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-CT8HO90O:/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-CT8HO90O:/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 intergers that meets the requirements.
3 5 8 13 21 34
Phase 2 passed!
Gate 3 :  input 2 intergers.
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

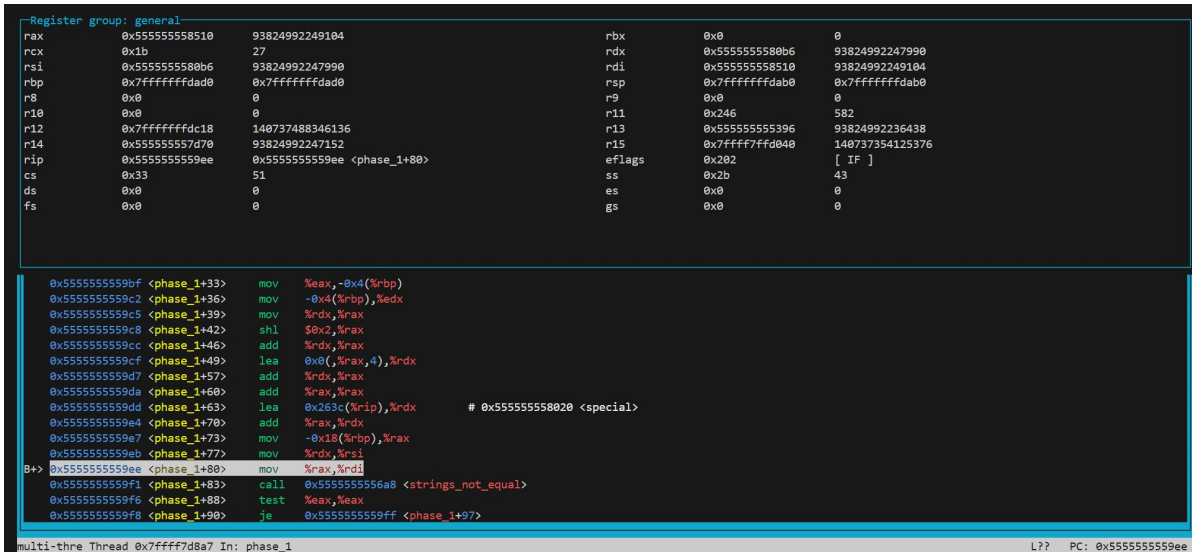

图 2 函数 phase_1

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。

```
0x5555555556f4 <strings_not_equal+76>    mov    -0x18(%rbp),%rax
0x5555555556f8 <strings_not_equal+80>    movzbl (%rax),%edx
0x5555555556fb <strings_not_equal+83>    mov    -0x10(%rbp),%rax
0x5555555556ff <strings_not_equal+87>    movzbl (%rax),%eax
0x555555555702 <strings_not_equal+90>    cmp    %al,%dl
0x555555555704 <strings_not_equal+92>    je     0x55555555570d <strings_not_equal+101>
0x555555555706 <strings_not_equal+94>    mov    $0x1,%eax
0x55555555570b <strings_not_equal+99>    jmp    0x555555555727 <strings_not_equal+127>
0x55555555570d <strings_not_equal+101>   addq   $0x1,-0x18(%rbp)
0x555555555712 <strings_not_equal+106>   addq   $0x1,-0x10(%rbp)
0x555555555717 <strings_not_equal+111>   mov    -0x18(%rbp),%rax
0x55555555571b <strings_not_equal+115>   movzbl (%rax),%eax
0x55555555571e <strings_not_equal+118>   test   %al,%al
0x555555555720 <strings_not_equal+120>   jne    0x5555555556f4 <strings_not_equal+76>
```

