

**实验报告**



**题目： 拆解二进制炸弹**

**班 级： 2020211305**

**学 号： 2020211346**

**姓 名： 倪玮昊**

**学 院： 计算机学院**

**2021年 11 月 15 日**

一、实验目的  
1.理解C语言程序的机器级表示。  
2.初步掌握GDB调试器的用法。  
3.阅读C编译器生成的x86-64机器代码，理解不同控制结构生成的基本指令模式，过程的实现。

1. 实验环境
2. shell
3. Objdump命令反汇编
4. GDB调试工具
5. 积分榜（http://10.120.11.13:19220/scoreboard）

三、实验内容

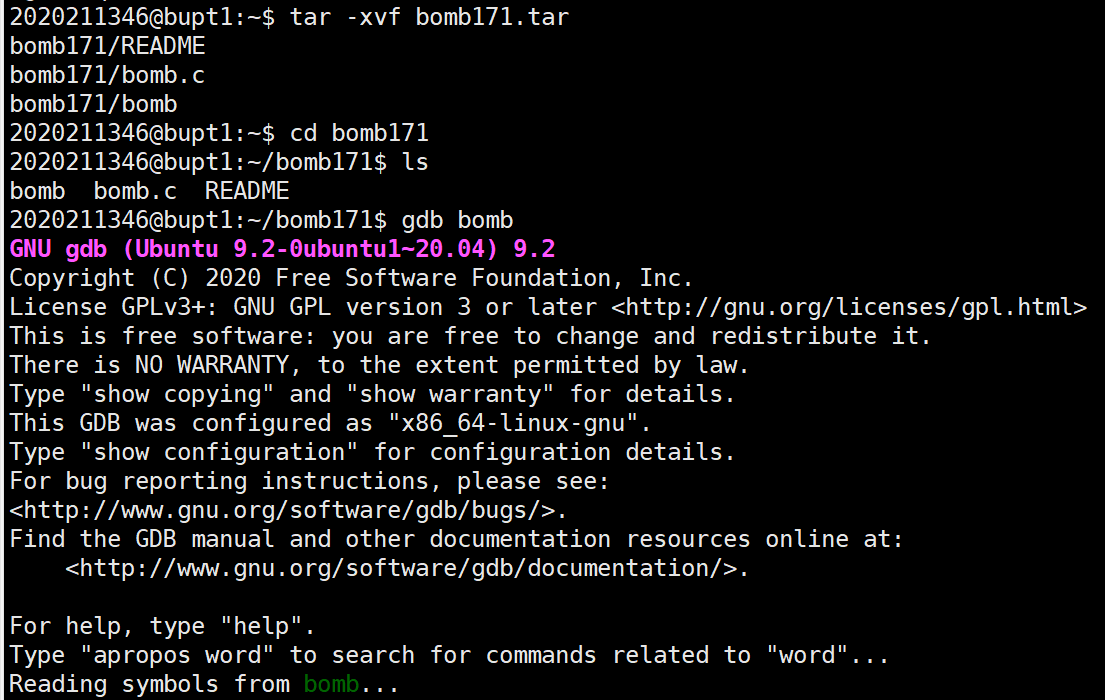
登录bupt1服务器，在home目录下可以找到Evil博士专门为你量身定制的一个bomb，当运行时，它会要求你输入一个字符串，如果正确，则进入下一关，继续要求你输入下一个字符串；否则，炸弹就会爆炸，输出一行提示信息并向计分服务器提交扣分信息。因此，本实验要求你必须通过反汇编和逆向工程对bomb执行文件进行分析，找到正确的字符串来解除这个的炸弹。

本实验通过要求使用课程所学知识拆除一个“binary bombs”来增强对程序的机器级表示、汇编语言、调试器和逆向工程等方面原理与技能的掌握。 “binary bombs”是一个Linux可执行程序，包含了5个阶段（或关卡）。炸弹运行的每个阶段要求你输入一个特定字符串，你的输入符合程序预期的输入，该阶段的炸弹就被拆除引信；否则炸弹“爆炸”，打印输出 “BOOM!!!”。炸弹的每个阶段考察了机器级程序语言的一个不同方面，难度逐级递增。

