

**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**实验名称： 二进制程序分析**

**院 系 ：计算机科学与技术**

**专业班级 ： CS2204**

**学 号 ： U202215458**

**姓 名 ： 柯晓豪**

**指导教师 ： 金良海**

**2024 年 5 月 29 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

phases.h 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);**

要求输入一个整数数据，该数据与程序自动生成的 一个数据比较，相等则过关。提示：在自动生成数据时，使用了 switch … case 语句。

**阶段4：递归调用和栈 phase\_4(char \*input);**

要求在一行中输入两个数，第一个数表示在一个有序的数组（或者binary search tree）中需要搜索到的数，该数是在一定范围之内的；第二个数表示找到搜索数的路径（在树的左边搜索编码为二进制位0，在树的有边搜索编码为二进制位1）。

**阶段5：指针和数组访问 phase\_5(char \*input);**

要求在一行中输入一个串，该串与程序自动生成的串相同。在生成串和比较串时，使用了数组和指针。

**阶段6：链表、结构、指针的访问 phase\_6(char \*input);**

要求在一行中输入6个数，这6个数是一个链表中结点的顺序号（从 1 到6）。按照输入的顺序号，将对应链表结点中的值形成一个数组。若该数组是按照降序排列的，则过关。

**三、实验记录及问题回答**

**（1）实验任务1 的实验记录**

实验结果截图如图1所示：

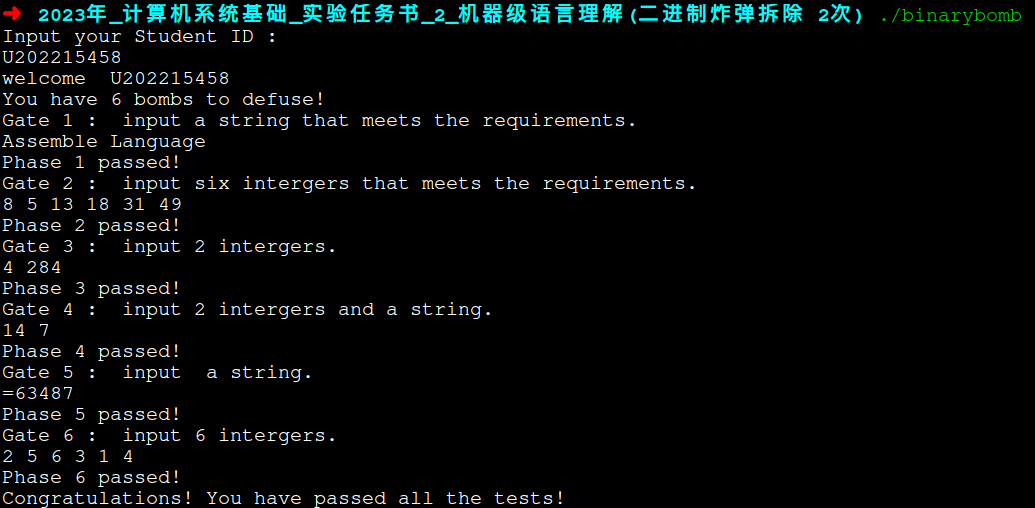


图1 实验结果截图

**（2）拆除炸弹的过程中关键操作**

初始化炸弹后我先将phase.o反汇编并重定向生成.asm文件，然后便开始拆除炸弹。

**phase\_1:**

根据汇编代码（如图2），我们可以推测出字符串存储在rdx寄存器中，之后查看rdx寄存器的内容，发现里面存储着的字符串为“Assemble Language”，如图3,输入之后成功通关。

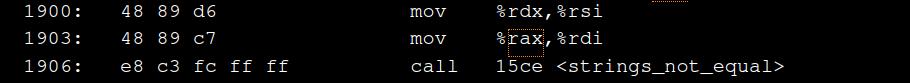


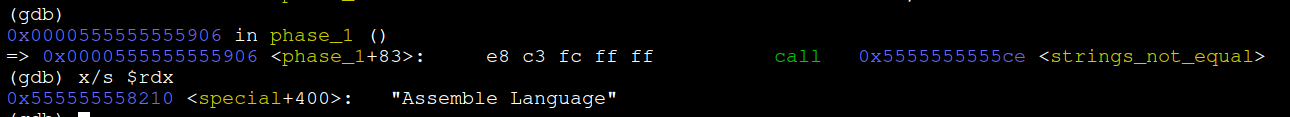
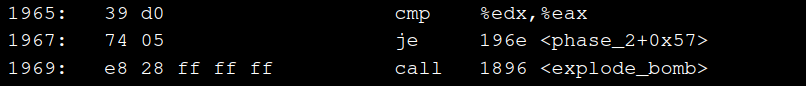
图2 phase\_1汇编代码段截图

