哈爾濱工業大學

实验报告

实 验（三）

题　　目 Binary Bomb

二进制炸弹

专　　业 计算学部

学　　号 1190200708

班　　级 1903008

学　　生 熊峰

指 导 教 师 吴锐

实 验 地 点 G709

实 验 日 期 2021.4.19

计算机科学与技术学院

目 录

第 1 章 实验基本信息 ..... - 3 -

1.1 实验目的 ..... - 3 -
1.2 实验环境与工具 ..... - 3 -
1.2.1 硬件环境 ..... - 3 -
1.2.2 软件环境 ..... - 3 -
1.2.3 开发工具 ..... - 3 -
1.3 实验预习 ..... - 3 -

第 2 章 实验环境建立 ..... - 5 -

2.1 UBUNTU 下 CODEBLOCKS 反汇编（10 分） ..... - 5 -
2.2 UBUNTU 下 EDB 运行环境建立（10 分） ..... - 6 -

第 3 章 各阶段炸弹破解与分析 ..... - 7 -

3.1 阶段 1 的破解与分析 ..... - 7 -
3.2 阶段 2 的破解与分析 ..... - 9 -
3.3 阶段 3 的破解与分析 ..... - 12 -
3.4 阶段 4 的破解与分析 ..... - 16 -
3.5 阶段 5 的破解与分析 ..... - 18 -
3.6 阶段 6 的破解与分析 ..... - 19 -
3.7 阶段 7 的破解与分析(隐藏阶段) ..... - 23 -

第 4 章 总结 ..... - 28 -

4.1 请总结本次实验的收获 ..... - 28 -
4.2 请给出对本次实验内容的建议 ..... - 28 -

参考文献 ..... -29-

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

X86-64 CPU；3.60GHz；16G RAM；256G SSD；1T SSD

1.2.2 软件环境

Win 10

Ubuntu 20.04.2 LTS

WSL

1.2.3 开发工具

Visual Studio 2019；Vim；GCC；GDB；Code::Blocks; CLion 2020.3.1 x64

1.3 实验预习

上实验课前，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。

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 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。