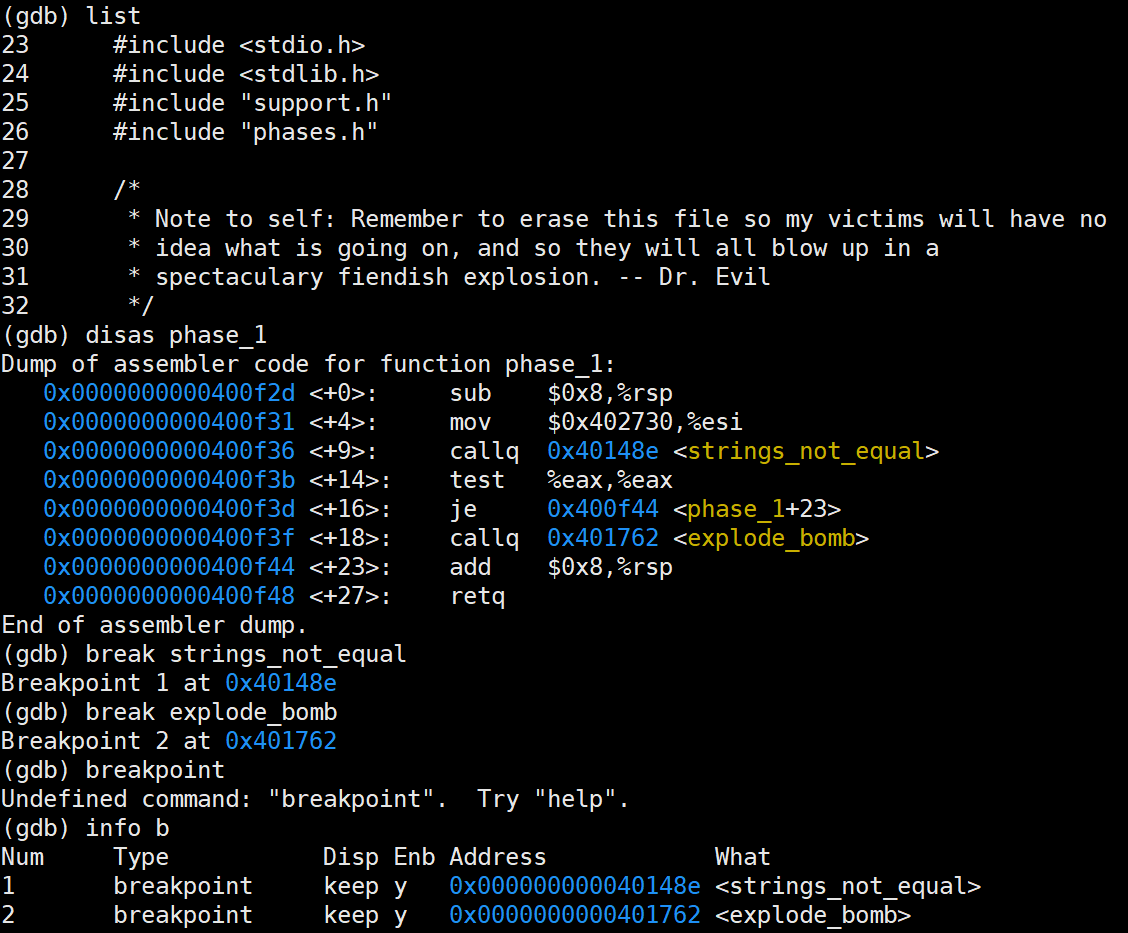
为完成二进制炸弹拆除任务，需要使用gdb调试器和objdump来反汇编bomb文件，可以单步跟踪调试每一阶段的机器代码，也可以阅读反汇编代码，从中理解每一汇编语言代码的行为或作用，进而设法推断拆除炸弹所需的目标字符串。实验2的具体内容见实验2说明。

四、实验步骤及实验分析

**准备阶段：**

（1.1）

将压缩包bomb171进行解压，使用cd命令切换到bomb171路径下，使用gdb对bomb进行调试。

（1.2）

输入list指令，发现头文件里有phases.h 根据教学文档内容判断该头文件包含了需要解决的6个题目。

**炸弹一：**

如图1.2所示，在+18行的位置有炸弹，如果想要跳过该炸弹必须实施+16行的跳转，又因为+16行的跳转条件为等于0，即在+9行使用call指令返回的值必须是0才能成功跳过炸弹，又因为call访问函数名字为字符串不相等，故判断必须输入一个和该函数中字符串相同的字符串才能返回1，又因为在call指令之前唯一有机会输入字符串的指令就是+4的mov，故访问源地址0x402730 如图（1.3）显示。

%(LE8)$IS17RD2$PDSMQ`{K（1.3）

故判断只有输入字符串The future will be better tomorrow.的时候才能拆除炸弹。

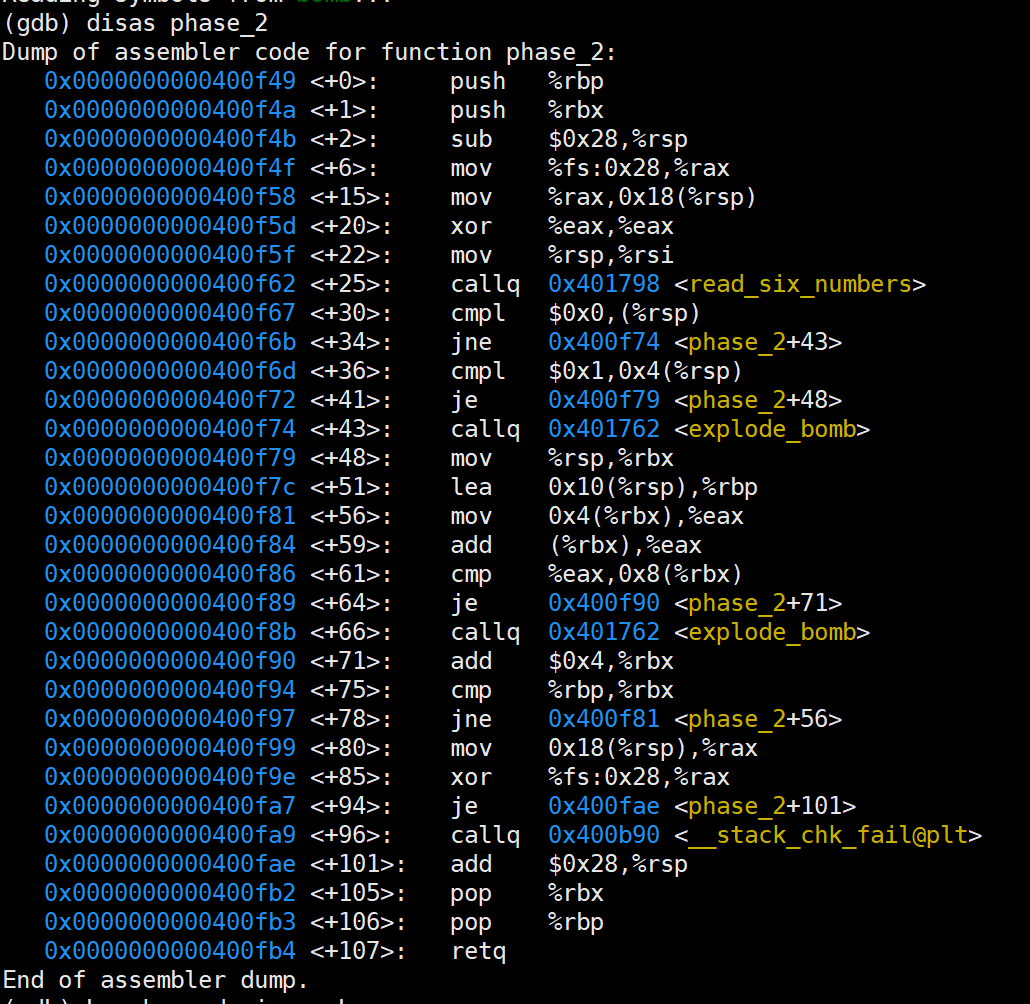
在explode\_bomb处设置断点（在图1.2最后面处）（后面五个拆炸弹都有这步，故在后面省略）

