

**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 ： 本硕博202001**

**学 号 ： U202115666**

**姓 名 ： 刘文博**

**指导教师 ： 许向阳**

**2022 年 10 月 1 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5.1** 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。

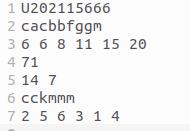
**阶段6：链表、结构、指针的访问 phase\_6(char \*input);**

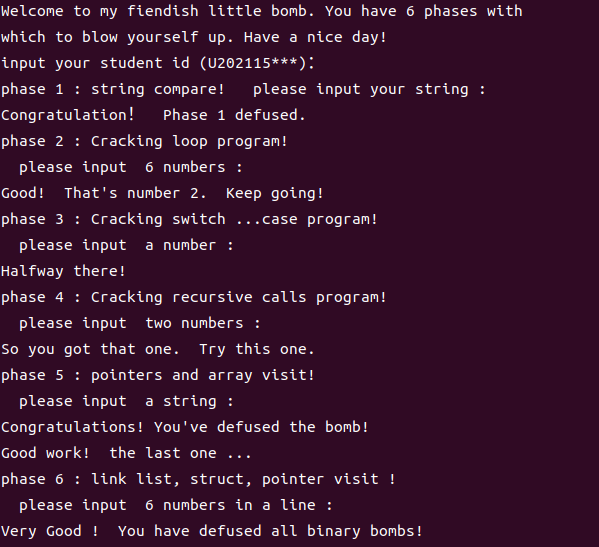
要求在一行中输入6个数，这6个数是一个链表中结点的顺序号（从 1 到6）。按照输入的顺序号，将对应链表结点中的值形成一个数组。若该数组是按照降序排列的，则过关。

**三、实验记录及问题回答**

**（1）实验任务 5.1 的实验记录**

采用文件输入的方式input.in文件内容如下

****

****

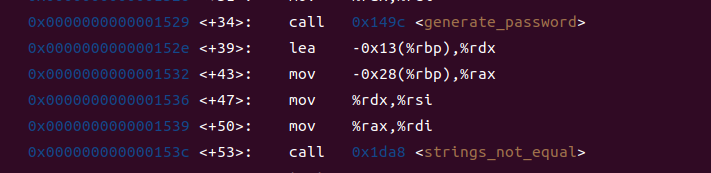
**（2）拆除炸弹的过程中关键操作**

通过阅读bomb.c 可知输入方式可以通过参数1 2 3来控制 initialize\_bomb();进行了炸弹的初始化 学生学号保存到了stuid中 之后分别根据输入来执行phase1至phase6 最终拆除炸弹

以下为反汇编phase1至phase6得到的结果与分析

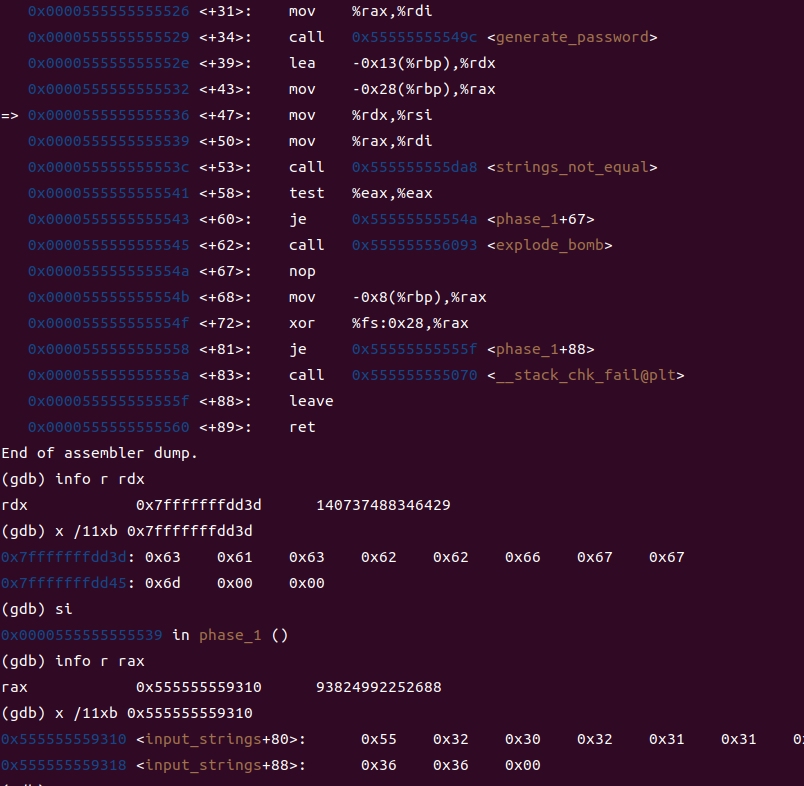
phase\_1

对phase\_1进行反汇编，由于在调用strings\_not\_equal之前将rdx和rax的值进行了mov操作拷贝到rsi rdi之中，推断特定串的地址保存在寄存器rdx中



