

华中科技大学

# 课程实验报告

课程名称： 计算机系统基础

实验名称： 机器级语言理解

院 系： 计算机科学与技术

专业班级： CS2304

学 号： U202315653

姓 名： 郝依凝

指导教师： 金良海

2025 年 4 月 7 日

## 一、实验目的与要求

通过逆向分析一个二进制程序（称为“二进制炸弹”）的构成和运行逻辑，加深对理论课中关于程序的机器级表示各方面知识点的理解，增强反汇编、跟踪、分析、调试等能力。

实验环境：Ubuntu, GCC, GDB 等

## 二、实验内容

### 任务 1 二进制炸弹拆除

作为实验目标的二进制炸弹(binary bombs)可执行程序由多个“关”组成。每一个“关”(阶段)要求输入一个特定字符串,如果输入满足程序代码的要求,该阶段即通过,否则程序输出失败。实验的目标是设法得到得出解除尽可能多阶段的字符串。

为了完成二进制炸弹的拆除任务,需要通过反汇编和分析跟踪程序每一阶段的机器代码,从中定位和理解程序的主要执行逻辑,包括关键指令、控制结构和相关数据变量等等,进而推断拆除炸弹所需要的目标字符串。

实验源程序及相关文件 bomb.rar

bomb.c 主程序

phases.o 各个阶段的目标程序

support.c 完成辅助功能的目标程序

support.h 公共头文件

#### 阶段 1: 串比较 phase\_1(char \*input);

要求输出的字符串(input)与程序中内置的某一特定字符串相同。提示:找到与input串相比较的特定串的地址,查看相应单元中的内容,从而确定input应输入的串。

#### 阶段 2: 循环 phase\_2(char \*input);

要求在一行上输入6个整数数据,与程序自动产生的6个数据进行比较,若一致,则过关。提示:将输入串input拆分成6个数据由函数read\_six\_numbers(input, numbers)完成。之后是各个数据与自动产生的数据的比较,在比较中使用了循环语句。

#### 阶段 3: 条件分支 phase\_3(char \*input);

要求输入两个整数数据,与程序中给定的数据比较,相等则过关。提示:在自动生成数据时,使用了switch...case语句。

### 三、实验记录及问题回答

#### (1) 实验任务 1 的实验记录

```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL
(base) gabriel@LAPTOP-CT8H0900:/mnt/d/计算机系统基础实验/计基3$ gcc -g -o bomb -D U3 bomb.c support.c phases.o
(base) gabriel@LAPTOP-CT8H0900:/mnt/d/计算机系统基础实验/计基3$ ./bomb
Input your Student ID :
U202315653
welcome U202315653
You have 6 bombs to defuse!
Gate 1 : input a string that meets the requirements.
Computer System Foundation.
Phase 1 passed!
Gate 2 : input six integers that meets the requirements.
3 5 8 13 21 34
Phase 2 passed!
Gate 3 : input 2 integers.
6 425
Phase 3 passed!
    
```