输入The future will be better tomorrow.炸弹成功拆除，进入炸弹2拆除过程。

![A6JL_7T[90_T$3](C5N_B5X](data:image/png;base64,)

**炸弹2：**

输入disas phase\_2 查看炸弹2源码

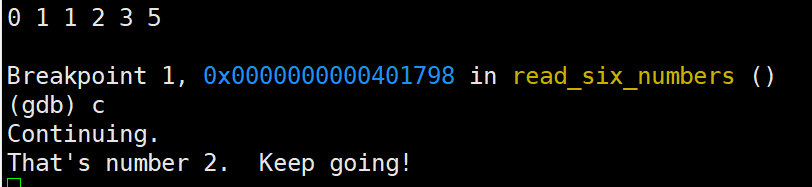


第二关调用了函数read\_six\_number判断该炸弹要输入6个数字

从+30行得知栈里存的第一个数为0（否则会跳转到+43直接爆炸），同理+36往栈里存第二个数1（一个int占4个字节，故栈往上加4个字节来容纳第二个数），从+51行和+56得到%rbp存储了栈里的第5个数字，%eax存了栈里的第二个数，+59行使栈里的第二个数等于自身加上第一个数，此时比较栈里第二个数和第三个数是否相等，若相等则避开炸弹并且栈底向上移动4个字节，如果%rbp==%rbx说明已经到了栈顶，跳出循环，否则回到+56行继续循环。

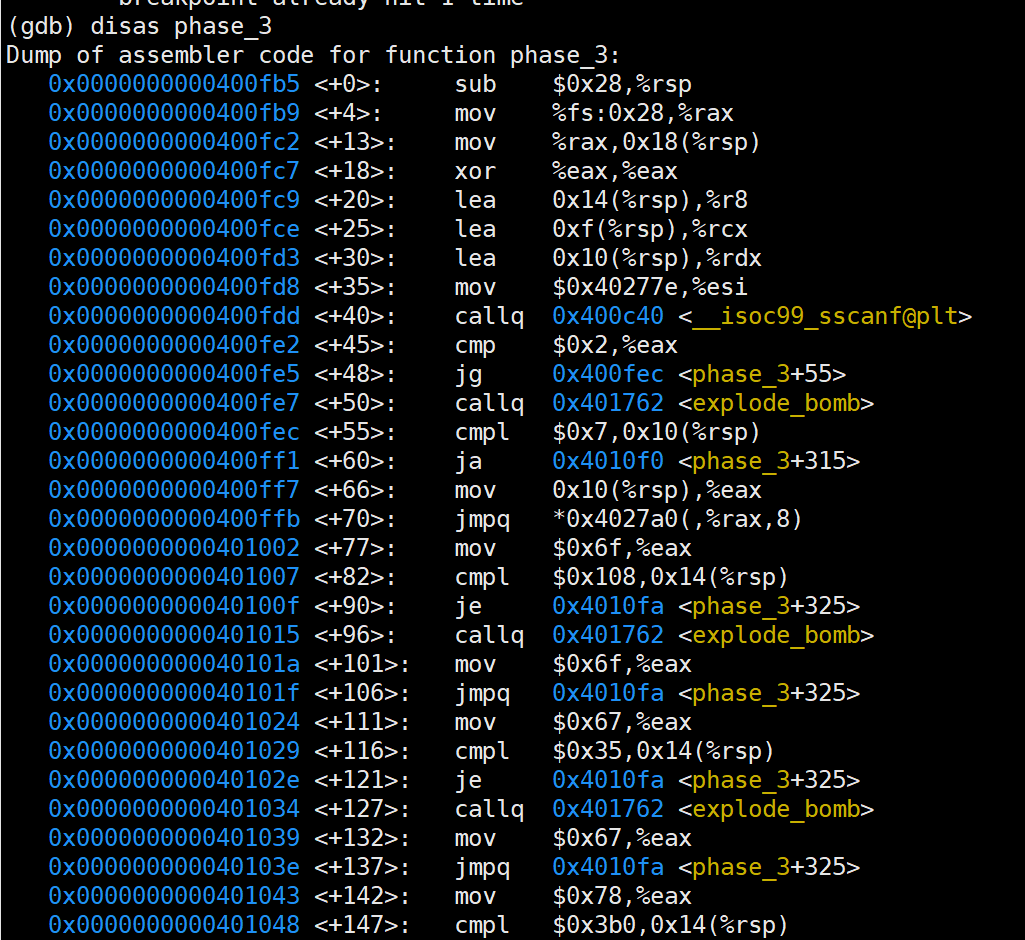
通过列出栈每个数的变化可以知道，每一个数都等于前两个数之和，又因为前两个数是0和1，所以六个数为0 1 1 2 3 5

