0 变为 27，其中每次加法的值为输入的 6 位字符串每一位数字对应的 0x403200+4*%rdx 处存储的值，通过指针移位进行循环。通过分析采用 5 次 0x403220（4）和 1 次 0x403224 处（7），因此前五位数字为 0x20÷4=8，第六位数字为 0x24÷4=9。

3.6 阶段 6 的破解与分析

密码如下：5 2 1 3 4 6

破解过程：

（1）首先分析汇编指令，与阶段 2 相似，程序读入 6 个数字，并将第一个数字存在-0x40(%rbp)处。

```
489   401609:       48 83 ec 58             sub    $0x58,%rsp
490   40160d:       48 8d 75 c0             lea    -0x40(%rbp),%rsi
491   401611:       e8 3c 03 00 00          call   401952 <read_six_numbers>
```

（2）497-516 行是一个两层循环，将输入的 6 个数字彼此比较，若存在相等数字则炸弹爆炸，可知输入的 6 个数字应各不相等。

```
497   40162a:       83 c3 01                add    $0x1,%ebx
498   40162d:       83 fb 05                cmp    $0x5,%ebx
499   401630:       7f 12                   jg     401644 <phase_6+0x44>
500   401632:       49 63 c4                movslq %r12d,%rax
501   401635:       48 63 d3                movslq %ebx,%rdx
502   401638:       8b 7c 95 c0             mov    -0x40(%rbp,%rdx,4),%edi
503   40163c:       39 7c 85 c0             cmp    %edi,-0x40(%rbp,%rax,4)
504   401640:       75 e8                   jne    40162a <phase_6+0x2a>
505   401642:       eb e1                   jmp    401625 <phase_6+0x25>
506   401644:       45 89 ec                mov    %r13d,%r12d
507   401647:       41 83 fc 05             cmp    $0x5,%r12d
508   40164b:       7f 19                   jg     401666 <phase_6+0x66>
509   40164d:       49 63 c4                movslq %r12d,%rax
510   401650:       8b 44 85 c0             mov    -0x40(%rbp,%rax,4),%eax
511   401654:       83 e8 01                sub    $0x1,%eax
512   401657:       83 f8 05                cmp    $0x5,%eax
513   40165a:       77 c2                   ja     40161e <phase_6+0x1e>
514   40165c:       45 8d 6c 24 01          lea    0x1(%r12),%r13d
515   401661:       44 89 eb                mov    %r13d,%ebx
516   401664:       eb c7                   jmp    40162d <phase_6+0x2d>
```

（3）510-513 行，将输入的参数赋给%eax，若%eax>5（即参数>6）则炸弹爆炸，故输入的 6 个参数应都小于等于 6，结合（2）推测 6 个数字为 1~6 的。

```
510   401650:       8b 44 85 c0             mov    -0x40(%rbp,%rax,4),%eax
511   401654:       83 e8 01                sub    $0x1,%eax
512   401657:       83 f8 05                cmp    $0x5,%eax
513   40165a:       77 c2                   ja     40161e <phase_6+0x1e>
```

（4）520-522 行，通过一个 6 次的循环，将输入的参数（用 arg[i]表示）替换为 7-arg[i]。

```
519   40166d:       48 63 c8                movslq %eax,%rcx
520   401670:       ba 07 00 00 00          mov    $0x7,%edx
521   401675:       2b 54 8d c0             sub    -0x40(%rbp,%rcx,4),%edx
522   401679:       89 54 8d c0             mov    %edx,-0x40(%rbp,%rcx,4)
523   40167d:       83 c0 01                add    $0x1,%eax
524   401680:       83 f8 05                cmp    $0x5,%eax
525   401683:       7e e8                   jle    40166d <phase_6+0x6d>
```