图 1 测试结果

1. 阶段 1  
Computer System Foundation.
2. 阶段 2  
3 5 8 13 21 34
3. 阶段 3  
6 425

#### (2) 拆除炸弹的过程中关键操作

1. 阶段 1  
在函数 phase\_1 处设置断点, 观察到将地址 0x5555555580b6 地址 0x555555558510 传入函数 strings\_not\_equal

```

Register group: general
rax    0x555555558510    93824992249104    rbx    0x0            0
rcx    0x1b            27                rdx    0x5555555580b6    93824992247990
rsi    0x5555555580b6    93824992247990    rdi    0x555555558510    93824992249104
rbp    0x7fffffffda0    0x7fffffffda0    rsp    0x7fffffffda0    0x7fffffffda0
r8     0x0            0                r9     0x0            0
r10    0x0            0                r11    0x246         582
r12    0x7fffffffdc18    140737488346136   r13    0x555555555396    93824992236438
r14    0x555555557d70    93824992247152    r15    0x7ffff7ffd040    140737354125376
rip    0x5555555599ee    0x5555555599ee    eflags 0x202         [ IF ]
cs     0x33            51                ss     0x2b         43
ds     0x0            0                es     0x0            0
fs     0x0            0                gs     0x0            0

0x5555555599bf <phase_1+33> mov    %eax,-0x4(%rbp)
0x5555555599c2 <phase_1+36> mov    -0x4(%rbp),%edx
0x5555555599c5 <phase_1+39> mov    %rdx,%rax
0x5555555599c8 <phase_1+42> shl    $0x2,%rax
0x5555555599cc <phase_1+46> add    %rdx,%rax
0x5555555599cf <phase_1+49> lea    0x0(,%rax,4),%rdx
0x5555555599d7 <phase_1+57> add    %rdx,%rax
0x5555555599da <phase_1+60> add    %rax,%rax
0x5555555599dd <phase_1+63> lea    0x263c(%rip),%rdx    # 0x5555555580b6 <special>
0x5555555599e4 <phase_1+70> add    %rax,%rdx
0x5555555599e7 <phase_1+73> mov    -0x18(%rbp),%rax
0x5555555599eb <phase_1+77> mov    %rdx,%rsi
0x5555555599ee <phase_1+80> mov    %rax,%rdi
B> 0x5555555599f1 <phase_1+83> call   0x5555555556a8 <strings_not_equal>
0x5555555599f6 <phase_1+88> test   %eax,%eax
0x5555555599f8 <phase_1+90> je     0x5555555599ff <phase_1+97>

Multi-thre Thread 0x7ffff7d8a7 In: phase_1    L?? PC: 0x5555555599ee
    
```

图 2 函数 phase\_1

进入函数 strings\_not\_equal, 观察到函数内分别计算地址 0x5555555580b6 地址 0x555555558510 字符串长度并比较, 而后循环比较每一个字符。由此可知该函数为比较输入字符串与目标字符串是否一致: 查看两地址后确定输入字符串地址 0x555555558510, 目标字符串地址 0x5555555580b6。

```

0x555555556f4 <strings_not_equal+76> mov     -0x18(%rbp),%rax
0x555555556f8 <strings_not_equal+80> movzbl (%rax),%edx
0x555555556fb <strings_not_equal+83> mov     -0x10(%rbp),%rax
0x555555556ff <strings_not_equal+87> movzbl (%rax),%eax
0x55555555702 <strings_not_equal+90> cmp     %al,%dl
0x55555555704 <strings_not_equal+92> je      0x5555555570d <strings_not_equal+101>
0x55555555706 <strings_not_equal+94> mov     $0x1,%eax
0x5555555570b <strings_not_equal+99> jmp     0x55555555727 <strings_not_equal+127>
0x5555555570d <strings_not_equal+101> addq    $0x1,-0x18(%rbp)
0x55555555712 <strings_not_equal+106> addq    $0x1,-0x10(%rbp)
0x55555555717 <strings_not_equal+111> mov     -0x18(%rbp),%rax
0x5555555571b <strings_not_equal+115> movzbl (%rax),%eax
0x5555555571e <strings_not_equal+118> test    %al,%al
0x55555555720 <strings_not_equal+120> jne     0x555555556f4 <strings_not_equal+76>
    
```