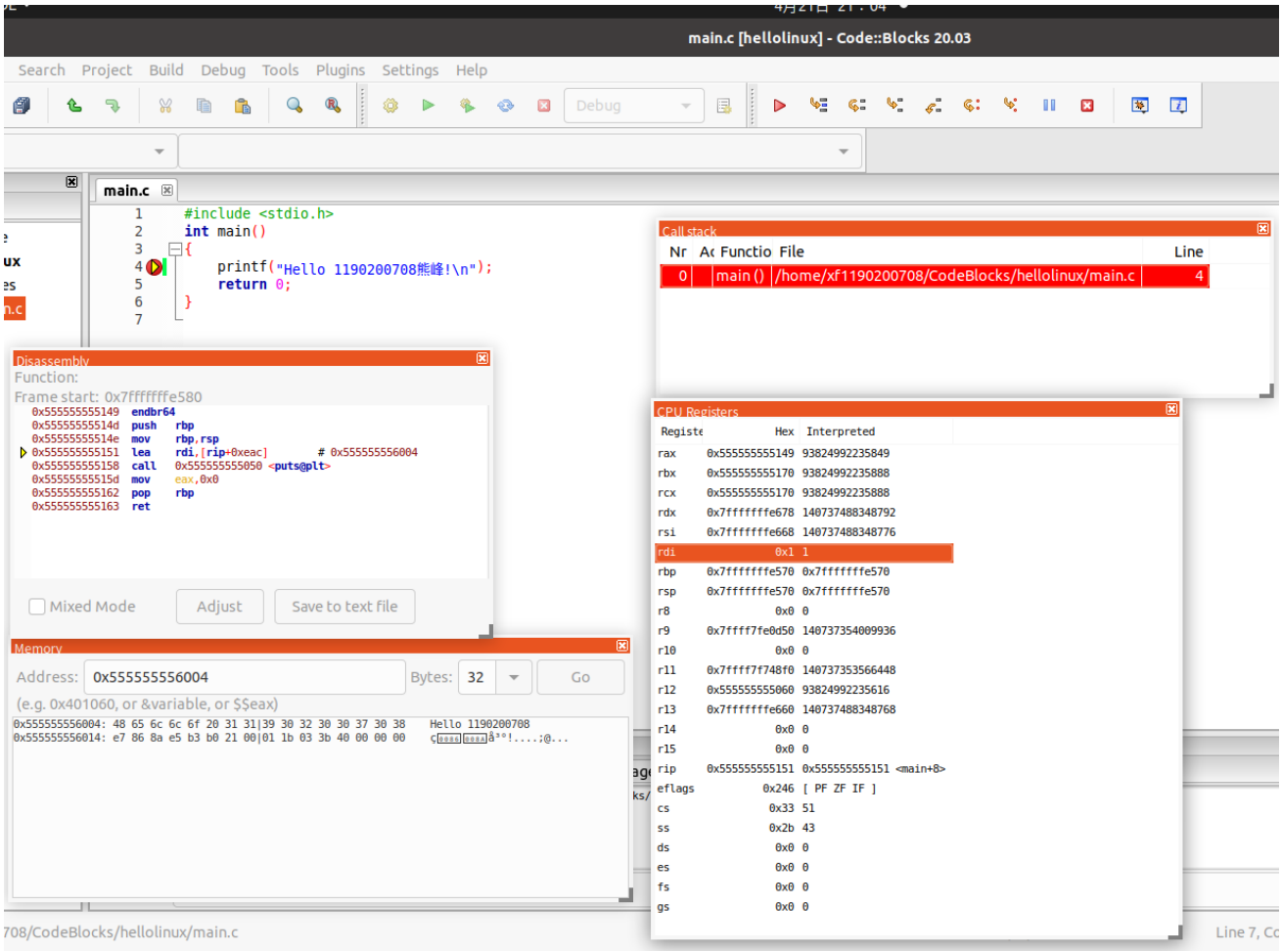


图 2-1 Ubuntu 下 CodeBlocks 反汇编截图

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

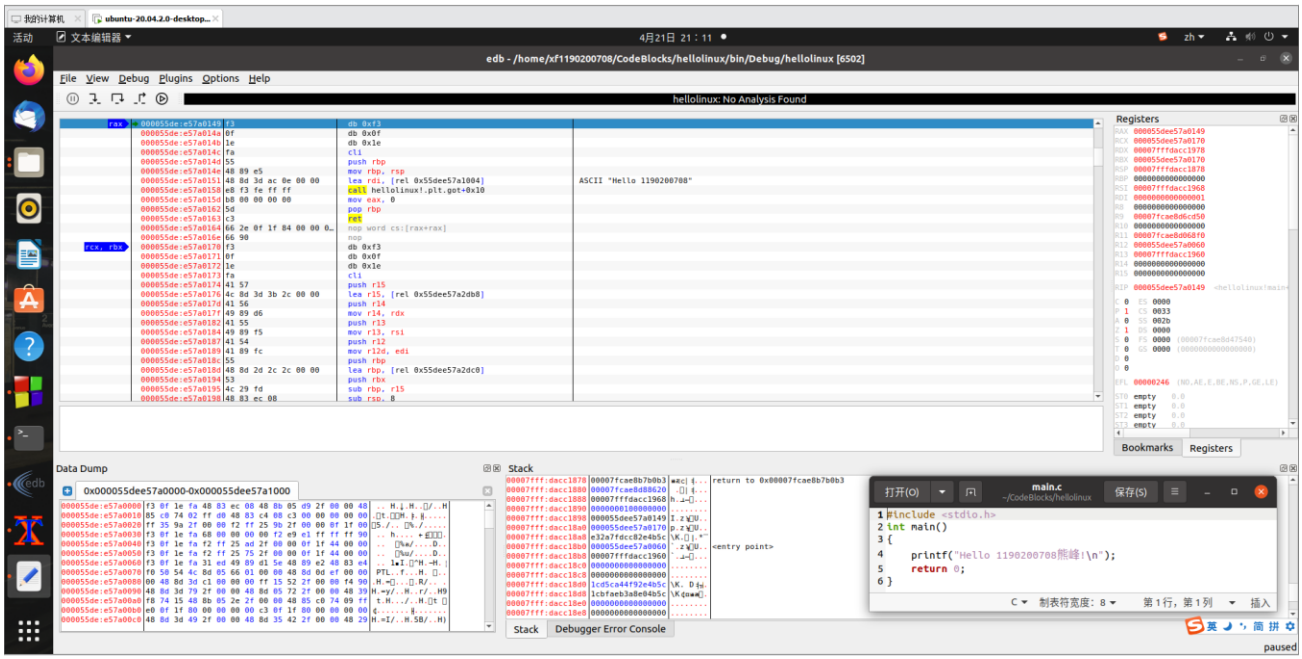


图 2-2　Ubuntu 下 EDB 截图

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：I was trying to give Tina Fey more material.

破解过程：

```
00000000004012a6 <main>:
  4012a6: 55                    push   %rbp
  4012a7: 48 89 e5              mov    %rsp,%rbp
  4012aa: 53                    push   %rbx
  4012ab: 48 83 ec 08           sub    $0x8,%rsp
  4012af: 83 ff 01              cmp    $0x1,%edi
  4012b2: 0f 84 ed 00 00 00     je     4013a5 <main+0xff>
  4012b8: 48 89 f3              mov    %rsi,%rbx
  4012bb: 83 ff 02              cmp    $0x2,%edi
  4012be: 0f 85 14 01 00 00     jne    4013d8 <main+0x132>
  4012c4: 48 8b 7e 08           mov    0x8(%rsi),%rdi
  4012c8: be 04 30 40 00        mov    $0x403004,%esi
  4012cd: e8 5e fe ff ff        callq  401130 <fopen@plt>
  4012d2: 48 89 05 97 44 00 00  mov    %rax,0x4497(%rip)        # 405770 <infile>
  4012d9: 48 85 c0              test   %rax,%rax
  4012dc: 0f 84 d6 00 00 00     je     4013b8 <main+0x112>
  4012e2: e8 af 05 00 00        callq  401896 <initialize_bomb>
  4012e7: bf 88 30 40 00        mov    $0x403088,%edi
  4012ec: e8 6f fd ff ff        callq  401060 <puts@plt>
  4012f1: bf c8 30 40 00        mov    $0x4030c8,%edi
  4012f6: e8 65 fd ff ff        callq  401060 <puts@plt>
  4012fb: e8 93 06 00 00        callq  401993 <read_line>
  401300: 48 89 c7              mov    %rax,%rdi
  401303: e8 f1 00 00 00        callq  4013f9 <phase_1>
  401308: e8 b1 07 00 00        callq  401abe <phase_defused>
```

首先主函数调用 phase_1 函数进行第一次拆弹。

```
00000000004013f9 <phase_1>:
  4013f9: 55                    push   %rbp
  4013fa: 48 89 e5              mov    %rsp,%rbp
  4013fd: be 50 31 40 00        mov    $0x403150,%esi
  401402: e8 32 04 00 00        callq  401839 <strings_not_equal>
  401407: 85 c0                 test   %eax,%eax
  401409: 75 02                 jne    40140d <phase_1+0x14>
  40140b: 5d                    pop    %rbp
  40140c: c3                    retq
  40140d: e8 23 05 00 00        callq  401935 <explode_bomb>
  401412: eb f7                 jmp    40140b <phase_1+0x12>
```

首先对 phase_1 分析，函数通过将地址 0x403150 的内容赋给寄存器%esi，再调用 strings_not_equal 函数，判断输入的字符串与 0x403150 处字符串是否相同，

若%eax 的值为 1，此时函数跳转到 0x40140d，并调用 explode_bomb 函数，引爆炸弹。

```
0000000000401839 <strings_not_equal>:
  401839: 55                    push   %rbp
  40183a: 48 89 e5              mov    %rsp,%rbp
  40183d: 41 55                 push   %r13
  40183f: 41 54                 push   %r12
  401841: 53                    push   %rbx
  401842: 48 83 ec 08           sub    $0x8,%rsp
  401846: 48 89 fb              mov    %rdi,%rbx
  401849: 49 89 f4              mov    %rsi,%r12
  40184c: e8 d4 ff ff ff        callq  401825 <string_length>
  401851: 41 89 c5              mov    %eax,%r13d
  401854: 4c 89 e7              mov    %r12,%rdi
  401857: e8 c9 ff ff ff        callq  401825 <string_length>
  40185c: 41 39 c5              cmp    %eax,%r13d
  40185f: 75 1e                 jne    40187f <strings_not_equal+0x46>
  401861: 0f b6 03              movzbl (%rbx),%eax
  401864: 84 c0                 test   %al,%al
  401866: 74 10                 je     401878 <strings_not_equal+0x3f>
  401868: 41 38 04 24           cmp    %al,(%r12)
  40186c: 75 21                 jne    40188f <strings_not_equal+0x56>
  40186e: 48 83 c3 01           add    $0x1,%rbx
  401872: 49 83 c4 01           add    $0x1,%r12
  401876: eb e9                 jmp    401861 <strings_not_equal+0x28>
  401878: b8 00 00 00 00        mov    $0x0,%eax
  40187d: eb 05                 jmp    401884 <strings_not_equal+0x4b>
  40187f: b8 01 00 00 00        mov    $0x1,%eax
  401884: 48 83 c4 08           add    $0x8,%rsp
  401888: 5b                    pop    %rbx
  401889: 41 5c                 pop    %r12
  40188b: 41 5d                 pop    %r13
  40188d: 5d                    pop    %rbp
  40188e: c3                    retq
  40188f: b8 01 00 00 00        mov    $0x1,%eax
  401894: eb ee                 jmp    401884 <strings_not_equal+0x4b>
```