图3 查看寄存器截图

**phase\_2:**

观察汇编代码，发现eax寄存器保存着我们输入的第一个数字，edx寄存器保存着学号的最后一位数字，并且最后判断eax和edx是否相等，因此可以推断我们输入的第一个数字应该是学号的最后一位数字8。同理可得第二个数字是学号的倒数第二位5。

图4 phase\_2汇编代码段截图（推断前两位数字）

接着我们进入一个循环结构，可以看到，循环体的计数器从2开始，也就是数组下标从2开始到数组的最后一个下标5进行循环，循环体中将当前下标的前两位数字相加，并判断是否与读入的数字相等，相等则继续循环，否则炸弹爆炸。因此可以推断出读入的数字应为8 5 13 18 31 49.

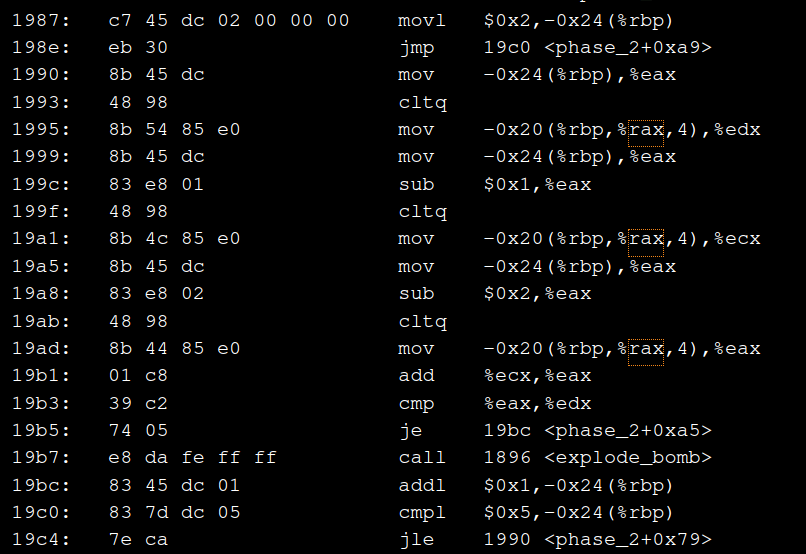
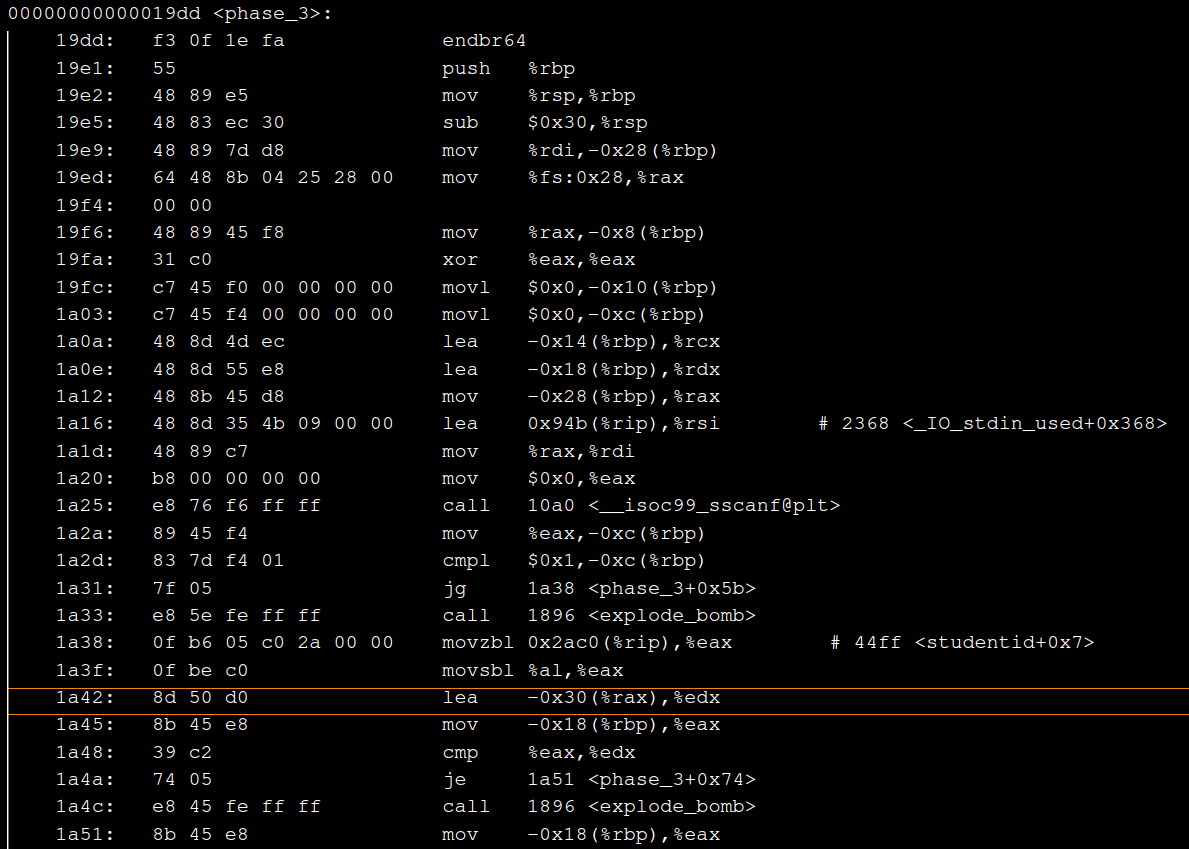