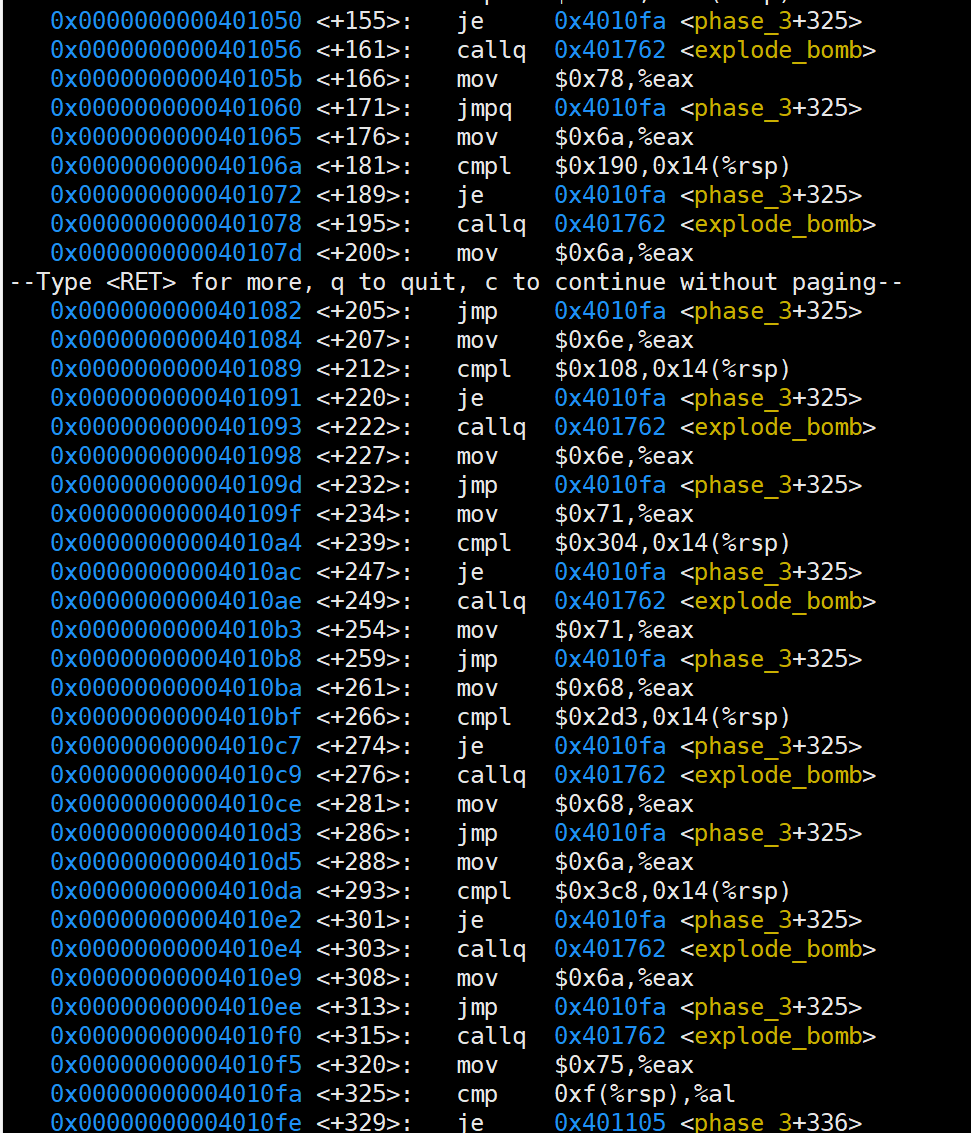
将函数运行到爆炸函数断点，输入该数列，成功解除第二个炸弹。

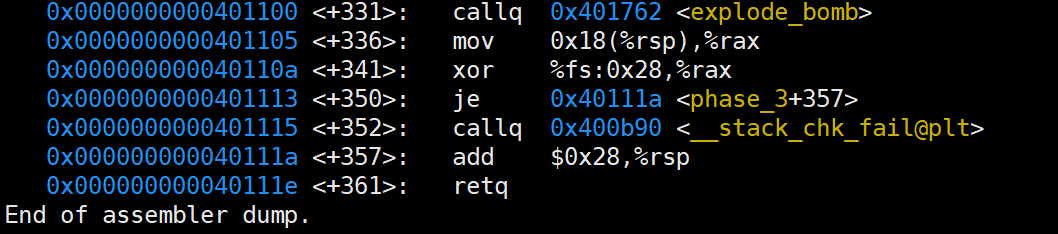


**炸弹3：**

查看phase\_3源代码：





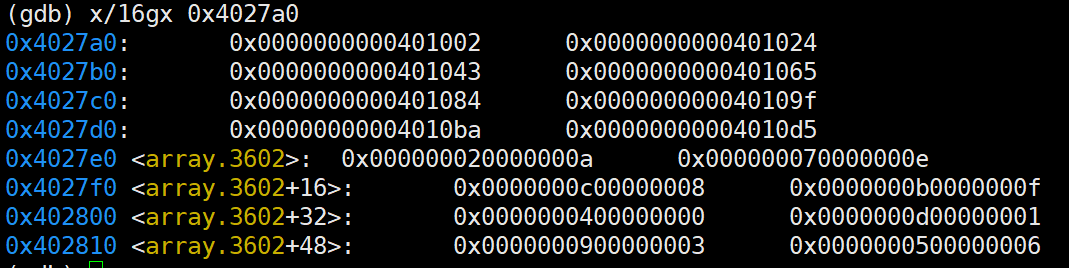


在调用的函数scanf前有一个地址，从判断可能是输入的内容，输入指令x/s40277e查看输入内容

R8YMSVG25_I_UL3JI5TV~SW

如图所示，输入的是两个数字和一个字符

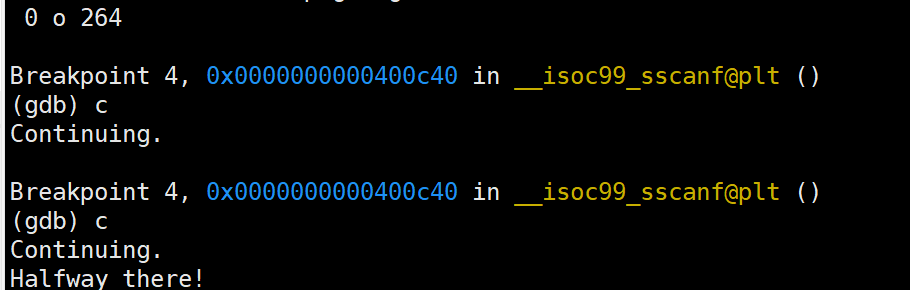
又由+20+25+30三行代码以及其位置确定0x10+%rsp存第一个数存在rdx，0xf+%rsp存第二个字符存在%rcx，0x1+%rsp存第三个数存在r8，又由+55行判断第一个数必须小于等于7，否则会爆炸。