图 3 字符比较

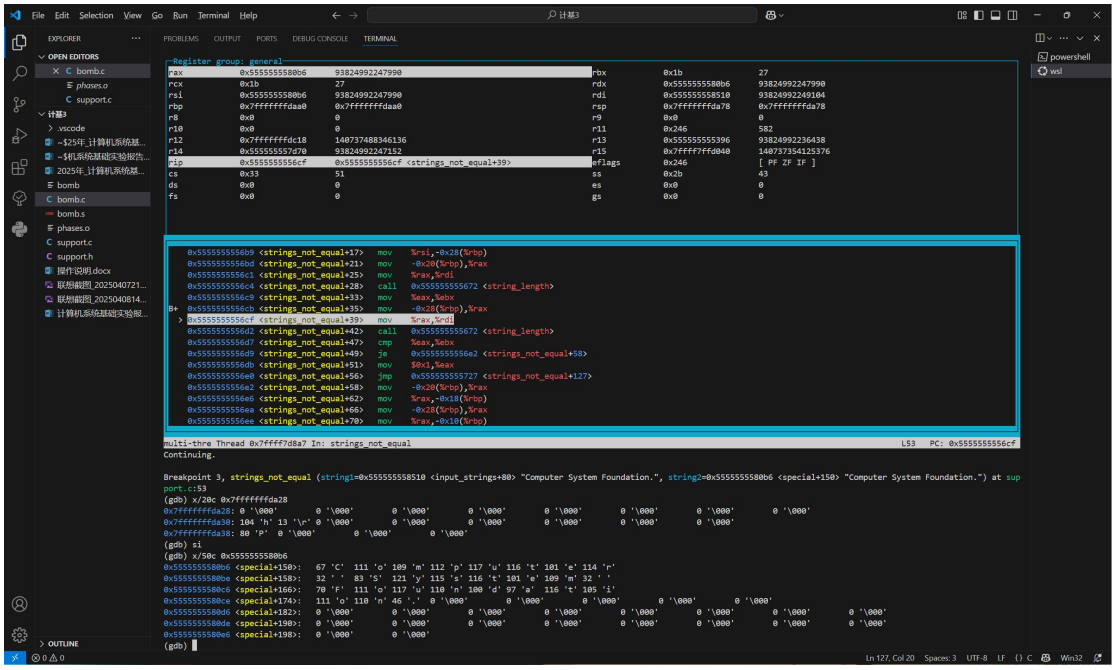

图 4 函数 strings_not_equal

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。

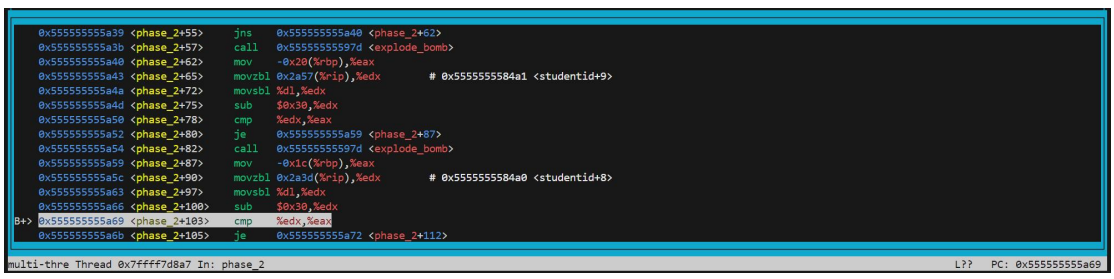

图 5 函数 phase_2 前 2 位

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。