图 3 字符比较

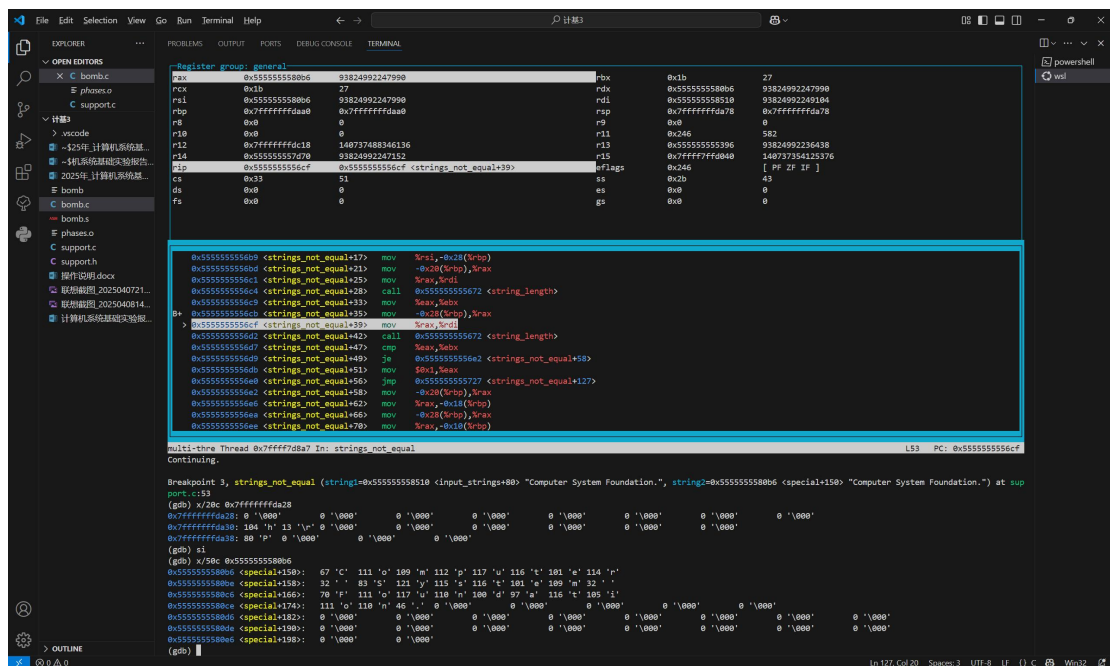


图 4 函数 strings\_not\_equal

## 2. 阶段 2

在函数 `phase_2` 处设置断点，观察到首先将输入串第一个数字存入 `eax` 寄存器，将学号第 10 位由字符转为数字后存入 `edx` 寄存器，比较二者是否相等，可推知第一个数字为学号（U202315653）第 10 位：3。

而后观察到输入串第二个数字存入 `eax` 寄存器，将学号第 9 位由字符转为数字后存入 `edx` 寄存器，比较二者是否相等，可推知第二个数字为学号（U202315653）第 9 位：5。