+70 有跳转指令，跳转地址为4027a0+8\*%rax，

使用x指令查看地址4027a0，如图：  


可见该跳转指令会根据不同的%rax来跳转到不同的位置去，所以显然phase\_3是一个switch结构。

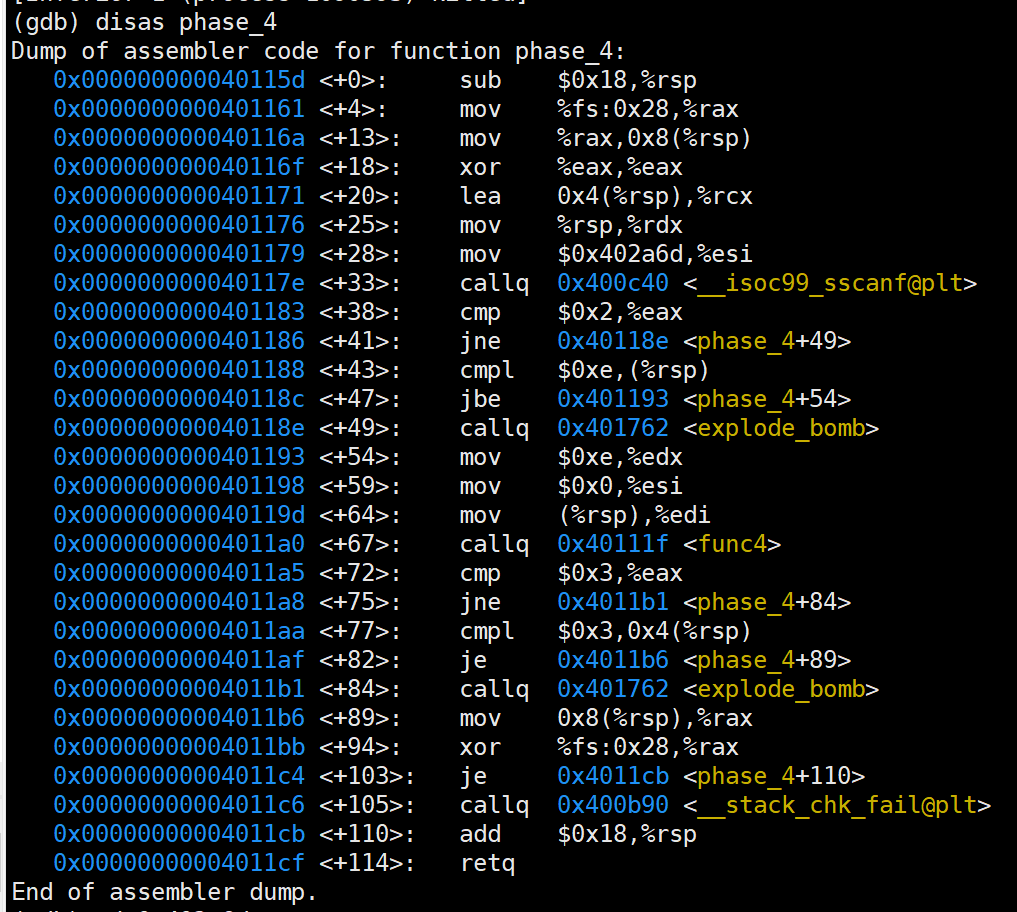
假设%rax储存值为0（即第一个数为0),跳转指令将跳转到地址0x401002,即图中的+77行，mov指令对%eax赋值，值为111，又在+325将第二个字符与%al比较（%al即%eax后8位），相同时能接触爆炸，又因为111对应ascii码为o，所以当第二个字符为o时能解除炸弹，在+82行，如果14+%rsp(上文提过，此地址储存的值为第三个数）等于264则跳过炸弹（跳转地址离开switch结构），所以第三个数为264.