（5）526-551 行通过循环将链表按照输入的参数顺序重组，例如输入为 3 6 5 2 1 4 时将原链表第三个节点设为表头，第六个节点接在表头之后，以此类推；其中 538 行将立即数 0x4052d0 存放在%edx，推测为链表表头地址。

```
526   401685:       be 00 00 00 00          mov    $0x0,%esi
527   40168a:       eb 18                   jmp    4016a4 <phase_6+0xa4>
528   40168c:       48 8b 52 08             mov    0x8(%rdx),%rdx
529   401690:       83 c0 01                add    $0x1,%eax
530   401693:       48 63 ce                movslq %esi,%rcx
531   401696:       39 44 8d c0             cmp    %eax,-0x40(%rbp,%rcx,4)
532   40169a:       7f f0                   jg     40168c <phase_6+0x8c>
533   40169c:       48 89 54 cd 90          mov    %rdx,-0x70(%rbp,%rcx,8)
534   4016a1:       83 c6 01                add    $0x1,%esi
535   4016a4:       83 fe 05                cmp    $0x5,%esi
536   4016a7:       7f 0c                   jg     4016b5 <phase_6+0xb5>
537   4016a9:       b8 01 00 00 00          mov    $0x1,%eax
538   4016ae:       ba d0 52 40 00          mov    $0x4052d0,%edx
539   4016b3:       eb de                   jmp    401693 <phase_6+0x93>
540   4016b5:       48 8b 5d 90             mov    -0x70(%rbp),%rbx
541   4016b9:       48 89 d9                mov    %rbx,%rcx
542   4016bc:       b8 01 00 00 00          mov    $0x1,%eax
543   4016c1:       eb 12                   jmp    4016d5 <phase_6+0xd5>
544   4016c3:       48 63 d0                movslq %eax,%rdx
545   4016c6:       48 8b 54 d5 90          mov    -0x70(%rbp,%rdx,8),%rdx
546   4016cb:       48 89 51 08             mov    %rdx,0x8(%rcx)
547   4016cf:       83 c0 01                add    $0x1,%eax
548   4016d2:       48 89 d1                mov    %rdx,%rcx
549   4016d5:       83 f8 05                cmp    $0x5,%eax
550   4016d8:       7e e9                   jle    4016c3 <phase_6+0xc3>
551   4016da:       48 c7 41 08 00 00 00    movq   $0x0,0x8(%rcx)
```

（6）553-564 行，从表头开始链表每个节点%rbx 依次与下一个节点 0x8(%rbx)比较，若出现递增则炸弹爆炸，故重组后的链表节点元素应为递减的。

```
553   4016e2:       41 bc 00 00 00 00       mov    $0x0,%r12d
554   4016e8:       eb 08                   jmp    4016f2 <phase_6+0xf2>
555   4016ea:       48 8b 5b 08             mov    0x8(%rbx),%rbx
556   4016ee:       41 83 c4 01             add    $0x1,%r12d
557   4016f2:       41 83 fc 04             cmp    $0x4,%r12d
558   4016f6:       7f 11                   jg     401709 <phase_6+0x109>
559   4016f8:       48 8b 43 08             mov    0x8(%rbx),%rax
560   4016fc:       8b 00                   mov    (%rax),%eax
561   4016fe:       39 03                   cmp    %eax,(%rbx)
562   401700:       7d e8                   jge    4016ea <phase_6+0xea>
563   401702:       e8 29 02 00 00          call   401930 <explode_bomb>
564   401707:       eb e1                   jmp    4016ea <phase_6+0xea>
```

（7）综上可知，该程序读取输入的 6 个数字，并按照对应的顺序将链表重组，使得重组后的链表元素递减排列，通过表头地址 0x4052d0 查询链表元素可知正确的顺序为 2 5 6 4 3 1，又因程序用 7-arg[i]替换 arg[i]，故密码应为 5 2 1 3 4 6。

```
(gdb) x/12xg 0x4052d0
0x4052d0 <node1>:       0x000000010000008d      0x00000000004052e0
0x4052e0 <node2>:       0x00000002000003b4      0x00000000004052f0
0x4052f0 <node3>:       0x0000000300000090      0x0000000000405300
0x405300 <node4>:       0x000000040000014b      0x0000000000405310
0x405310 <node5>:       0x0000000500000278      0x0000000000405320
0x405320 <node6>:       0x00000006000001a8      0x0000000000000000
```

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

（1）更加深入理解了汇编语言的指令和反汇编代码，了解了不同机器级语言程序的 C 语句在汇编语言下的形式，增强了对逆向工程的理解；
（2）掌握了 Linux 系统下使用 gdb 反汇编调试的方法，学会了使用 OllyDBG 和 EDB 等调试工具。

4.2 请给出对本次实验内容的建议

（1）希望可以举例讲解一下每个阶段的拆弹密码和对应机器级语言程序的关系，如阶段 1 是输入的字符串与密码直接比较，阶段 2 是通过循环读取和操作输入的参数；
（2）希望 ppt 内容解释的更详细一点，相关指令添加适当注释。
注：本章为酌情加分项。

参考文献

[1] 深入理解计算机系统 CSAPP（第三版）.

[2] 汇编指令--移位操作 SHL、SHR、SAL、SAR 以及它们的区别 https://blog.csdn.net/deniece1/article/details/103274744.

[3] 【汇编语言】（x86）test 与跳转指令（je jle jge jg jl……）组合的含义 https://blog.csdn.net/weixin_42929607/article/details/106579347.

[4] 汇编语言——跳转指令: JMP、JECXZ、JA、JB、JG、JL、JE、JZ、JS、JC、JO、JP https://blog.csdn.net/wq57885/article/details/80700032.