对 strings_not_equal 函数分析，首先函数判断字符串的长度，如长度不相同则返回值 1。函数通过调用 string_length 确定字符串的长度。分别通过%rdi 和%rsi 保存两个字符串的地址。若两个字符串长度相同，再分别对字符串的每一位比较。

```
(gdb) disass string_length
Dump of assembler code for function string_length:
   0x0000000000401825 <+0>:     mov    $0x0,%eax
   0x000000000040182a <+5>:     cmpb   $0x0,(%rdi)
   0x000000000040182d <+8>:     je     0x401838 <string_length+19>
   0x000000000040182f <+10>:    add    $0x1,%rdi
   0x0000000000401833 <+14>:    add    $0x1,%eax
   0x0000000000401836 <+17>:    jmp    0x40182a <string_length+5>
   0x0000000000401838 <+19>:    retq
End of assembler dump.
```

string_length 函数通过对%eax 进行+1 操作，同时使%rdi 所表示的地址+1，不

断进行位的比较，相当于每次移动位一次，计算出长度。

```
(gdb) disass phase_1
Dump of assembler code for function phase_1:
=> 0x00000000004013f9 <+0>:     push   %rbp
   0x00000000004013fa <+1>:     mov    %rsp,%rbp
   0x00000000004013fd <+4>:     mov    $0x403150,%esi
   0x0000000000401402 <+9>:     callq  0x401839 <strings_not_equal>
   0x0000000000401407 <+14>:    test   %eax,%eax
   0x0000000000401409 <+16>:    jne    0x40140d <phase_1+20>
   0x000000000040140b <+18>:    pop    %rbp
   0x000000000040140c <+19>:    retq
   0x000000000040140d <+20>:    callq  0x401935 <explode_bomb>
   0x0000000000401412 <+25>:    jmp    0x40140b <phase_1+18>
End of assembler dump.
(gdb) x/s 0x403150
0x403150:       "I was trying to give Tina Fey more material."
```

通过反汇编，找到 0x403150 处所存放的字符串。

故 phase_1 的密码为 I was trying to give Tina Fey more material.

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：

```
40130d: bf f8 30 40 00        mov    $0x4030f8,%edi
401312: e8 49 fd ff ff        callq  401060 <puts@plt>
401317: e8 77 06 00 00        callq  401993 <read_line>
40131c: 48 89 c7              mov    %rax,%rdi
40131f: e8 f0 00 00 00        callq  401414 <phase_2>
401324: e8 95 07 00 00        callq  401abe <phase_defused>
```

main 函数首先将读到的内容地址传给%rdi 寄存器，再调用 phase_2 函数。

```
(gdb) disass phase_2
Dump of assembler code for function phase_2:
=> 0x0000000000401414 <+0>:     push   %rbp
   0x0000000000401415 <+1>:     mov    %rsp,%rbp
   0x0000000000401418 <+4>:     push   %rbx
   0x0000000000401419 <+5>:     sub    $0x28,%rsp
   0x000000000040141d <+9>:     lea    -0x30(%rbp),%rsi
   0x0000000000401421 <+13>:    callq  0x401957 <read_six_numbers>
   0x0000000000401426 <+18>:    cmpl   $0x0,-0x30(%rbp)
   0x000000000040142a <+22>:    jne    0x401432 <phase_2+30>
   0x000000000040142c <+24>:    cmpl   $0x1,-0x2c(%rbp)
   0x0000000000401430 <+28>:    je     0x401437 <phase_2+35>
   0x0000000000401432 <+30>:    callq  0x401935 <explode_bomb>
   0x0000000000401437 <+35>:    mov    $0x2,%ebx
   0x000000000040143c <+40>:    jmp    0x401446 <phase_2+50>
   0x000000000040143e <+42>:    callq  0x401935 <explode_bomb>
   0x0000000000401443 <+47>:    add    $0x1,%ebx
   0x0000000000401446 <+50>:    cmp    $0x5,%ebx
   0x0000000000401449 <+53>:    jg     0x401469 <phase_2+85>
   0x000000000040144b <+55>:    movslq %ebx,%rdx
   0x000000000040144e <+58>:    lea    -0x2(%rbx),%ecx
   0x0000000000401451 <+61>:    movslq %ecx,%rcx
   0x0000000000401454 <+64>:    lea    -0x1(%rbx),%eax
   0x0000000000401457 <+67>:    cltq
   0x0000000000401459 <+69>:    mov    -0x30(%rbp,%rax,4),%eax
   0x000000000040145d <+73>:    add    -0x30(%rbp,%rcx,4),%eax
   0x0000000000401461 <+77>:    cmp    %eax,-0x30(%rbp,%rdx,4)
   0x0000000000401465 <+81>:    je     0x401443 <phase_2+47>
   0x0000000000401467 <+83>:    jmp    0x40143e <phase_2+42>
   0x0000000000401469 <+85>:    add    $0x28,%rsp
   0x000000000040146d <+89>:    pop    %rbx
   0x000000000040146e <+90>:    pop    %rbp
   0x000000000040146f <+91>:    retq
End of assembler dump.
```

对 phase_2 函数分析，函数通过判断%ebx 是否大于 5，控制循环的次数为 4，同时还需满足-0x30(%rbp,%rdx,4)与%eax 是否相等，若相等则继续循环，否则调用 explode_bomb 函数引爆炸弹。

```
(gdb) disass read_six_numbers
Dump of assembler code for function read_six_numbers:
   0x0000000000401957 <+0>:     push   %rbp
   0x0000000000401958 <+1>:     mov    %rsp,%rbp
   0x000000000040195b <+4>:     mov    %rsi,%rdx
   0x000000000040195e <+7>:     lea    0x4(%rsi),%rcx
   0x0000000000401962 <+11>:    lea    0x14(%rsi),%rax
   0x0000000000401966 <+15>:    push   %rax
   0x0000000000401967 <+16>:    lea    0x10(%rsi),%rax
   0x000000000040196b <+20>:    push   %rax
   0x000000000040196c <+21>:    lea    0xc(%rsi),%r9
   0x0000000000401970 <+25>:    lea    0x8(%rsi),%r8
   0x0000000000401974 <+29>:    mov    $0x403303,%esi
   0x0000000000401979 <+34>:    mov    $0x0,%eax
   0x000000000040197e <+39>:    callq  0x401110 <__isoc99_sscanf@plt>
   0x0000000000401983 <+44>:    add    $0x10,%rsp
   0x0000000000401987 <+48>:    cmp    $0x5,%eax
   0x000000000040198a <+51>:    jle    0x40198e <read_six_numbers+55>
   0x000000000040198c <+53>:    leaveq
   0x000000000040198d <+54>:    retq
   0x000000000040198e <+55>:    callq  0x401935 <explode_bomb>
End of assembler dump.
```

在对 read_six_numbers 函数分析，根据 phase_2 的函数%rsi = %rsp - 0x30。由 phase_2 可得第一个数为 0，第二个数为 1。

第三个数：%ebx = 2 -> %rdx = 2 -> %ecx = 0 -> %rcx = 0 -> %eax = 1 -> %eax = 1 -> %eax = 1，故第三个数为 1.

第四个数：%ebx = 3 -> %rdx = 3 -> %ecx = 1 -> %rcx = 1 -> %eax -> 2 -> %eax = 1 -> %eax = 2，故第四个数为 2.