之后将断点设置在generate\_password之后 且在strings\_not\_equal之前

找到内存中以rdx和rax为地址的值

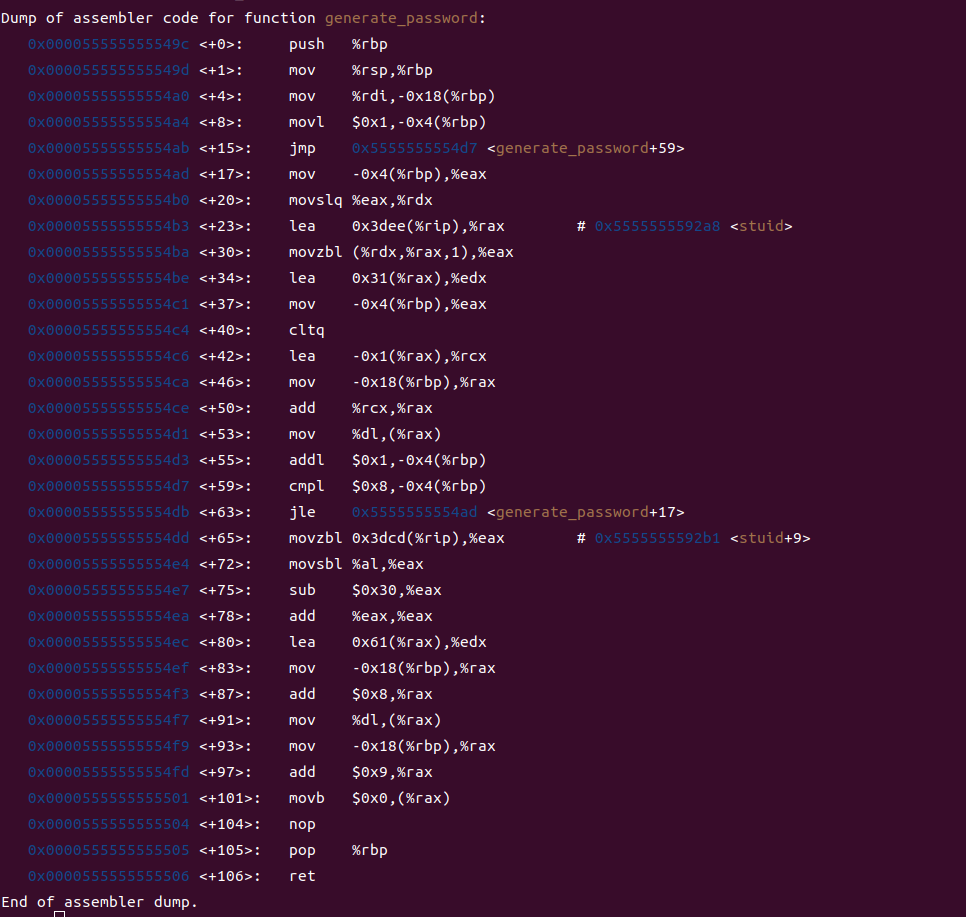


发现此时以rdx为地址的字符串为cacbbfggm

以rax为地址的字符串为U202115666

至此字符串炸弹得以破解

通过反汇编发现phase\_1调用了一个generate password的子函数，这个子函数是得到正确拆除串比较炸弹的关键。故对其进行反汇编，结果如下



可以发现rbp-4为一个下标从1~8递增，当rbp>8时跳出循环

rax中保存了input字符串的地址

注意到以下指令语句

0x00005555555554be <+34>: lea 0x31(%rax),%edx

可知对于U之后的8位数字进行了 c+31的操作

但对第九位数字进行了特别处理 ,通过阅读以下指令语句可以看出

0x00005555555554e7 <+75>: sub $0x30,%eax

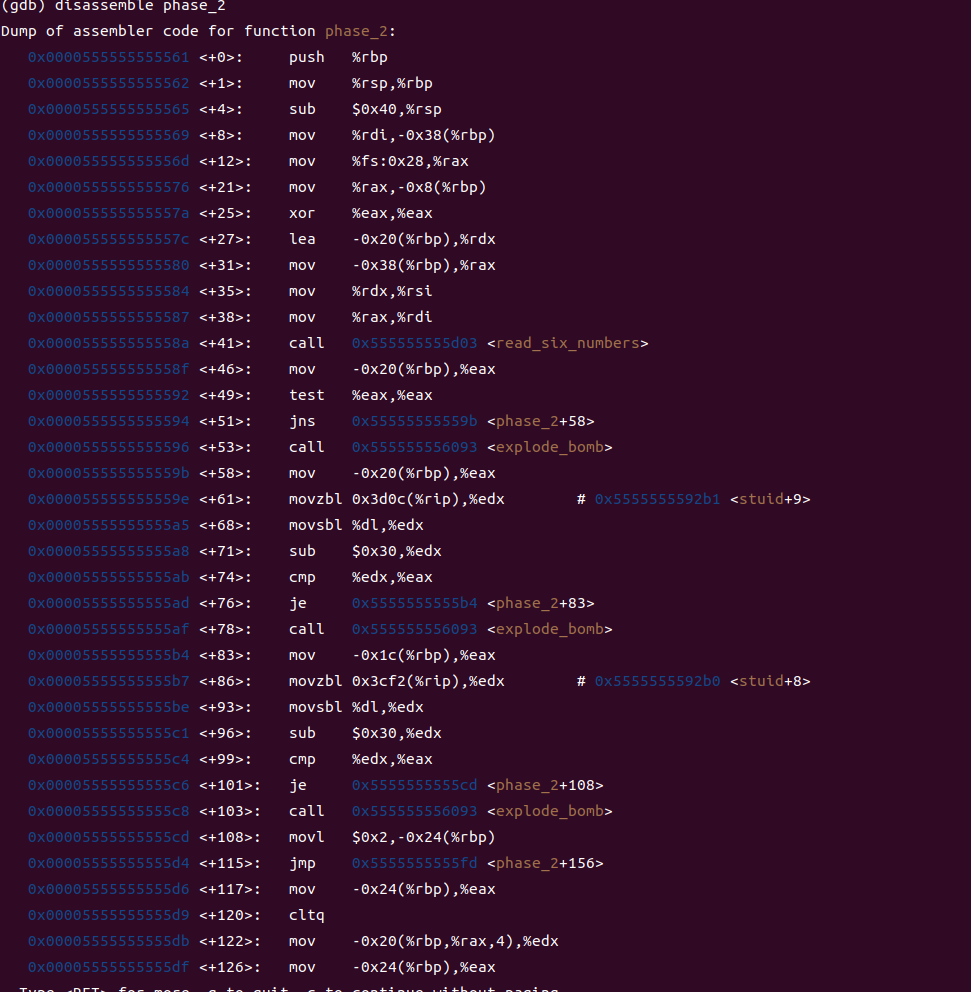
0x00005555555554ea <+78>: add %eax,%eax

0x00005555555554ec <+80>: lea 0x61(%rax),%edx

对第九位数字进行了(c-’0’)\*2+’A’的处理

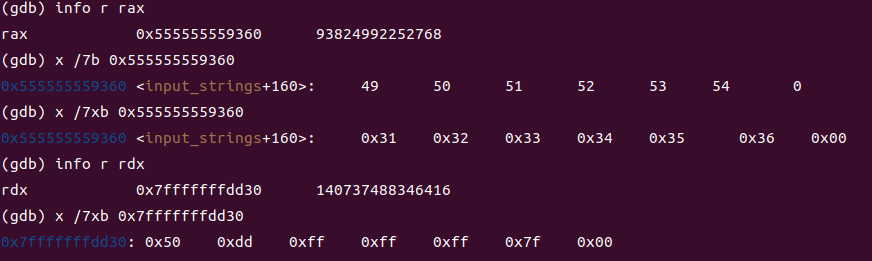
phase\_2

通过反汇编

****

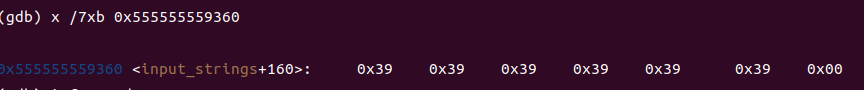
注意到read\_six\_numbers有两个参数input和numbers，作用是将input字符串读入到numbers字符数组中。