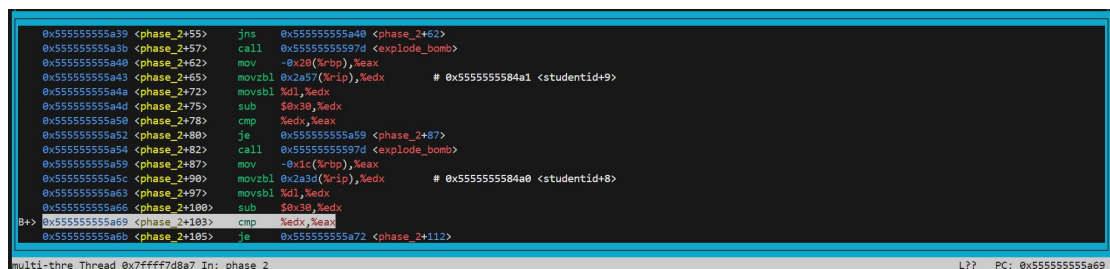


图 5 函数 phase\_2 前 2 位

剩余 4 个数字由循环产生：循环中将输入数字读取到 `edx` 寄存器中，将目标数字前 1 位数字读取到 `ecx` 寄存器中，将目标数字前 2 位数字读取到 `eax` 寄存器中，两位数字相加结果保存到 `eax` 寄存器中，得到目标数字与输入数字比较。

```

Register group: general
rax 0x8 8 rbx 0x0 0
rcx 0x5 5 rdx 0x8 8
rsi 0x22 34 rdi 0x7fffffffda00 140737488344064
rbp 0x7fffffffda00 0x7fffffffda00 rsp 0x7fffffffda00 0x7fffffffda00
r8 0x1999999999999999 1844674407370955161 r9 0x0 0
r10 0x7fffffff4bac0 140737353398976 r11 0x7fffffff4c3c0 140737353401280
r12 0x7fffffffdc18 140737488346136 r13 0x55555555396 93824992236438
r14 0x55555557d70 93824992247152 r15 0x7fffffffda00 140737354125376
rip 0x55555555a9e 0x55555555a9e <phase_2+156> eflags 0x202 [ IF ]
cs 0x33 51 ss 0x2b 43
ds 0x0 0 es 0x0 0
fs 0x0 0 gs 0x0 0

0x55555555a9e <phase_2+156> cldq
0x55555555a9c <phase_2+158> mov -0x20(%rbp,%rax,4),%ecx
0x55555555a98 <phase_2+142> mov -0x24(%rbp),%eax
0x55555555a93 <phase_2+145> sub $0x2,%eax
0x55555555a96 <phase_2+148> cldq
0x55555555a98 <phase_2+150> mov -0x20(%rbp,%rax,4),%eax
0x55555555a9c <phase_2+154> add %ecx,%eax
> 0x55555555a9e <phase_2+156> cmp %eax,%edx
0x55555555aa0 <phase_2+158> je 0x55555555aa7 <phase_2+165>
0x55555555aa2 <phase_2+160> call 0x5555555597d <explode_bomb>
0x55555555aa7 <phase_2+165> addi $0x4,-0x24(%rbp)
0x55555555aab <phase_2+169> cmpl $0x5,-0x24(%rbp)
0x55555555aaf <phase_2+173> jle 0x55555555a70 <phase_2+121>
0x55555555ab1 <phase_2+175> nop
0x55555555ab2 <phase_2+176> mov -0x8(%rbp),%rax

```

图 6 函数 phase\_2 循环部分

构成斐波那契数列：其中  $a_0$ =学号第 10 位， $a_1$ =学号第 9 位， $a_n=a_{n-1}+a_{n-2}$ 。

### 3. 阶段 3

在函数 `phase_3` 处设置断点，观察到首先将学号第 8 位由字符转为数字后存入 `edx` 寄存器，将输入第一个数字存入 `eax` 寄存器，比较二者是否相等，可推知第一个数字为学号（U02315653）第 8 位：6。

```

Register group: general
rax 0x6 6 rbx 0x0 0
rcx 0x20 32 rdx 0x6 6
rsi 0x1a9 425 rdi 0x7fffffffda00 140737488344144
rbp 0x7fffffffda00 0x7fffffffda00 rsp 0x7fffffffda00 0x7fffffffda00
r8 0x1999999999999999 1844674407370955161 r9 0x0 0
r10 0x7fffffff4bac0 140737353398976 r11 0x7fffffff4c3c0 140737353401280
r12 0x7fffffffdc18 140737488346136 r13 0x55555555396 93824992236438
r14 0x55555557d70 93824992247152 r15 0x7fffffffda00 140737354125376
rip 0x55555555b33 0x55555555b33 <phase_3+107> eflags 0x202 [ IF ]
cs 0x33 51 ss 0x2b 43
ds 0x0 0 es 0x0 0
fs 0x0 0 gs 0x0 0

0x55555555b18 <phase_3+80> cmpl $0x1,-0xc(%rbp)
0x55555555b1c <phase_3+84> jg 0x55555555b23 <phase_3+91>
0x55555555b1e <phase_3+86> call 0x5555555597d <explode_bomb>
0x55555555b23 <phase_3+91> movzbl 0x2975(%rip),%eax # 0x55555555849f <studentid+7>
0x55555555b2a <phase_3+98> movzbl %eax,%eax
0x55555555b2d <phase_3+101> lea -0x30(%rax),%edx
0x55555555b30 <phase_3+104> mov -0x18(%rbp),%eax
> 0x55555555b33 <phase_3+107> cmp %eax,%edx
0x55555555b35 <phase_3+109> je 0x55555555b3c <phase_3+116>
0x55555555b37 <phase_3+111> call 0x5555555597d <explode_bomb>
0x55555555b3c <phase_3+116> mov -0x18(%rbp),%eax
0x55555555b3f <phase_3+119> cmp $0x9,%eax
0x55555555b42 <phase_3+122> ja 0x55555555b41 <phase_3+249>
0x55555555b44 <phase_3+124> mov %eax,%eax
0x55555555b46 <phase_3+126> lea 0x0(%rax,4),%rdx