第五个数：%ebx = 4 -> %rdx = 4 -> %ecx = 2 -> %rcx = 2 ->%eax = 3 -> %eax = 2 -> %eax = 3，故第五个数位 3.

第六个数：%ebx = 5 -> %rdx = 5 -> %ecx = 3 -> %rcx = 3 -> %eax = 4 -> %eax = 3 -> %eax = 5，故第六个数位 5.

```
(gdb) disass read_six_numbers
Dump of assembler code for function read_six_numbers:
   0x0000000000401957 <+0>:     push   %rbp
   0x0000000000401958 <+1>:     mov    %rsp,%rbp
   0x000000000040195b <+4>:     mov    %rsi,%rdx
   0x000000000040195e <+7>:     lea    0x4(%rsi),%rcx
   0x0000000000401962 <+11>:    lea    0x14(%rsi),%rax
   0x0000000000401966 <+15>:    push   %rax
   0x0000000000401967 <+16>:    lea    0x10(%rsi),%rax
   0x000000000040196b <+20>:    push   %rax
   0x000000000040196c <+21>:    lea    0xc(%rsi),%r9
   0x0000000000401970 <+25>:    lea    0x8(%rsi),%r8
   0x0000000000401974 <+29>:    mov    $0x403303,%esi
   0x0000000000401979 <+34>:    mov    $0x0,%eax
   0x000000000040197e <+39>:    callq  0x401110 <__isoc99_sscanf@plt>
   0x0000000000401983 <+44>:    add    $0x10,%rsp
   0x0000000000401987 <+48>:    cmp    $0x5,%eax
   0x000000000040198a <+51>:    jle    0x40198e <read_six_numbers+55>
   0x000000000040198c <+53>:    leaveq
   0x000000000040198d <+54>:    retq
   0x000000000040198e <+55>:    callq  0x401935 <explode_bomb>
End of assembler dump.
(gdb) x/s 0x403303
0x403303:       "%d %d %d %d %d %d"
```

故由图可知，输入格式为”%d %d %d %d %d %d”，因此密码为 0 1 1 2 3 5.

3.3 阶段 3 的破解与分析

密码如下：0 -158 或 1 -631 或 2 -315 或 3 -464 或 4 0 或 5 -464

破解过程：

```
(gdb) p/x *0x4031b0
$1 = 0x4014e6
(gdb) p/x *0x4031b8
$2 = 0x4014ac
(gdb) p/x *0x4031c0
$3 = 0x4014ed
(gdb) p/x *0x4031c8
$4 = 0x4014f4
(gdb) p/x *0x4031d0
$5 = 0x4014fb
(gdb) p/x *0x4031d8
$6 = 0x401502
(gdb) p/x *0x4031e0
$7 = 0x401509
(gdb) p/x *0x4031e8
$8 = 0x401510
```

```
0000000000401470 <phase_3>:
  401470: 55                      push   %rbp
  401471: 48 89 e5                mov    %rsp,%rbp
  401474: 48 83 ec 10             sub    $0x10,%rsp
  401478: 48 8d 4d f8             lea    -0x8(%rbp),%rcx
  40147c: 48 8d 55 fc             lea    -0x4(%rbp),%rdx
  401480: be 0f 33 40 00          mov    $0x40330f,%esi
  401485: b8 00 00 00 00          mov    $0x0,%eax
  40148a: e8 81 fc ff ff          callq  401110 <__isoc99_sscanf@plt>
  40148f: 83 f8 01                cmp    $0x1,%eax
  401492: 7e 11                   jle    4014a5 <phase_3+0x35>
  401494: 8b 45 fc                mov    -0x4(%rbp),%eax
  401497: 83 f8 07                cmp    $0x7,%eax
  40149a: 77 7b                   ja     401517 <phase_3+0xa7>
  40149c: 89 c0                   mov    %eax,%eax
  40149e: ff 24 c5 b0 31 40 00    jmpq   *0x4031b0(,%rax,8)
  4014a5: e8 8b 04 00 00          callq  401935 <explode_bomb>
  4014aa: eb e8                   jmp    401494 <phase_3+0x24>
  4014ac: b8 00 00 00 00          mov    $0x0,%eax
  4014b1: 2d 3c 01 00 00          sub    $0x13c,%eax
  4014b6: 05 95 00 00 00          add    $0x95,%eax
  4014bb: 2d d0 01 00 00          sub    $0x1d0,%eax
  4014c0: 05 d0 01 00 00          add    $0x1d0,%eax
  4014c5: 2d d0 01 00 00          sub    $0x1d0,%eax
  4014ca: 05 d0 01 00 00          add    $0x1d0,%eax
  4014cf: 2d d0 01 00 00          sub    $0x1d0,%eax
  4014d4: 83 7d fc 05             cmpl   $0x5,-0x4(%rbp)
  4014d8: 7f 05                   jg     4014df <phase_3+0x6f>
  4014da: 39 45 f8                cmp    %eax,-0x8(%rbp)
  4014dd: 74 05                   je     4014e4 <phase_3+0x74>
  4014df: e8 51 04 00 00          callq  401935 <explode_bomb>
  4014e4: c9                      leaveq
  4014e5: c3                      retq
  4014e6: b8 d9 01 00 00          mov    $0x1d9,%eax
  4014eb: eb c4                   jmp    4014b1 <phase_3+0x41>
  4014ed: b8 00 00 00 00          mov    $0x0,%eax
  4014f2: eb c2                   jmp    4014b6 <phase_3+0x46>
  4014f4: b8 00 00 00 00          mov    $0x0,%eax
  4014f9: eb c0                   jmp    4014bb <phase_3+0x4b>
  4014fb: b8 00 00 00 00          mov    $0x0,%eax
  401500: eb be                   jmp    4014c0 <phase_3+0x50>
  401502: b8 00 00 00 00          mov    $0x0,%eax
  401507: eb bc                   jmp    4014c5 <phase_3+0x55>
  401509: b8 00 00 00 00          mov    $0x0,%eax
  40150e: eb ba                   jmp    4014ca <phase_3+0x5a>
  401510: b8 00 00 00 00          mov    $0x0,%eax
  401515: eb b8                   jmp    4014cf <phase_3+0x5f>
  401517: e8 19 04 00 00          callq  401935 <explode_bomb>
  40151c: b8 00 00 00 00          mov    $0x0,%eax
  401521: eb b1                   jmp    4014d4 <phase_3+0x64>
```