观察汇编源程序，rbp-20最后作为rsi，rbp-38最后作为rdi



通过对寄存器窗口的查看rdi最终保存的是我输入的数字（例如：123456），rsi保存的是0x50 0xdd 0xff 0xff 0xff 0x7f 0x00

之后尝试更改输入（例如：999999） 发现rsi中保存的值并未发生改变



**截图 2022-12-07 11-06-14**

阅读汇编代码注意到如下指令

0x000055555555559b <+58>: mov -0x20(%rbp),%eax

0x000055555555559e <+61>: movzbl 0x3d0c(%rip),%edx #0x5555555592b1<stuid+9>

0x00005555555555a5 <+68>: movsbl %dl,%edx

0x00005555555555a8 <+71>: sub $0x30,%edx

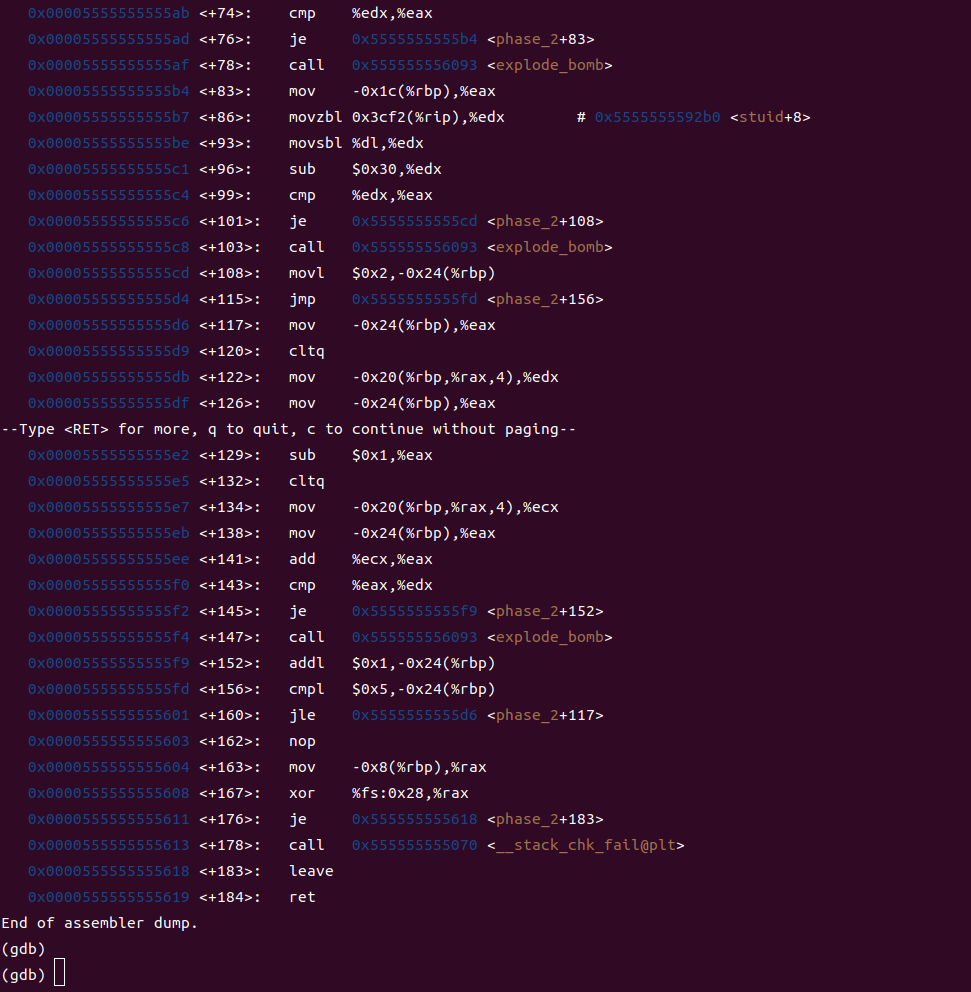
0x00005555555555ab <+74>: cmp %edx,%eax

0x00005555555555ad <+76>: je 0x5555555555b4 <phase\_2+83>

0x00005555555555af <+78>: call 0x555555556093 <explode\_bomb>

可以发现之后的指令段和这些指令类似

其做的操作是将rdi中的前两个双字数据依次与学号的9和8 减去30作比较

****

观察上述反汇编代码，可以发现rbp-24将从2变到5正好对应了4个数字

观察以下具有循环结构的指令段，发现