```

图 7 函数 phase\_3 第 1 位数字

```

rax 0x6 6 rbx 0x0 0
rcx 0x20 32 rdx 0x6 6
rsi 0x1a9 425 rdi 0x7fffffffda00 140737488344144
rbp 0x7fffffffda00 0x7fffffffda00 rsp 0x7fffffffda00 0x7fffffffda00
r8 0x1999999999999999 1844674407370955161 r9 0x0 0
r10 0x7fffffff4bac0 140737353398976 r11 0x7fffffff4c3c0 140737353401280
r12 0x7fffffffdc18 140737488346136 r13 0x55555555396 93824992236438
r14 0x55555557d70 93824992247152 r15 0x7fffffffda00 140737354125376
rip 0x55555555b3f 0x55555555b3f <phase_3+119> eflags 0x246 [ PF ZF IF ]
cs 0x33 51 ss 0x2b 43
ds 0x0 0 es 0x0 0
fs 0x0 0 gs 0x0 0

0x55555555b30 <phase_3+104> mov -0x18(%rbp),%eax
> 0x55555555b33 <phase_3+107> cmp %eax,%edx
0x55555555b35 <phase_3+109> je 0x55555555b3c <phase_3+116>
0x55555555b37 <phase_3+111> call 0x5555555597d <explode_bomb>
0x55555555b3c <phase_3+116> mov -0x18(%rbp),%eax
> 0x55555555b3f <phase_3+119> cmp $0x9,%eax
0x55555555b42 <phase_3+122> ja 0x55555555b41 <phase_3+249>

```

图 8 确认输入值小于 9



## 计算机系统基础实验报告

将 `rax` 寄存器值（即第一个数字）\*4 存入 `rdx` 寄存器，此时 `rdx` 寄存器值为 24，将 `0x5555 5555 6370*1+24` 地址处值取 32 位存入 `rax` 寄存器中，得到 `rax` 中值为 `0xffff f82d`。

```

Register group: general
rax 0xfffff82d 4294965293 rdx 0x0 0
rcx 0x20 32 rdx 0x18 24
rsi 0x1a9 425 rdi 0x7fffffffda50 140737488344144
rbp 0x7fffffffda0 0x7fffffffda0 rbp 0x7fffffffdaa0 0x7fffffffdaa0
r8 0x1999999999999999 1844674407370955161 r9 0x0 0
r10 0x7ffff7f4bac0 140737353388976 r11 0x7ffff7f4c3c0 140737353401280
r12 0x7ffff7f4dc18 140737488346136 r13 0x55555555396 93824992236438
r14 0x55555555d70 93824992247152 r15 0x7ffff7fda040 140737354125376
rip 0x55555555b58 0x55555555b58 <phase_3+144> eflags 0x293 [ CF AF SF IF ]
cs 0x33 51 ss 0x2b 43
ds 0x0 0 es 0x0 0
fs 0x0 0 gs 0x0 0

0x55555555b30 <phase_3+104> mov -0x18(%rbp),%eax
0x55555555b33 <phase_3+107> cmp %eax,%edx
0x55555555b35 <phase_3+109> je 0x55555555b3c <phase_3+116>
0x55555555b37 <phase_3+111> call 0x5555555597d <explode_bomb>
0x55555555b3c <phase_3+116> mov -0x18(%rbp),%eax
0x55555555b3f <phase_3+119> cmpl $0,%eax
0x55555555b42 <phase_3+122> je 0x55555555b5c1 <phase_3+249>
0x55555555b44 <phase_3+124> mov %eax,%eax
0x55555555b46 <phase_3+126> leaq 0x0(,%rax,4),%rdx
0x55555555b4e <phase_3+134> leaq 0x81b(%rip),%rax # 0x555555556370
0x55555555b55 <phase_3+141> mov (%rdx,%rax,1),%eax
0x55555555b58 <phase_3+144> cltq
0x55555555b5a <phase_3+146> leaq 0x80f(%rip),%rdx # 0x555555556370
0x55555555b61 <phase_3+153> add %rdx,%rax
0x55555555b64 <phase_3+156> notrack jmp *%rax

Multi-Thread Thread 0x7ffff7d8a7 In: phase_3
(gdb) si
0x000055555555b3f in phase_3 ()
(gdb) si
0x000055555555b42 in phase_3 ()
(gdb) si
0x000055555555b44 in phase_3 ()
(gdb) si
0x000055555555b46 in phase_3 ()
(gdb) si
0x000055555555b4e in phase_3 ()
(gdb) si
0x000055555555b55 in phase_3 ()
(gdb) x/20c 0x555555556388
0x555555556388: 45 '-' -8 '\370' -1 '\377' -1 '\377' 54 '6' -8 '\370' -1 '\377' -1 '\377'
0x555555556390: 63 '?' -8 '\370' -1 '\377' -1 '\377' 72 'H' -8 '\370' -1 '\377' -1 '\377'
0x555555556398: 98 'b' 114 'n' 117 'u' 105 'i'
  