对 phase_3 函数分析，phase_3 函数部分主要为 switch 语句。%eax<=7，故在地址 40149e 的间接跳转，根据%eax 的值分别跳转到*0x4014e6, *0x4014ac, *0x4031c0, *0x4031c8, *0x4031d0, 0x4031d8, 0x4031e0, 0x4031e8，通过 gdb 查看地址为 0x4014ed, 0x4014f4, 0x4014fb, 0x401502, 0x401509, 0x401510。依次对对应地址汇编代码分析，得出源码：

```
void phase_3()
{
    switch(val)
    {
        case0:
            a=0x1d9;
            a-=0x13c;
            a+=0x95;
            a-=0x1d0;    //a=-158
            break;
        case1:
            a=0;
            a-=0x13c;
            a+=0x95;
            a-=0x1d0;    //a=-631
            break;
        case2:
            a=0;
            a+=0x95;
            a-=0x1d0;    //a=-315
            break;
        case3:
```

```
            a=0;
            a-=0x1d0;    //a=-464
            break;
        case4:
            a=0;      //a=0
            break;
        case5:
            a=0;
            a-=0x1d0;    //a=-464
            break;
        case6:
            a=0;      //a=0
            break;
        case7:
            a=0;
            a-=0x1d0;    //a=-464
            break;
    }
    if(val>5)
        explode_bomb();
}
```

```
That's number 2.  Keep going!
0 -158
Halfway there!
```

```
That's number 2.  Keep going!
1 -631
Halfway there!
```

```
That's number 2.  Keep going!
2 -315
Halfway there!
```

```
That's number 2.  Keep going!
3 -464
Halfway there!
```

```
That's number 2.  Keep going!
4 0
Halfway there!
```

```
That's number 2.  Keep going!
5 -464
Halfway there!
```

3.4 阶段 4 的破解与分析

密码如下：24 2 或 36 3 或 48 4

破解过程：

```
000000000040156e <phase_4>:
  40156e: 55                    push   %rbp
  40156f: 48 89 e5              mov    %rsp,%rbp
  401572: 48 83 ec 10           sub    $0x10,%rsp
  401576: 48 8d 4d fc           lea    -0x4(%rbp),%rcx
  40157a: 48 8d 55 f8           lea    -0x8(%rbp),%rdx
  40157e: be 0f 33 40 00        mov    $0x40330f,%esi
  401583: b8 00 00 00 00        mov    $0x0,%eax
  401588: e8 83 fb ff ff        callq  401110 <__isoc99_sscanf@plt>
  40158d: 83 f8 02              cmp    $0x2,%eax
  401590: 75 0d                 jne    40159f <phase_4+0x31>
  401592: 8b 45 fc              mov    -0x4(%rbp),%eax
  401595: 83 f8 01              cmp    $0x1,%eax
  401598: 7e 05                 jle    40159f <phase_4+0x31>
  40159a: 83 f8 04              cmp    $0x4,%eax
  40159d: 7e 05                 jle    4015a4 <phase_4+0x36>
  40159f: e8 91 03 00 00        callq  401935 <explode_bomb>
  4015a4: 8b 75 fc              mov    -0x4(%rbp),%esi
  4015a7: bf 05 00 00 00        mov    $0x5,%edi
  4015ac: e8 72 ff ff ff        callq  401523 <func4>
  4015b1: 39 45 f8              cmp    %eax,-0x8(%rbp)
  4015b4: 75 02                 jne    4015b8 <phase_4+0x4a>
  4015b6: c9                    leaveq
  4015b7: c3                    retq
  4015b8: e8 78 03 00 00        callq  401935 <explode_bomb>
  4015bd: eb f7                 jmp    4015b6 <phase_4+0x48>
```

对 phase_4 的函数分析，首先对__isoc99_sscanf@plt 函数传参，通过 gdb 工具，

查看 0x40330f 处字符串，得到%esi 中控制的输入格式为%d %d。

```
That's number 2.  Keep going!
0 -158
Halfway there!
1 1

Breakpoint 1, phase_4 (input=0x40
124     phases.c: No such file or
(gdb) x/s 0x40330f
0x40330f:       "%d %d"
(gdb)
```

由于指令 lea -0x4(%rbp),%rcx; lea -0x8(%rbp),%rdx，可以得到，输入控制符中第一个数为-0x8(%rbp)，第二个数为-0x4(%rbp)。代码中存在 cmp 与 jle 的混合使用，判断输入的第二个数，若第二个数大于 4 或小于等于 1，则调用 explode_bomb()函数，炸弹爆炸；若大于 1 小于等于 4，则调用 func4()函数，在调用 func4()函数前，第一个参数%rdi = 5; %rsi = -0x4(%rbp).

对 func4()函数分析：

该函数存在两个参数%rdi 与%rsi，根据 phase_4()函数，可得，第一个参数为 5，第二个参数为输入进的第二个数。分析可得源码大致如下：

```c
#include <stdio.h>
int fun4(int v1,int v2)
{
    int a,b,c,d;
    if(v1<=0)
        return 0;
    else if(v1==1)
        return v2;
    else
    {
        return fun4( v1: v1-1,v2)+fun4( v1: v1-2,v2)+v2;
    }
}
int main() {
    for(int i=2;i<=4;i++)
        printf( format: "%d %d\n",fun4( v1: 5,i),i);
    return 0;
}
```

运行可得：

```
/mnt/d/Co
24 2
36 3
48 4
```

故答案为 24 2 或 36 3 或 48 4.

3.5 阶段 5 的破解与分析

密码如下：IOAPUW(答案不唯一)

破解过程：

对 phase_5 函数分析：

```
callq  401825 <string_length>
cmp    $0x6,%eax
jne    4015f9 <phase_5+0x3a>
```

0 函数调用 string_length()函数(在阶段 1 中已详细分析)，若字符串长度不等于 6，则调用 explode_bomb()函数，炸弹爆炸，故字符串长度为 6。

```
0x00000000004015df <+32>:    movslq %eax,%rcx
0x00000000004015e2 <+35>:    movzbl (%rbx,%rcx,1),%edx
0x00000000004015e6 <+39>:    and    $0xf,%edx
0x00000000004015e9 <+42>:    movzbl 0x4031f0(%rdx),%edx
0x00000000004015f0 <+49>:    mov    %dl,-0x17(%rbp,%rcx,1)
0x00000000004015f4 <+53>:    add    $0x1,%eax
```

```
(gdb) x/s 0x4031f0
0x4031f0 <array.3401>:  "maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?"
```

```
0x0000000000401600 <+65>:    movb   $0x0,-0x11(%rbp)
0x0000000000401604 <+69>:    mov    $0x4031a6,%esi
0x0000000000401609 <+74>:    lea    -0x17(%rbp),%rdi
0x000000000040160d <+78>:    callq  0x401839 <strings_not_equal>
```

```
(gdb) x/s 0x4031a6
0x4031a6:       "flames"
```

对%rcx 依次赋值 0-5，由于%rbx 是输入的字符串的地址，通过%rcx 的赋值，实现对输入的字符串进行每一位的操作。将输入的每一位字符与 0xf 做与操作，并将其存入%edx 中，在通过%edx 对 0x4031f0 处的字符串寻址，0x4031f0 处的字符串为“maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?”，而

在接下来的函数中，将对每一位输入的字符对 0x4031f0 处的字符串寻址后，得到的新字符串与 0x4031a6 处的字符串相比。通过 gdb 工具查看 0x4031a6 处的字符串可得“flames”字符串，故相对 0x4031f0 的偏移量依次为 9、15、1、0、5、7。根据此偏移量，可以计算应输入的字符串的 ASCII 值，计算公式为偏移量+n*16。偏移量取不同值或每个字符 n 都可以取不同值，因此关卡答案较多。答案取 n=4，此时答案为 IOA@EG；取 n=5，答案为 Y_QPUW；部分取 n=4，部分取 n=5 时，答案为 IOAPUW。本题答案众多，仅需按照所示公式计算即可。

```
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.
IOAPUW
Good work!  On to the next...
```

```
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.
IOA@EG
Good work!  On to the next...
```

```
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.
Y_QPUW
Good work!  On to the next...
```