0x000055555555556a <+117>: mov -0x24(%rbp),%eax

0x000055555555556d <+120>: cltq

0x000055555555556f <+122>: mov -0x20(%rbp,%rax,4),%edx

0x0000555555555573 <+126>: mov -0x24(%rbp),%eax

0x0000555555555576 <+129>: sub $0x1,%eax

0x0000555555555579 <+132>: cltq

0x000055555555557b <+134>: mov -0x20(%rbp,%rax,4),%ecx

0x000055555555557f <+138>: mov -0x24(%rbp),%eax

0x0000555555555582 <+141>: add %ecx,%eax

0x0000555555555584 <+143>: cmp %eax,%edx

0x0000555555555586 <+145>: je 0x55555555558d <phase\_2+152>

0x0000555555555588 <+147>: call 0x555555556027 <explode\_bomb>

0x000055555555558d <+152>: addl $0x1,-0x24(%rbp)

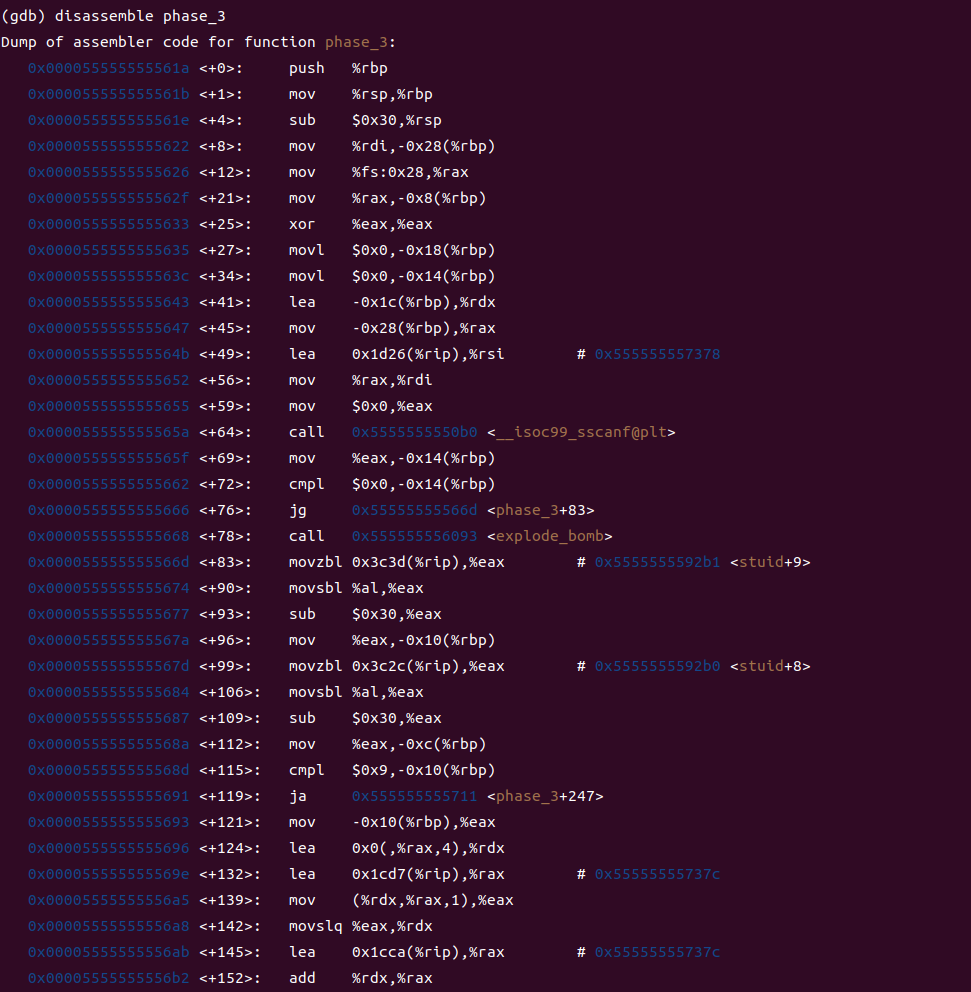
可以看出-0x20(%rbp,%rax,4)含义为从以rbp-20为基址rax为下标4为元素长度的基址加变址中找到相应元素