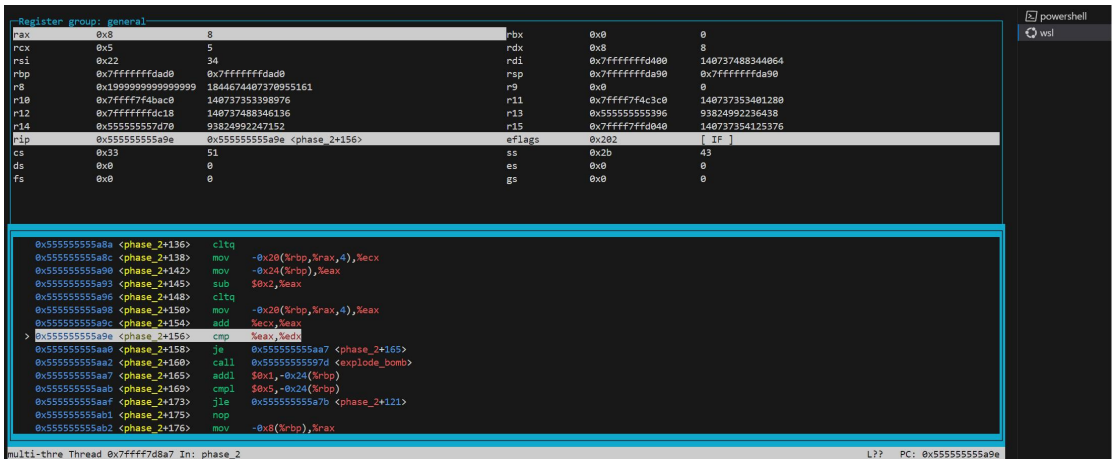

图 6 函数 phase_2 循环部分

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（U202315653）第 8 位：6。

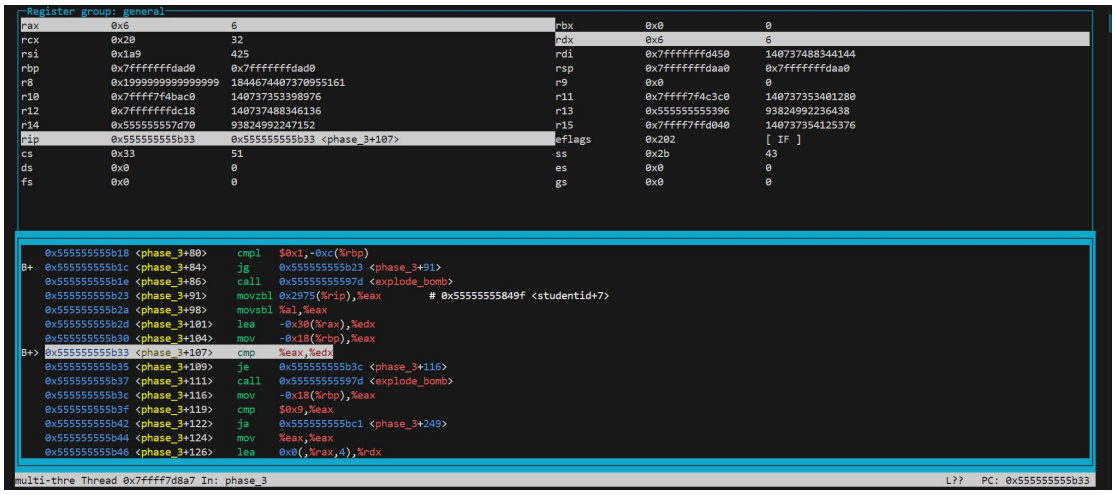

图 7 函数 phase_3 第 1 位数字

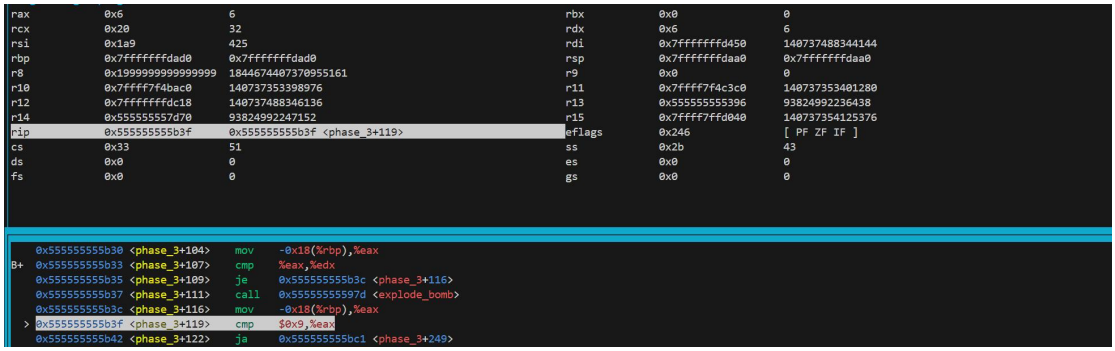

图 8 确认输入值小于 9