图5 phase\_2的汇编代码段截图（推断后4位数字）

**phase\_3:**

通过汇编代码可以看出，rdi 保存了输入的 int 型数值(亦即 rbp-0x28 处的值) ，接着会调用 \_\_isoc99\_sscanf@plt,这个函数会判断是否为%d。由汇编代码可看出，第一个数字是学号的倒数第三个数字，第二个数字保存在rbp-0x10中，是一个switch语句结构，只要我们能够定位到指令跳转到跳转表的位置即可得到，因为所有case都是简单的mov语句，只是简单地赋值。可以得到第二个数字是0x11c,即284。



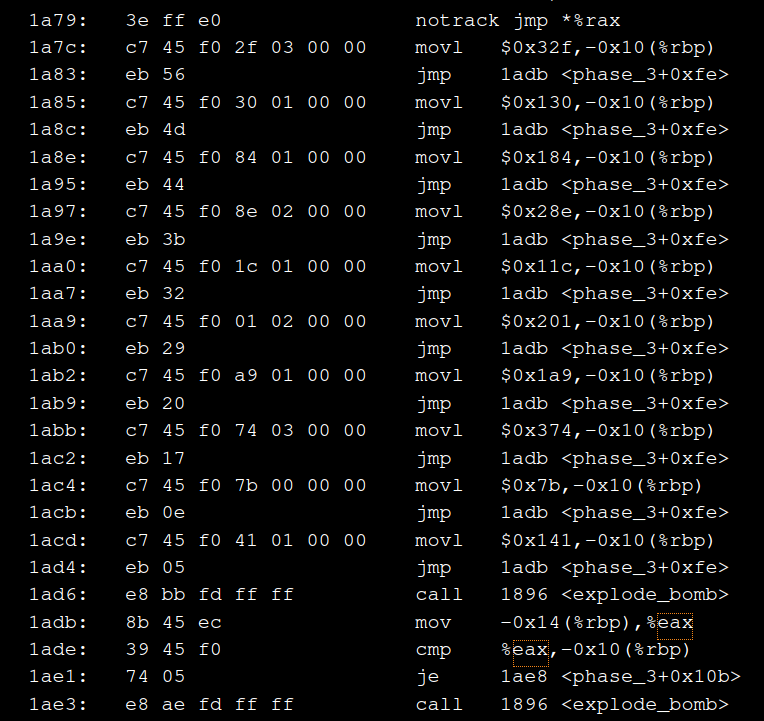


图6 phase\_3汇编代码段截图

**phase\_4:**

根据下面两段汇编代码段可得出输入的第一个数字必须在0-14之间：

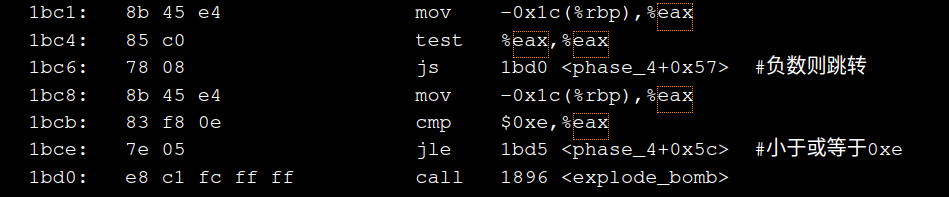


图7 phase\_4汇编代码段截图

接着根据将0 7 14送入相应寄存器的操作，猜测要输入的数字。可以看到最后要eax与rbp-0x10处的值进行比较，这个值为7,逆推得到第一个数应该是14,第二个数是7.

**phase\_5:**

通过汇编代码可以得到，输入的字符串长度应为6，且有一个数组，其内容为"maduiersnfotvbyl8\003"，输入的字符串将取出其低4位数字并从字符串数组中取出对应下标的字符拼接在一起，最后与字符串”bruins”进行比较。