```

图 9 0x555555556388 处值

将 `rax` 寄存器中值扩展为 64 位得到 `0xffff ffff ffff f82d`，将 `0x5555 5555 6370` 存入 `rdx` 寄存器，二者相加结果存入 `rax` 寄存器，得到地址 `0x5555 5555 5b9d`，跳转至此处。

```

Register group: general
rax 0x55555555b9d 93824992238493 rdx 0x0 0
rcx 0x20 32 rdx 0x555555556370 93824992240496
rsi 0x1a9 425 rdi 0x7fffffffda50 140737488344144
rbp 0x7fffffffda0 0x7fffffffda0 rbp 0x7fffffffdaa0 0x7fffffffdaa0
r8 0x1999999999999999 1844674407370955161 r9 0x0 0
r10 0x7ffff7f4bac0 140737353388976 r11 0x7ffff7f4c3c0 140737353401280
r12 0x7ffff7f4dc18 140737488346136 r13 0x55555555396 93824992236438
r14 0x55555555d70 93824992247152 r15 0x7ffff7fda040 140737354125376
rip 0x55555555b64 0x55555555b64 <phase_3+156> eflags 0x203 [ CF IF ]
cs 0x33 51 ss 0x2b 43
ds 0x0 0 es 0x0 0
fs 0x0 0 gs 0x0 0

0x55555555b42 <phase_3+122> ja 0x55555555b5c1 <phase_3+249>
0x55555555b44 <phase_3+124> mov %eax,%eax
0x55555555b46 <phase_3+126> leaq 0x0(,%rax,4),%rdx
0x55555555b4e <phase_3+134> leaq 0x81b(%rip),%rax # 0x555555556370
0x55555555b55 <phase_3+141> mov (%rdx,%rax,1),%eax
0x55555555b58 <phase_3+144> cltq
0x55555555b5a <phase_3+146> leaq 0x80f(%rip),%rdx # 0x555555556370
0x55555555b61 <phase_3+153> add %rdx,%rax
0x55555555b64 <phase_3+156> notrack jmp *%rax
0x55555555b67 <phase_3+159> movl $0x32f,-0x10(%rbp)
0x55555555b6e <phase_3+166> jmp 0x55555555b6c <phase_3+254>
0x55555555b70 <phase_3+168> movl $0x130,-0x10(%rbp)
0x55555555b77 <phase_3+175> jmp 0x55555555b6c <phase_3+254>
0x55555555b79 <phase_3+177> movl $0x184,-0x10(%rbp)
0x55555555b80 <phase_3+184> jmp 0x55555555b6c <phase_3+254>

Multi-Thread Thread 0x7ffff7d8a7 In: phase_3
  
```

图 10 地址跳转

地址 `0x5555 5555 5b9d` 处执行命令将 `0x1a9` 存入地址 `-0x10(%rbp)` 处。

```

0x55555555b9d <phase_3+213> movl $0x1a9,-0x10(%rbp)
0x55555555ba4 <phase_3+220> jmp 0x55555555bc6 <phase_3+254>
  