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。

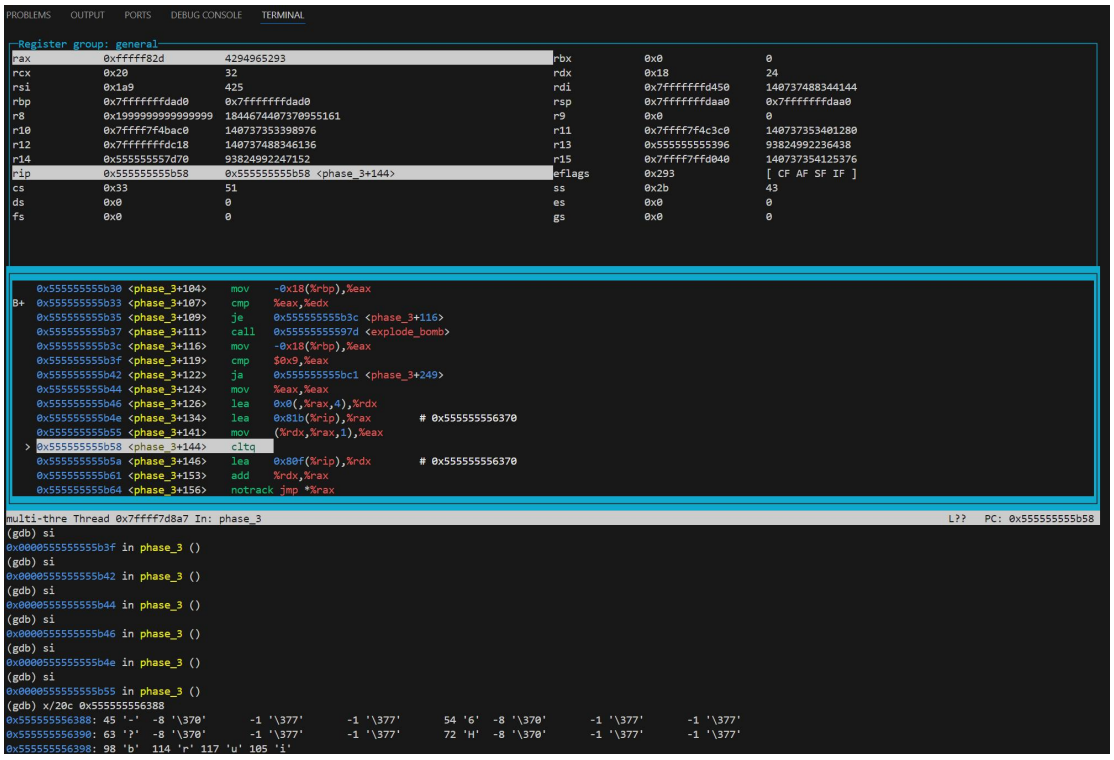

图 9 0x555555556388 处值

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。

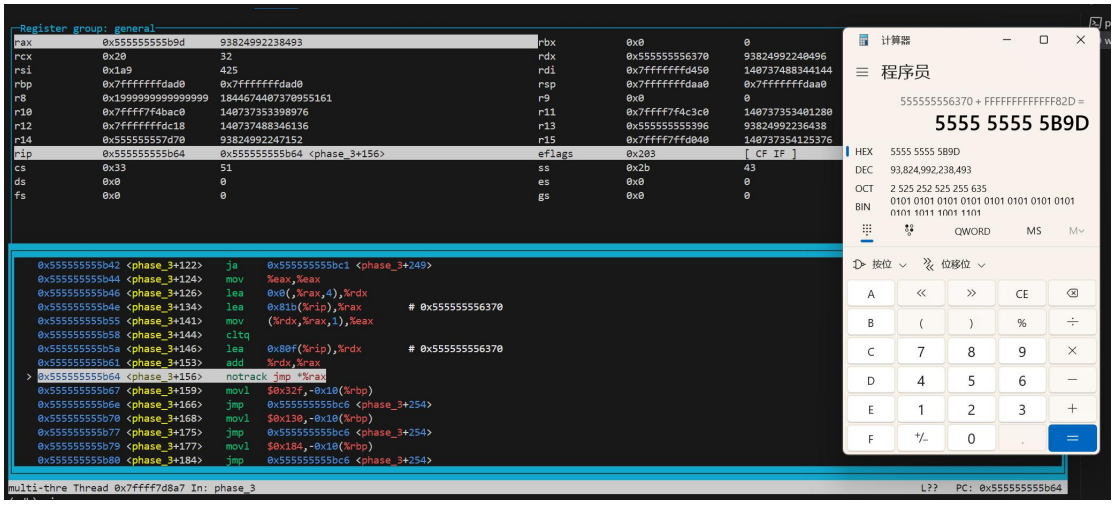

图 10 地址跳转