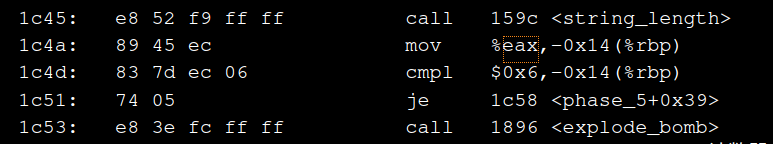


图8 确定字符长度为6



图9 查看字符池截图

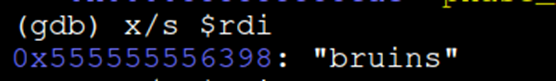
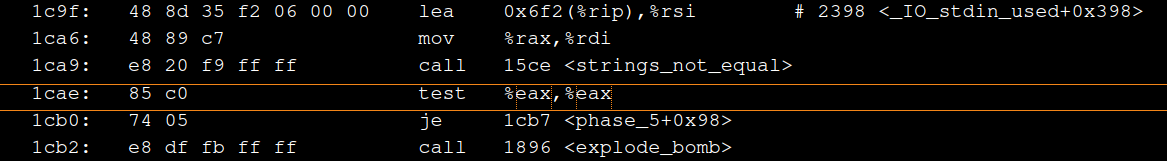
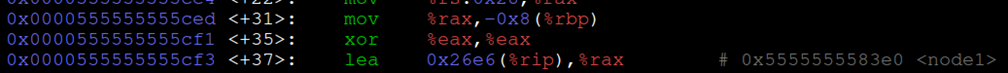


图10 确定输出的字符串

**phase\_6:**

在反汇编过程中发现node1的具体地址，利用gdb的功能查看相应地址的内容。这里只展示node1的内容，其他节点内容同理可得。从node1到node6的值分别为281 907 322 121 824 528。

****

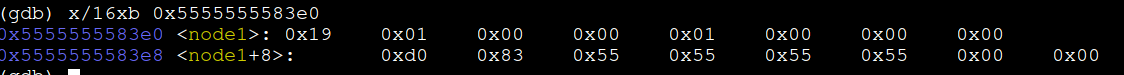


图11 确定node1的地址以及其值

**四、体会**

本次实验过程中，我学习到了通过gdb进行调试，以及学会了使用反汇编工具进行调试，对汇编语言有了更深的了解，比如跳转表、寄存器等内容。本次实验难度较大，需要我们有耐心解决问题，要学会一点一点追踪，掌握好常用的调试指令，才能成功完成实验并且从中有所收获。