3.6 阶段 6 的破解与分析

密码如下：4 3 2 6 1 5

破解过程：

```
0000000000401624 <phase_6>:
  401624: 55                      push   %rbp
  401625: 48 89 e5                mov    %rsp,%rbp
  401628: 41 55                   push   %r13
  40162a: 41 54                   push   %r12
  40162c: 53                      push   %rbx
  40162d: 48 83 ec 58             sub    $0x58,%rsp
  401631: 48 8d 75 c0             lea    -0x40(%rbp),%rsi
  401635: e8 1d 03 00 00          callq  401957 <read_six_numbers>
  40163a: 41 bc 00 00 00 00       mov    $0x0,%r12d
  401640: eb 29                   jmp    40166b <phase_6+0x47>
  401642: e8 ee 02 00 00          callq  401935 <explode_bomb>
  401647: eb 37                   jmp    401680 <phase_6+0x5c>
  401649: 83 c3 01                add    $0x1,%ebx
  40164c: 83 fb 05                cmp    $0x5,%ebx
  40164f: 7f 17                   jg     401668 <phase_6+0x44>
  401651: 49 63 c4                movslq %r12d,%rax
  401654: 48 63 d3                movslq %ebx,%rdx
  401657: 8b 7c 95 c0             mov    -0x40(%rbp,%rdx,4),%edi
  40165b: 39 7c 85 c0             cmp    %edi,-0x40(%rbp,%rax,4)
  40165f: 75 e8                   jne    401649 <phase_6+0x25>
  401661: e8 cf 02 00 00          callq  401935 <explode_bomb>
  401666: eb e1                   jmp    401649 <phase_6+0x25>
  401668: 45 89 ec                mov    %r13d,%r12d
  40166b: 41 83 fc 05             cmp    $0x5,%r12d
  40166f: 7f 19                   jg     40168a <phase_6+0x66>
  401671: 49 63 c4                movslq %r12d,%rax
  401674: 8b 44 85 c0             mov    -0x40(%rbp,%rax,4),%eax
  401678: 83 e8 01                sub    $0x1,%eax
  40167b: 83 f8 05                cmp    $0x5,%eax
  40167e: 77 c2                   ja     401642 <phase_6+0x1e>
  401680: 45 8d 6c 24 01          lea    0x1(%r12),%r13d
  401685: 44 89 eb                mov    %r13d,%ebx
  401688: eb c2                   jmp    40164c <phase_6+0x28>
```

对 phase_6 函数分析：

函数通过调用 read_six_numbers()函数，实现六个数字的输入，假设 int_num[6]为输入的六个数字。函数将%r12d 初始化为 0，并与 5 相比，形成循环。函数将%eax 赋值为 int_num 数组中每个数时，进行减 1 操作，并与 5 做比较，若%eax 大于等于 5，则炸弹爆炸，故数组中的每个数都小于 6。函数还检查输入的六个数是否相等，若六个数字中，存在相等的情况，则炸弹爆炸。

```
Contents of section .data:
 4052d0 eb010000 01000000 e0524000 00000000  .........R@.....
 4052e0 1e020000 02000000 f0524000 00000000  .........R@.....
 4052f0 ce020000 03000000 00534000 00000000  .........S@.....
 405300 ba030000 04000000 10534000 00000000  .........S@.....
 405310 ae010000 05000000 20534000 00000000  ........ S@.....
 405320 06020000 06000000 00000000 00000000  ................
 405330 9b010000 00000000 00000000 00000000  ................
 405340 69334000 00000000 83334000 00000000  i3@......3@.....
 405350 9d334000 00000000 00000000 00000000  .3@.............
 405360 00000000 00000000 00000000 00000000  ................
```

通过反汇编查看 0x4052d0 的数据，为链表类型数据。

struct link

{

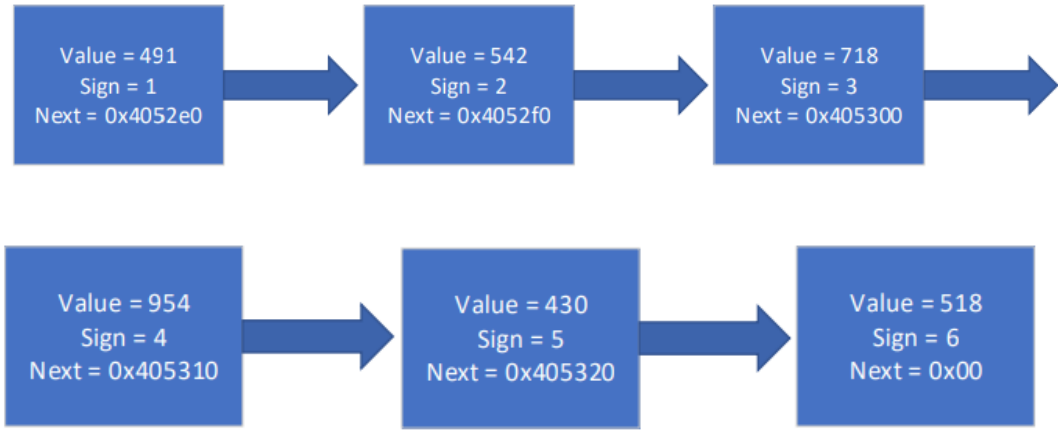
int value;

int sign;

struct link* next;