判断的条件是edx（保存当前元素，相当于c[i]）与eax（保存前一个元素加上当前元素的下标，相当于c[i-1]+i）相等。

故可以推算出后4个数字为6+2=8 8+3=11 11+4=15 15+5=20

phase\_3

通过反汇编



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

之后会将学号第9位-30后送到rbp-0x10处 将学号第八位-30后送到rbp-0x12处

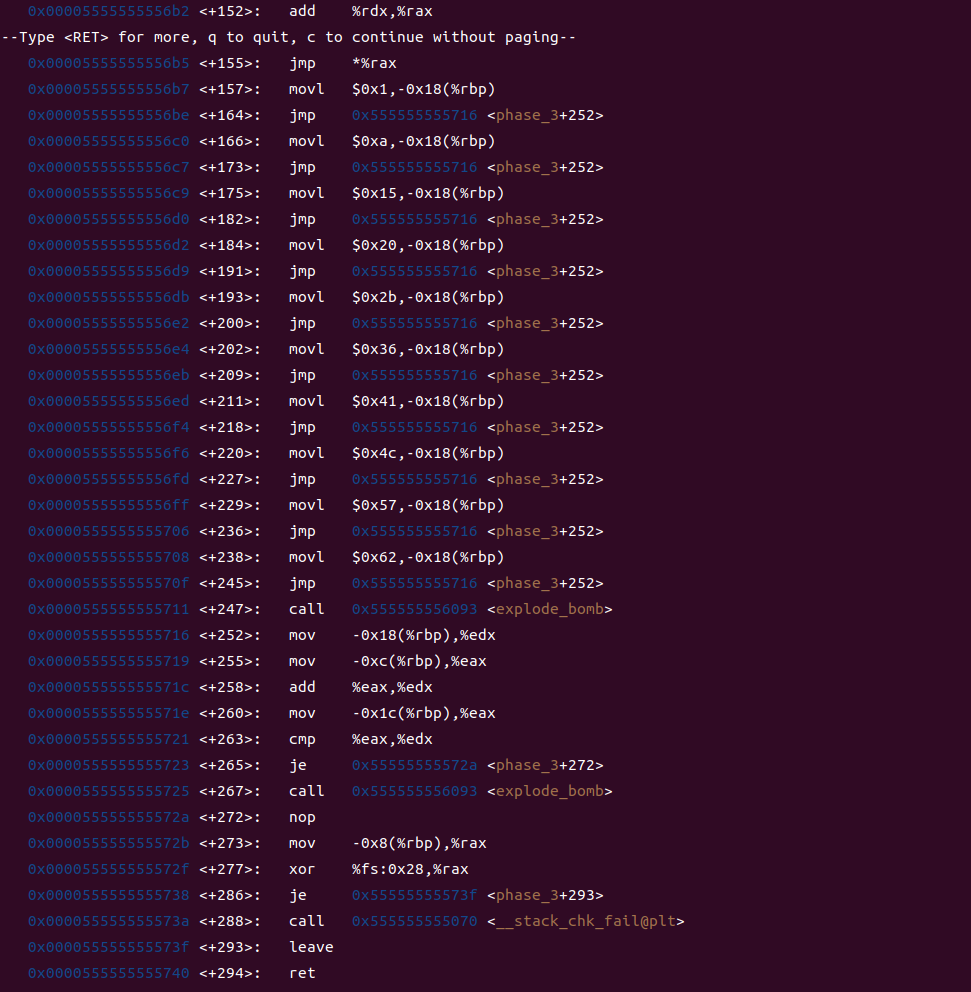
如果rbp-10处的值小于等于9则将其乘4,加上偏移量之后得到rax的地址，jmp \*%rax跳转到相应的case语句

通过寄存器窗口监视rax的值

截图 2022-12-07 16-39-05

发现对应了下面的语句 0x0000555555555681 <+211>: movl $0x41,-0x18(%rbp)