故再次运行到爆炸函数，输入0 o 264，如图所示，成功拆解第三个炸弹。



**炸弹4：**

查看phase\_4的源代码，如图：



先查看地址0x402a6d，确定输入值：

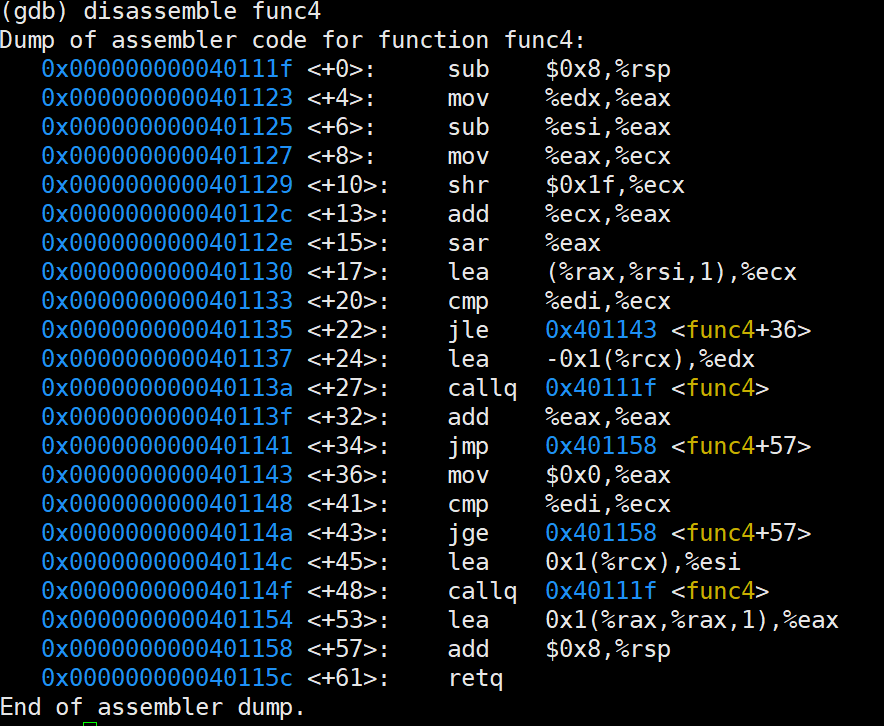
XOP}7(GHHR$`OB`D5Q1WN_L

输入值为两个整数。

+43行对输入的第一个值进行判断，如果第一个值小于等于14则不会爆炸，否则会爆炸。

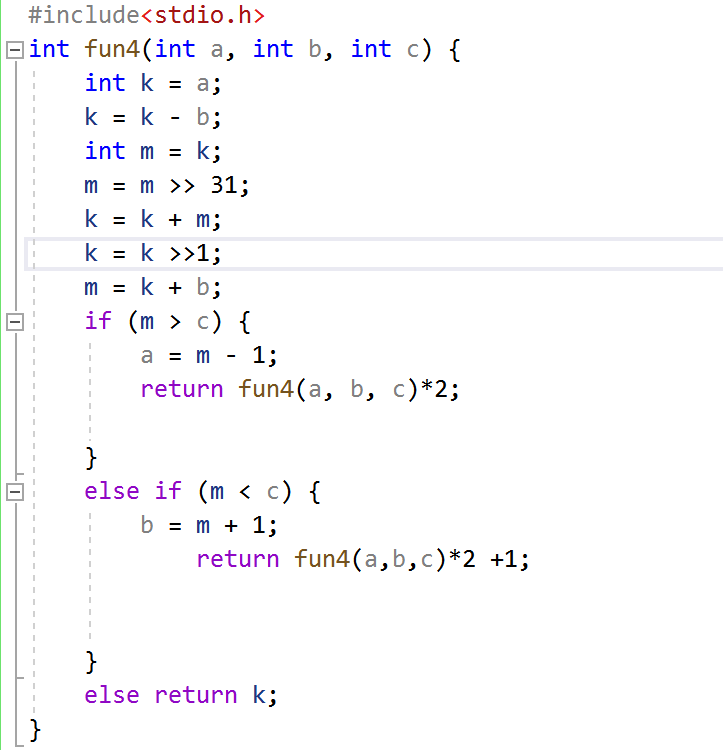
+54,+59,+64行分别向寄存器%edx赋值14，%esi赋值0，%edi赋值输入的第一个数。

+67行调用函数func4，使用disassemble指令来查看func4的源码

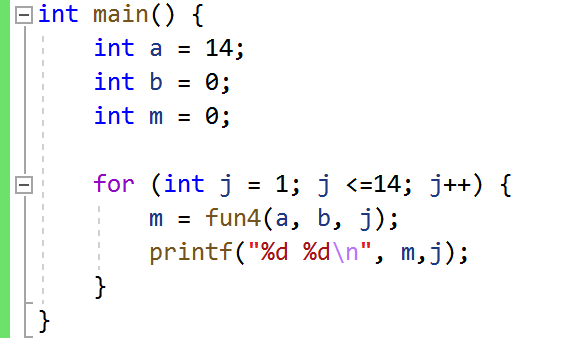


设%eax存储值为k，%ecx为m，%edx，%edi，%esi为带入函数的参数。

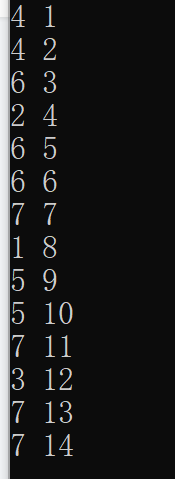
可以将汇编语言转换为c语言代码：



又因为+72行将函数返回值和3比较，如果返回值是3则跳过炸弹，+77将输入的第二个数和3比较，如果相等则跳过炸弹。故可以得到第二个输入数为3，且第一个数带入func4函数的返回值等于3，且第一个数小于等于14.故使用循环结构调用func4：