}

初始链接状态为：



```
40168a: be 00 00 00 00        mov    $0x0,%esi
40168f: eb 08                 jmp    401699 <phase_6+0x75>
401691: 48 89 54 cd 90        mov    %rdx,-0x70(%rbp,%rcx,8)
401696: 83 c6 01              add    $0x1,%esi
401699: 83 fe 05              cmp    $0x5,%esi
40169c: 7f 1c                 jg     4016ba <phase_6+0x96>
40169e: b8 01 00 00 00        mov    $0x1,%eax
4016a3: ba d0 52 40 00        mov    $0x4052d0,%edx
4016a8: 48 63 ce              movslq %esi,%rcx
4016ab: 39 44 8d c0           cmp    %eax,-0x40(%rbp,%rcx,4)
4016af: 7e e0                 jle    401691 <phase_6+0x6d>
4016b1: 48 8b 52 08           mov    0x8(%rdx),%rdx
4016b5: 83 c0 01              add    $0x1,%eax
4016b8: eb ee                 jmp    4016a8 <phase_6+0x84>
4016ba: 48 8b 5d 90           mov    -0x70(%rbp),%rbx
4016be: 48 89 d9              mov    %rbx,%rcx
4016c1: b8 01 00 00 00        mov    $0x1,%eax
4016c6: eb 12                 jmp    4016da <phase_6+0xb6>
4016c8: 48 63 d0              movslq %eax,%rdx
4016cb: 48 8b 54 d5 90        mov    -0x70(%rbp,%rdx,8),%rdx
4016d0: 48 89 51 08           mov    %rdx,0x8(%rcx)
4016d4: 83 c0 01              add    $0x1,%eax
4016d7: 48 89 d1              mov    %rdx,%rcx
4016da: 83 f8 05              cmp    $0x5,%eax
4016dd: 7e e9                 jle    4016c8 <phase_6+0xa4>
4016df: 48 c7 41 08 00 00 00  movq   $0x0,0x8(%rcx)
4016e6: 00
4016e7: 41 bc 00 00 00 00     mov    $0x0,%r12d
4016ed: eb 08                 jmp    4016f7 <phase_6+0xd3>
4016ef: 48 8b 5b 08           mov    0x8(%rbx),%rbx
4016f3: 41 83 c4 01           add    $0x1,%r12d
4016f7: 41 83 fc 04           cmp    $0x4,%r12d
4016fb: 7f 11                 jg     40170e <phase_6+0xea>
4016fd: 48 8b 43 08           mov    0x8(%rbx),%rax
401701: 8b 00                 mov    (%rax),%eax
401703: 39 03                 cmp    %eax,(%rbx)
401705: 7d e8                 jge    4016ef <phase_6+0xcb>
401707: e8 29 02 00 00        callq  401935 <explode_bomb>
40170c: eb e1                 jmp    4016ef <phase_6+0xcb>
40170e: 48 83 c4 58           add    $0x58,%rsp
401712: 5b                    pop    %rbx
401713: 41 5c                 pop    %r12
401715: 41 5d                 pop    %r13
401717: 5d                    pop    %rbp
401718: c3                    retq
```

phase_6 函数按照所输入的数字的大小，进入第 int_num[i]-1 个节点，并读取 next，例如：第一个数为 4 时，则进入 sign=3 的节点，并读取 next，即 0x405300，保存到栈中。

在读取完毕后，栈中保存的地址如表格所示：

0x405310
0x4052d0
0x405320
0x4052e0
0x4052f0
0x405300

函数 phase_6 对已经存放好的栈中的地址链接。

```
(gdb) p/x $rdx
$4 = 0x4052f0
(gdb) p/x $rcx
$6 = 0x405300
```

```
(gdb) p/x $rdx
$7 = 0x4052e0
(gdb) p/x $rcx
$8 = 0x4052f0
```

```
(gdb) p/x $rdx
$10 = 0x405320
(gdb) p/x $rcx
$11 = 0x4052e0
```

```
(gdb) p/x $rdx
$13 = 0x4052d0
(gdb) p/x $rcx
$14 = 0x405320
```

```
(gdb) p/x $rdx
$15 = 0x405310
(gdb) p/x $rcx
$16 = 0x4052d0
```

根据汇编代码，依次使 M[%rcx+8] = %rdx，使各个节点相互连接，根据输入的数字进行重新排序。

接下来 phase_6 函数对新排列好的链表进行检查，依次检查前一个节点的值是否大于等于后一个节点的值，如果不是，则调用 explode_bomb，炸弹爆炸。若检查完链表符合降序，则拆解成功。

因此对应的六个数字应该是 4 3 2 6 1 5.

答案为 4 3 2 6 1 5

3.7 阶段 7 的破解与分析(隐藏阶段)

密码如下：50

破解过程：

```
0000000000401abe <phase_defused>:
  401abe: 83 3d a7 3c 00 00 06  cmpl   $0x6,0x3ca7(%rip)        # 40576c <num_input_strings>
  401ac5: 74 01                 je     401ac8 <phase_defused+0xa>
  401ac7: c3                    retq
  401ac8: 55                    push   %rbp
  401ac9: 48 89 e5              mov    %rsp,%rbp
  401acc: 48 83 ec 60           sub    $0x60,%rsp
  401ad0: 4c 8d 45 b0           lea    -0x50(%rbp),%r8
  401ad4: 48 8d 4d a8           lea    -0x58(%rbp),%rcx
  401ad8: 48 8d 55 ac           lea    -0x54(%rbp),%rdx
  401adc: be 59 33 40 00        mov    $0x403359,%esi
  401ae1: bf 70 58 40 00        mov    $0x405870,%edi
  401ae6: b8 00 00 00 00        mov    $0x0,%eax
  401aeb: e8 20 f6 ff ff        callq  401110 <__isoc99_sscanf@plt>
  401af0: 83 f8 03              cmp    $0x3,%eax
  401af3: 74 0c                 je     401b01 <phase_defused+0x43>
  401af5: bf 98 32 40 00        mov    $0x403298,%edi
  401afa: e8 61 f5 ff ff        callq  401060 <puts@plt>
  401aff: c9                    leaveq
  401b00: c3                    retq
  401b01: be 62 33 40 00        mov    $0x403362,%esi
  401b06: 48 8d 7d b0           lea    -0x50(%rbp),%rdi
  401b0a: e8 2a fd ff ff        callq  401839 <strings_not_equal>
  401b0f: 85 c0                 test   %eax,%eax
  401b11: 75 e2                 jne    401af5 <phase_defused+0x37>
  401b13: bf 38 32 40 00        mov    $0x403238,%edi
  401b18: e8 43 f5 ff ff        callq  401060 <puts@plt>
  401b1d: bf 60 32 40 00        mov    $0x403260,%edi
  401b22: e8 39 f5 ff ff        callq  401060 <puts@plt>
  401b27: b8 00 00 00 00        mov    $0x0,%eax
  401b2c: e8 22 fc ff ff        callq  401753 <secret_phase>
  401b31: eb c2                 jmp    401af5 <phase_defused+0x37>
```

```
(gdb) x/s 0x403362
0x403362:       "DrEvil"
```

通过对 phase_defused 函数解读，可以发现，该函数调用了 secret_phase 函数，观察到将 0x403362 传给%esi，并调用 string_not_equal 函数检查是否相等，若不相等，则跳过该隐藏关卡，若相等，则调用隐藏关卡函数。使用 gbd 工具查看位于 0x403362 地址处的字符串，为 DrEvil，即为隐藏关卡打开的字符串，在 phase_4 后添加该字符串后，出现隐藏关卡。

```
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...
```

```
0000000000401753 <secret_phase>:
  401753: 55                    push   %rbp
  401754: 48 89 e5              mov    %rsp,%rbp
  401757: 53                    push   %rbx
  401758: 48 83 ec 08           sub    $0x8,%rsp
  40175c: e8 32 02 00 00        callq  401993 <read_line>
  401761: 48 89 c7              mov    %rax,%rdi
  401764: e8 d7 f9 ff ff        callq  401140 <atoi@plt>
  401769: 89 c3                 mov    %eax,%ebx
  40176b: 8d 40 ff              lea    -0x1(%rax),%eax
  40176e: 3d e8 03 00 00        cmp    $0x3e8,%eax
  401773: 77 27                 ja     40179c <secret_phase+0x49>
  401775: 89 de                 mov    %ebx,%esi
  401777: bf f0 50 40 00        mov    $0x4050f0,%edi
  40177c: e8 98 ff ff ff        callq  401719 <fun7>
  401781: 83 f8 01              cmp    $0x1,%eax
  401784: 75 1d                 jne    4017a3 <secret_phase+0x50>
  401786: bf 80 31 40 00        mov    $0x403180,%edi
  40178b: e8 d0 f8 ff ff        callq  401060 <puts@plt>
  401790: e8 29 03 00 00        callq  401abe <phase_defused>
  401795: 48 83 c4 08           add    $0x8,%rsp
  401799: 5b                    pop    %rbx
  40179a: 5d                    pop    %rbp
  40179b: c3                    retq
  40179c: e8 94 01 00 00        callq  401935 <explode_bomb>
  4017a1: eb d2                 jmp    401775 <secret_phase+0x22>
  4017a3: e8 8d 01 00 00        callq  401935 <explode_bomb>
  4017a8: eb dc                 jmp    401786 <secret_phase+0x33>
```