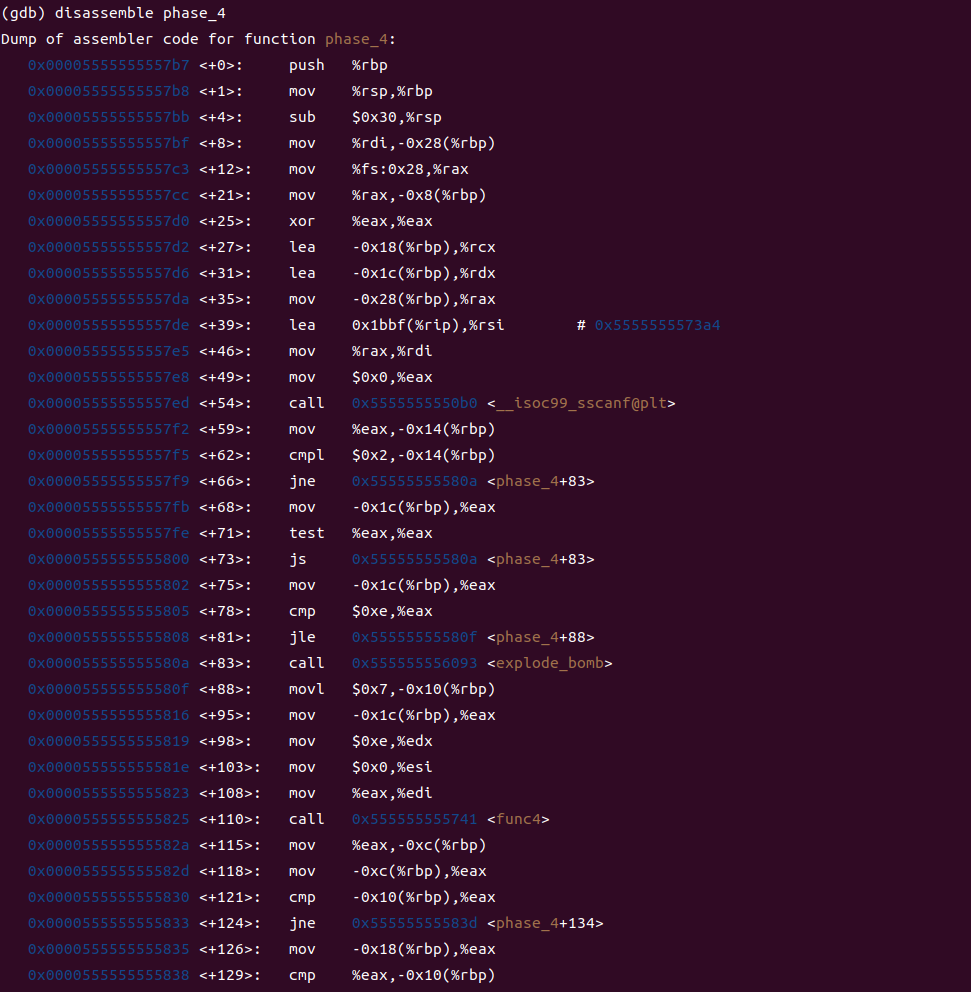
说明最后将65保存在了rbp-0x18处

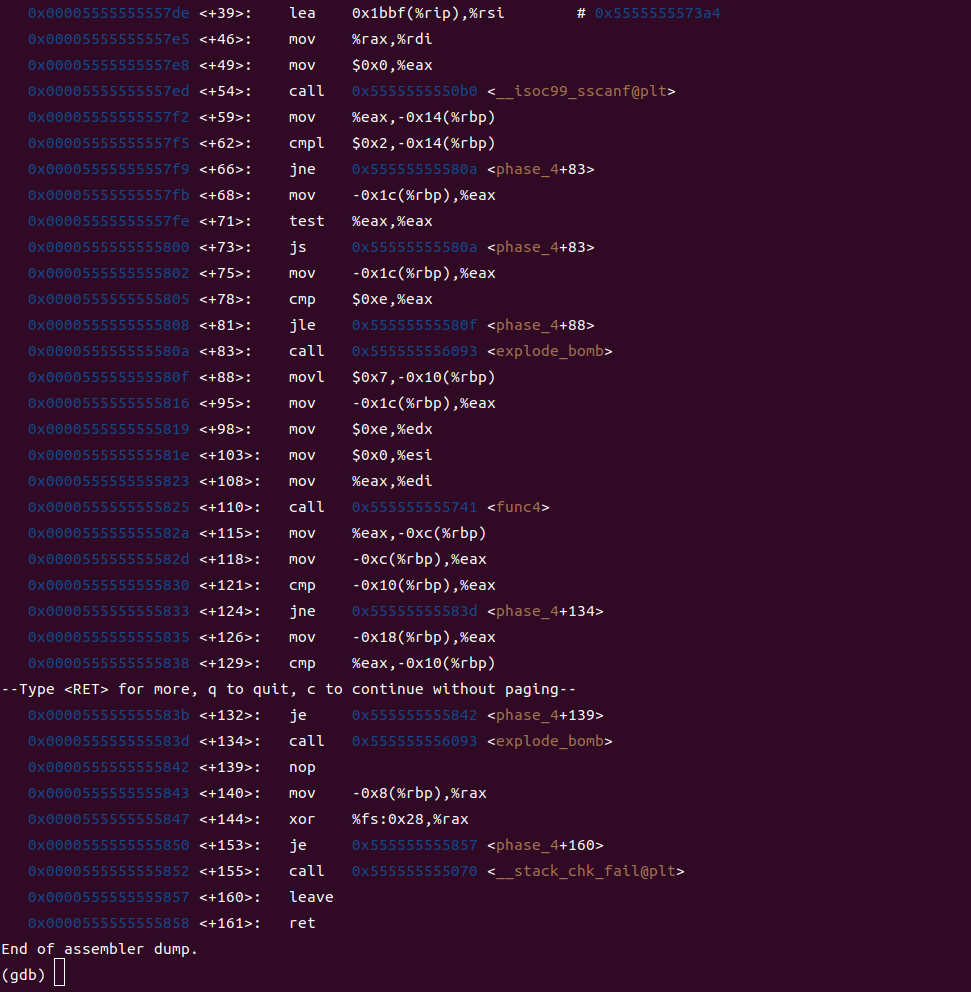


后面的指令则表示，将rbp-0x18和学号第8位相加，得到的数字和rbp-0x1c处的数据进行比较判断是否相等。此时rbp-0x1c保存的正是输入的整数。所以输入的整数应该是65+6=71.

phase\_4

反汇编得到





首先根据

0x0000555555555781 <+54>: call 0x5555555550b0 <\_\_isoc99\_sscanf@plt>

0x0000555555555786 <+59>: mov %eax,-0x14(%rbp)

0x0000555555555789 <+62>: cmpl $0x2,-0x14(%rbp)

0x000055555555578d <+66>: jne 0x55555555579e <phase\_4+83>

得出如果输入不足2个整数会出错

然后根据

0x000055555555578f <+68>: mov -0x1c(%rbp),%eax

0x0000555555555792 <+71>: test %eax,%eax

0x0000555555555794 <+73>: js 0x55555555579e <phase\_4+83>

得出rbp-0x1c保存了输入的第一个数字，如果这个数字是负数的话则会出错

=> 0x0000555555555799 <+78>: cmp $0xe,%eax

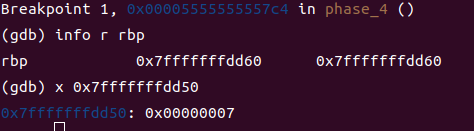
0x000055555555579c <+81>: jle 0x5555555557a3 <phase\_4+88>