```

图 11 执行对应命令

将输入第二个数字存入 `eax` 寄存器，与 `-0x10(%rbp)` 处值比较，可推知第二个数字即为 `0x1a9`（十进制 425）。

```

--Register group: general
rax    0x1a9      425
rcx    0x20      32
rsi    0x1a9      425
rbp    0x7fffffffda0 0x7fffffffda0
r8     0x1999999999999999 1844674407370955161
r10    0x7ffff7f4bac0 140737353398976
r12    0x7ffff7f4c18 140737488346136
r14    0x55555557d70 93824992247152
rip    0x55555555bcc 0x55555555bcc <phase_3+260>
cs     0x33      51
ds     0x0       0
fs     0x0       0

rbx    0x0       0
rdx    0x555555556370 93824992240496
rdi    0x7fffffffda0 140737488344144
rsp    0x7fffffffda0 0x7fffffffda0
r9     0x0       0
r11    0x7ffff7f4c3c0 140737353401280
r13    0x555555555396 93824992236438
r15    0x7ffff7f4d040 140737354125376

eflags 0x246     [ PF ZF IF ]
ss     0x2b      43
es     0x0       0
gs     0x0       0

0x55555555bba6 <phase_3+222> movl $0x374,-0x10(%rbp)
0x55555555bad <phase_3+229> jmp 0x55555555bcc <phase_3+254>
0x55555555baef <phase_3+231> movl $0x7b,-0x10(%rbp)
0x55555555bb0c <phase_3+238> jmp 0x55555555bcc <phase_3+254>
0x55555555bb1b <phase_3+240> movl $0x141,-0x10(%rbp)
0x55555555bb2f <phase_3+247> jmp 0x55555555bcc <phase_3+254>
0x55555555bb41 <phase_3+249> call 0x5555555597d <explode_bomb>
0x55555555bb5c <phase_3+254> mov -0x14(%rbp),%eax
0x55555555bb69 <phase_3+257> cmp %eax,-0x10(%rbp)
> 0x55555555bbcc <phase_3+260> je 0x55555555bd3 <phase_3+267>
0x55555555bbce <phase_3+262> call 0x5555555597d <explode_bomb>
0x55555555bbd5 <phase_3+267> nop
0x55555555bbd4 <phase_3+266> mov -0x8(%rbp),%rax
0x55555555bbd8 <phase_3+272> xor %rax,%rax
0x55555555bbe1 <phase_3+281> je 0x55555555be8 <phase_3+288>
    
```

图 12 phase\_3 第 2 位数字

## 四、体会

通过本次实验，我深刻理解了程序在机器级的执行逻辑和控制结构的实现方式。在逆向分析二进制炸弹的过程中，我掌握了以下关键技能：

1. GDB 调试工具的应用：通过设置断点、单步执行、查看寄存器和内存内容，能够快速定位关键代码段。例如，在阶段 1 中，通过比较 `strings_not_equal` 函数的参数地址，直接提取目标字符串。
2. 反汇编代码分析：识别循环、条件分支等结构。例如，阶段 2 的斐波那契数列生成逻辑通过循环累加实现，阶段 3 的 `switch-case` 语句通过跳转表完成分支跳转。
3. 数据关联与推导：结合学号信息推导输入值。例如，阶段 2 的前两个数字直接来源于学号特定位数，阶段 3 的第一个数字由学号第 8 位决定，第二个数字通过跳转表计算得出。
4. 内存地址与指针操作：理解如何通过基址+偏移量访问数据。例如，阶段 3 中通过 `rax*4` 计算跳转表偏移，进而获取目标值。

实验过程中，我意识到耐心和细致至关重要。例如，阶段 3 的跳转表分析需要反复验证地址计算和内存值的正确性。此外，逆向工程不仅需要技术能力，还需要逻辑推理能力，将零散的机器指令串联成完整的执行流程。

本次实验让我对程序的机器级表示有了更直观的认识，提升了反汇编、调试和逆向分析的能力，为后续学习系统底层原理奠定了坚实基础。