分析 secret_phase 函数，注意到在调用 fun7 函数时，此时%edi=0x4050f0，%esi 为输入的值，且该值小于 0x3e8，否则调用 explode_bomb 函数，炸弹爆炸。在调用 fun7 函数后，若返回值不为 1，则炸弹爆炸，故 fun7 函数返回值为 1。

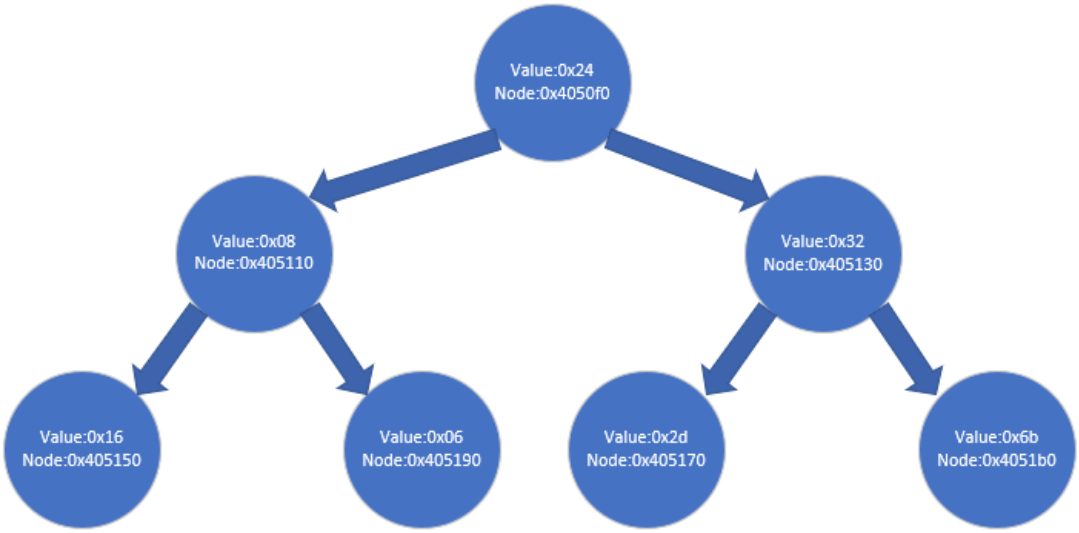
```
0000000000401719 <fun7>:
  401719: 48 85 ff              test   %rdi,%rdi
  40171c: 74 2f                 je     40174d <fun7+0x34>
  40171e: 55                    push   %rbp
  40171f: 48 89 e5              mov    %rsp,%rbp
  401722: 8b 07                 mov    (%rdi),%eax
  401724: 39 f0                 cmp    %esi,%eax
  401726: 7f 09                 jg     401731 <fun7+0x18>
  401728: 75 14                 jne    40173e <fun7+0x25>
  40172a: b8 00 00 00 00        mov    $0x0,%eax
  40172f: 5d                    pop    %rbp
  401730: c3                    retq
  401731: 48 8b 7f 08           mov    0x8(%rdi),%rdi
  401735: e8 df ff ff ff        callq  401719 <fun7>
  40173a: 01 c0                 add    %eax,%eax
  40173c: eb f1                 jmp    40172f <fun7+0x16>
  40173e: 48 8b 7f 10           mov    0x10(%rdi),%rdi
  401742: e8 d2 ff ff ff        callq  401719 <fun7>
  401747: 8d 44 00 01           lea    0x1(%rax,%rax,1),%eax
  40174b: eb e2                 jmp    40172f <fun7+0x16>
  40174d: b8 ff ff ff ff        mov    $0xffffffff,%eax
  401752: c3                    retq
```

对函数 fun7 分析并得出代码大致如下：

```
fun7(x, y)
{
    if(x==0)
        return 0xffffffff;
    if(M[x]>y)
        return 2*fun7(M[%rdi+0x8],y)
    else if (M[x] != y)
        return 2*fun7(M[%rdi+0x10],y)+1;
    else
        return 0;
}
```

```
Contents of section .data:
 4050f0 24000000 00000000 10514000 00000000  $........Q@.....
 405100 30514000 00000000 00000000 00000000  0Q@.............
 405110 08000000 00000000 90514000 00000000  .........Q@.....
 405120 50514000 00000000 00000000 00000000  PQ@.............
 405130 32000000 00000000 70514000 00000000  2.......pQ@.....
 405140 b0514000 00000000 00000000 00000000  .Q@.............
 405150 16000000 00000000 70524000 00000000  ........pR@.....
 405160 30524000 00000000 00000000 00000000  0R@.............
 405170 2d000000 00000000 d0514000 00000000  -........Q@.....
 405180 90524000 00000000 00000000 00000000  .R@.............
 405190 06000000 00000000 f0514000 00000000  .........Q@.....
 4051a0 50524000 00000000 00000000 00000000  PR@.............
 4051b0 6b000000 00000000 10524000 00000000  k........R@.....
 4051c0 b0524000 00000000 00000000 00000000  .R@.............
 4051d0 28000000 00000000 00000000 00000000  (...............
 4051e0 00000000 00000000 00000000 00000000  ................
 4051f0 01000000 00000000 00000000 00000000  ................
 405200 00000000 00000000 00000000 00000000  ................
```

对 0x4050f0 处的数据分析，该处数据为二叉树型数据结构。



通过计算可得，当输入为 50 时，fun7 函数最终返回值为 1，符合条件。即答案为 50.

第 4 章 总结

4.1 请总结本次实验的收获

通过本次实验，我掌握了计算机系统的 ISA 指令系统与寻址方式，同时能够熟练掌握 linux 下调试器的反汇编调试跟踪分析机器语言的方法，对 GDB 调试的功能更加得心应手，同时更深入的了解到汇编语言对应的高级程序语言，对函数的过程，栈的使用更加清晰。

4.2 请给出对本次实验内容的建议

希望 ppt 内容更加翔实，以便更好理解实验。

注：本章为酌情加分项。

参考文献

[1] RANDALE.BRYANT, DAVIDR.O‘HALLARON. 深入理解计算机系统[M]. 机械工业出版社, 2011.