得出rbp-0x1c保存了输入的第一个数字，如果这个数字不小于14的话则会出错

根据将0 7 14送入相应寄存器的操作，猜测要在0-14的二叉搜索树中找到输入的数字

由于输入的数字为5 猜测其后的数字应该是2(编码为10) ，通过查看寄存器窗口发现此时eax为2

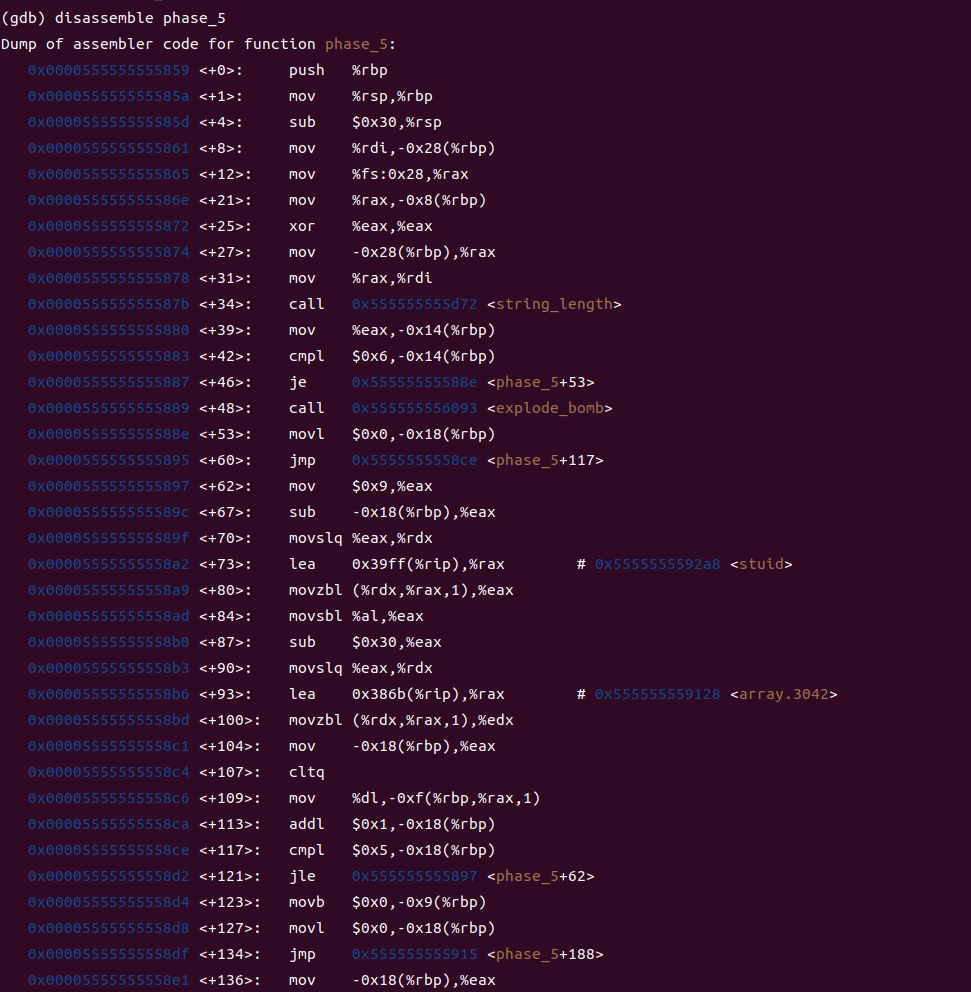
截图 2022-12-07 17-54-25

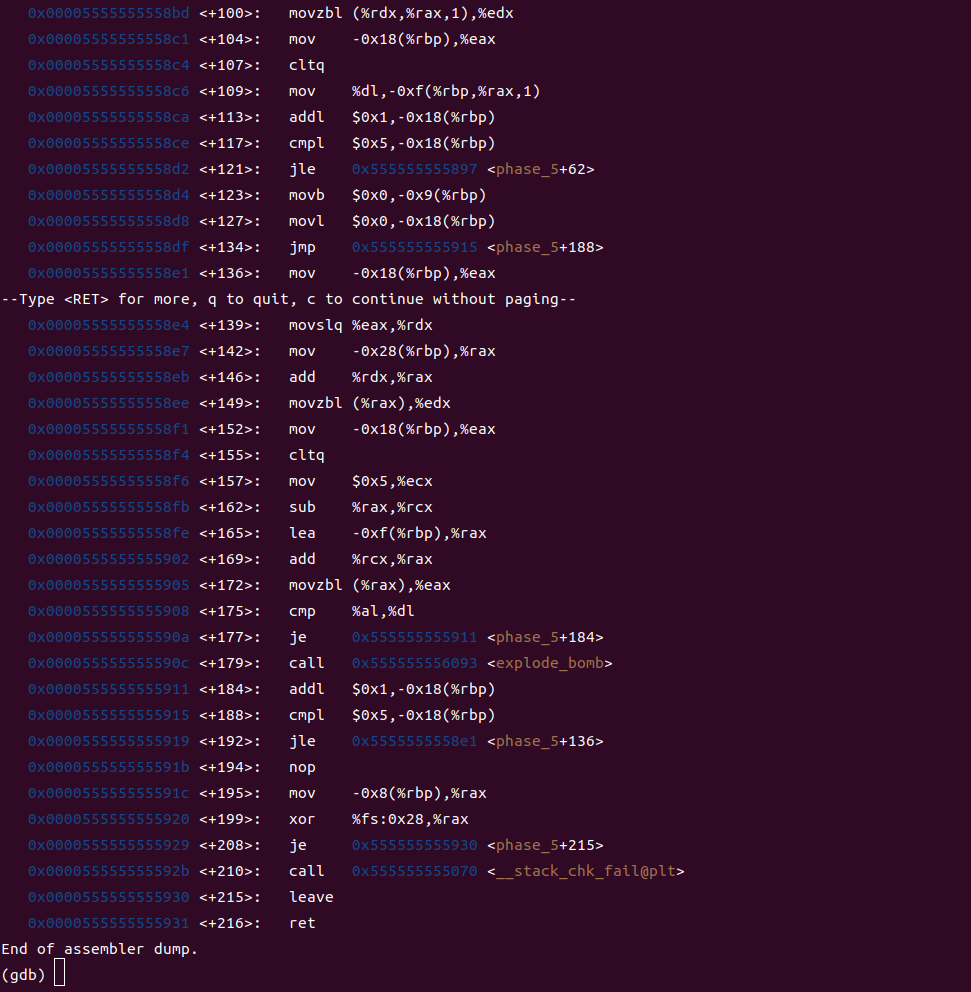
对于其他的数字（0-14）也可以用类似的方法得到其编码 

可以看到最后要eax与rbp-0x10处的值比较，这个值为7（编码为111），逆推得到第一个数应该为14 第二个数为7 至此phase\_4的炸弹破解完毕。

phase\_5

反汇编得到

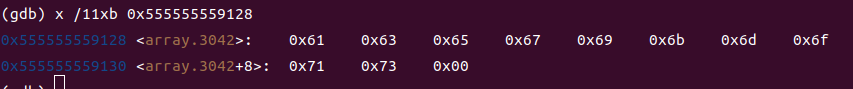




通过观察可以得出输入的字符串长度为6 保存在rbp-0x28处

0x000055555555582b <+62>到0x0000555555555866 <+121>的语句表示从0~5进行循环 将学号的9~4位存入rdx中 之后将array.3042的第rdx位存入edx，最后存入以rbp-0xf为首址的字符串中

使用gdb查看array.3042的内容



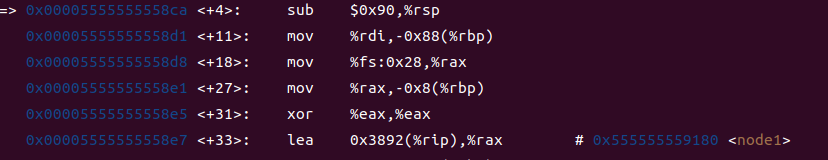
之后的0x0000555555555875 <+136>到0x00005555555558ad <+192>的内容表示将以rbp-0xf为首址的字符串以倒序和输入的字符串作比较，判断是否相等。