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

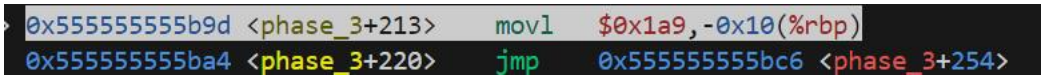

图 11 执行对应命令

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）。

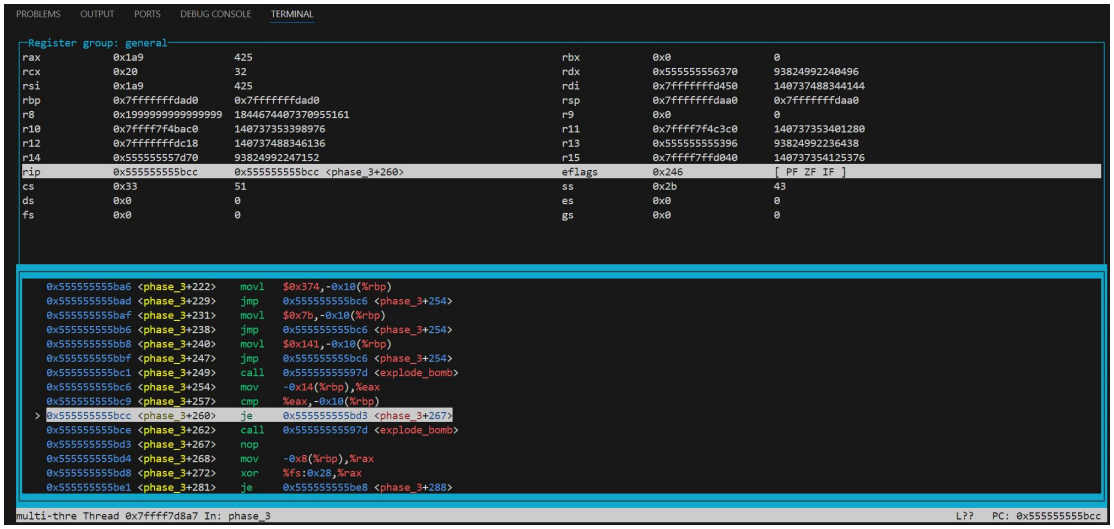

图 12 phase_3 第 2 位数字

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。