有输出结果：



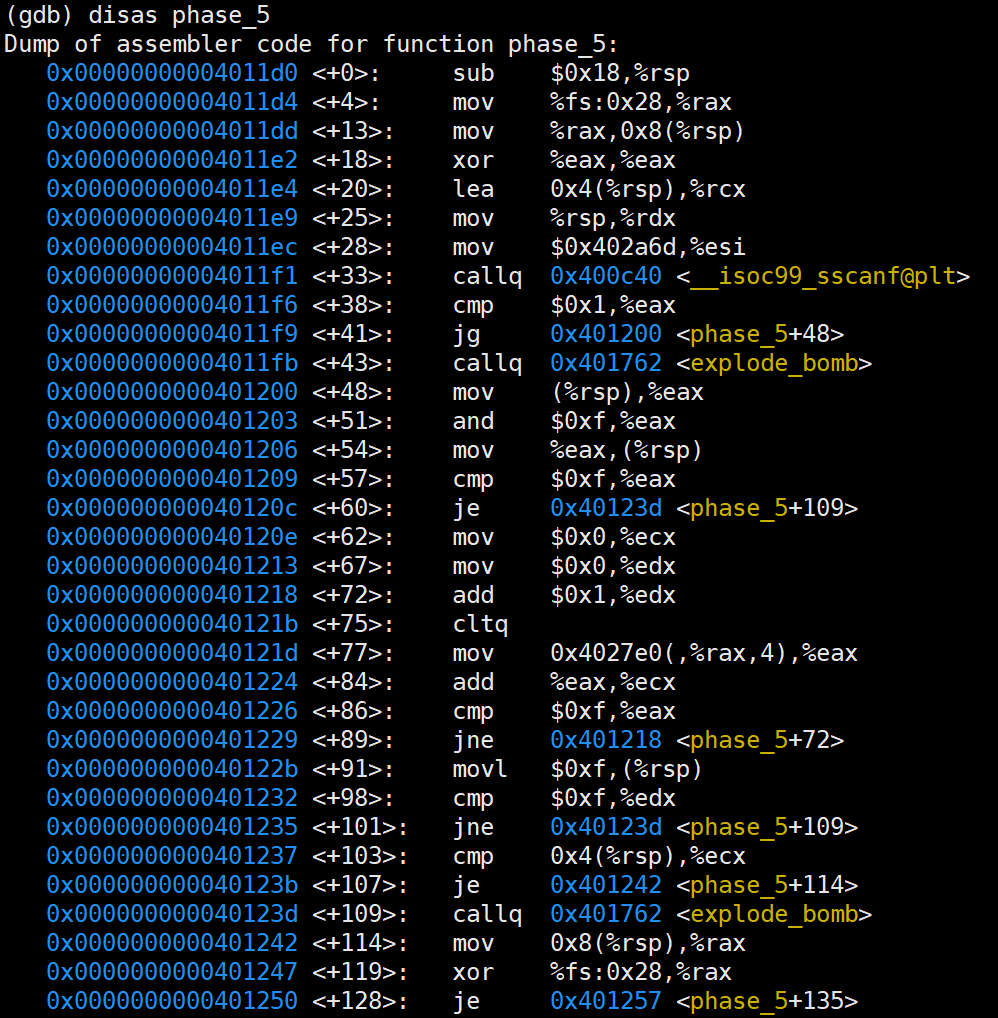
可知，能够使范围值为3的数是12，所以输入的两个数为12 3

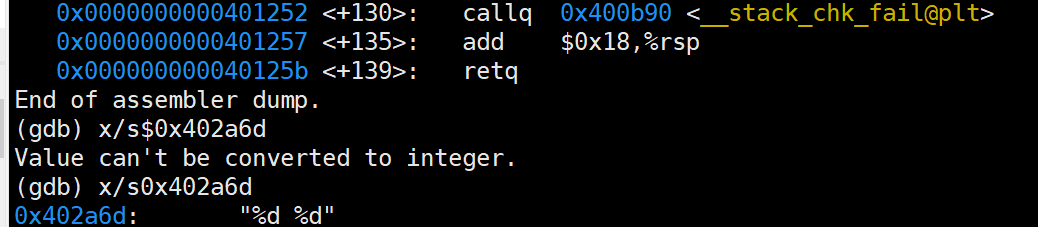
运行到爆炸函数，输入12 3，炸弹成功被拆除：

RH(SDRW(WRK%W7{8XQ3~403

**炸弹五：**

首先访问phase\_5源码





再访问地址0x40a6d确定输入形式：

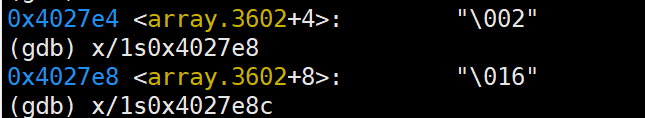
XOP}7(GHHR$`OB`D5Q1WN_L

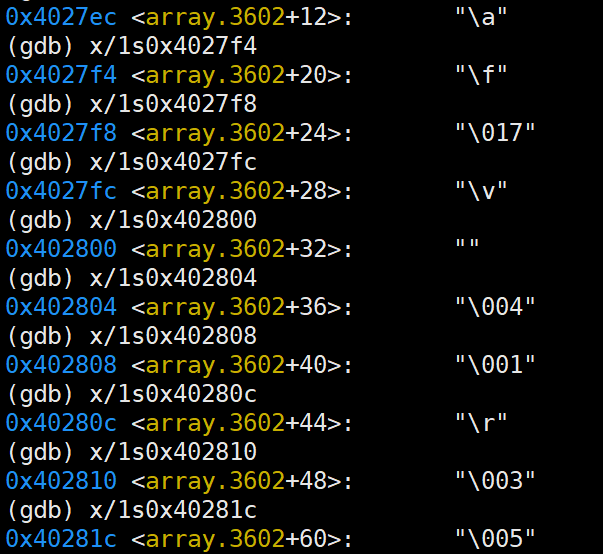
输入值为两个整数。

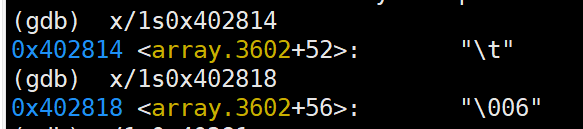
+48行中%eax被赋值为输入的第一个数,+57行中%eax与15做并预算，如果并运算过后%eax等于15则函数爆炸，所以输入的第一个数小于15（即后四位不全为1），再将%ecx赋值为0，%edx赋值为0，通过观察+86行，只有当%eax等于15，才会跳出+72到+86的循环，又因为+98行中，%eax必须等于15才能拆除炸弹，所以+72到+86的循环必须要进行15次（每次循环在+72行都会让%eax加1）。同时通过+77行得知，每一次eax的值等于上一次eax+4+0x4027e0里储存的值。