根据学号为U202115666，则将后6位逆序666511分别与array.3042中的元素对应，则为mmmkcc,再倒序为cckmmm，至此phase\_5炸弹破解完毕。

phase\_6

老样子，反汇编

phase\_6的反汇编代码太长这里不再插入图片。



在反汇编过程中发现了node1的地址为0x555555559180

在phase.h中查看结构体的定义

typedef struct nodeStruct

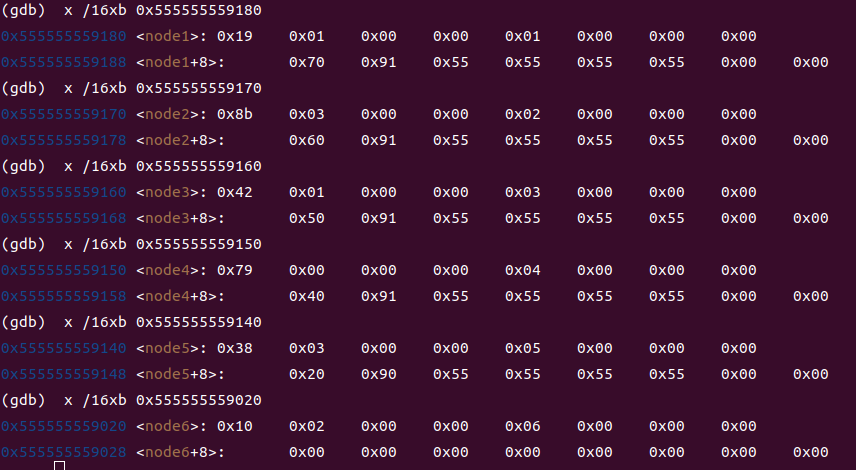
{

int value;

int index;

struct nodeStruct \*next;

} listNode;

利用gdb的查看功能，查看相应地址中的内容，并顺藤摸瓜，找出来所有6个node

容易知道前4个字节的内容为相应节点的value值，故降序排列顺序应该为2 5 6 3 1 4

至此phase\_6破解完毕。

1. **体会**

破解二进制炸弹是一个较难的工作，在这一过程中获得的成就感也是与以往编程所不同的，面对复杂问题时，必须要有不怕困难、敢于钻研的韧劲，要学会一点一点追踪，掌握好用的调试指令，这样才能成功。

凡事都需要一个认识与熟悉的过程，初次接触二进制炸弹，phase\_1虽然是最简单的一个环节但破解它花费的时间占据了整个实验的50%，之后的问题在经过一定的调试步骤之后都迎刃而解，所以遇到问题不能一开始就被吓到，掌握了方法之后就可以事半功倍！