此时访问地址0x4027e0，不断加4，看每一个位置含的字符是什么：

VDD7ZNT@$H4@~@D]99L{ZPE



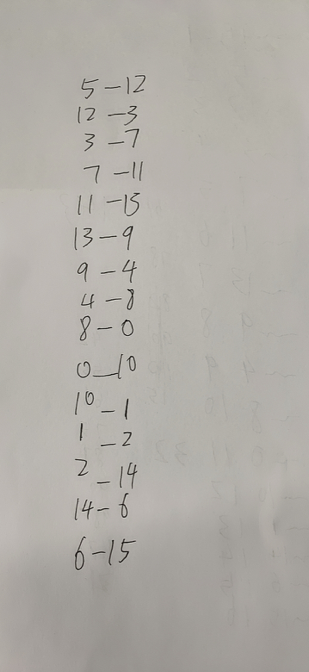




通过查询ascii码得知

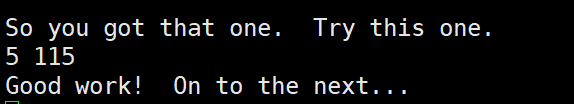
\n=10,\t=9,\b=8,\r=13,\v=11,\a=7,\f=12,\017=15,\016=14,””=0,\001=1,\002=2,\003=3,\004=4,\005=5

又由递推关系可知，要想让最后一次%eax等于15且循环15次，必须按以下顺序（左为%rax，右为%eax）



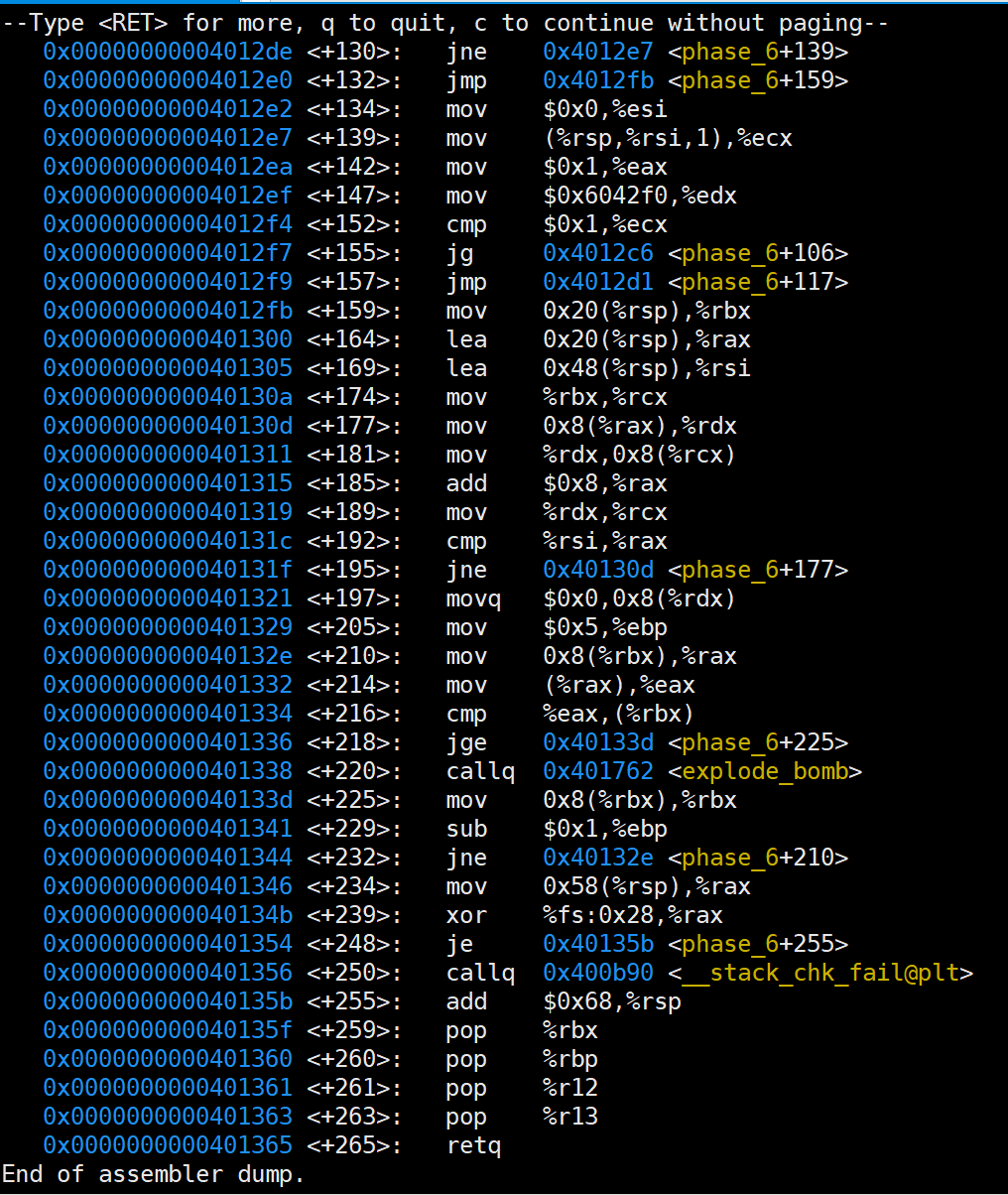
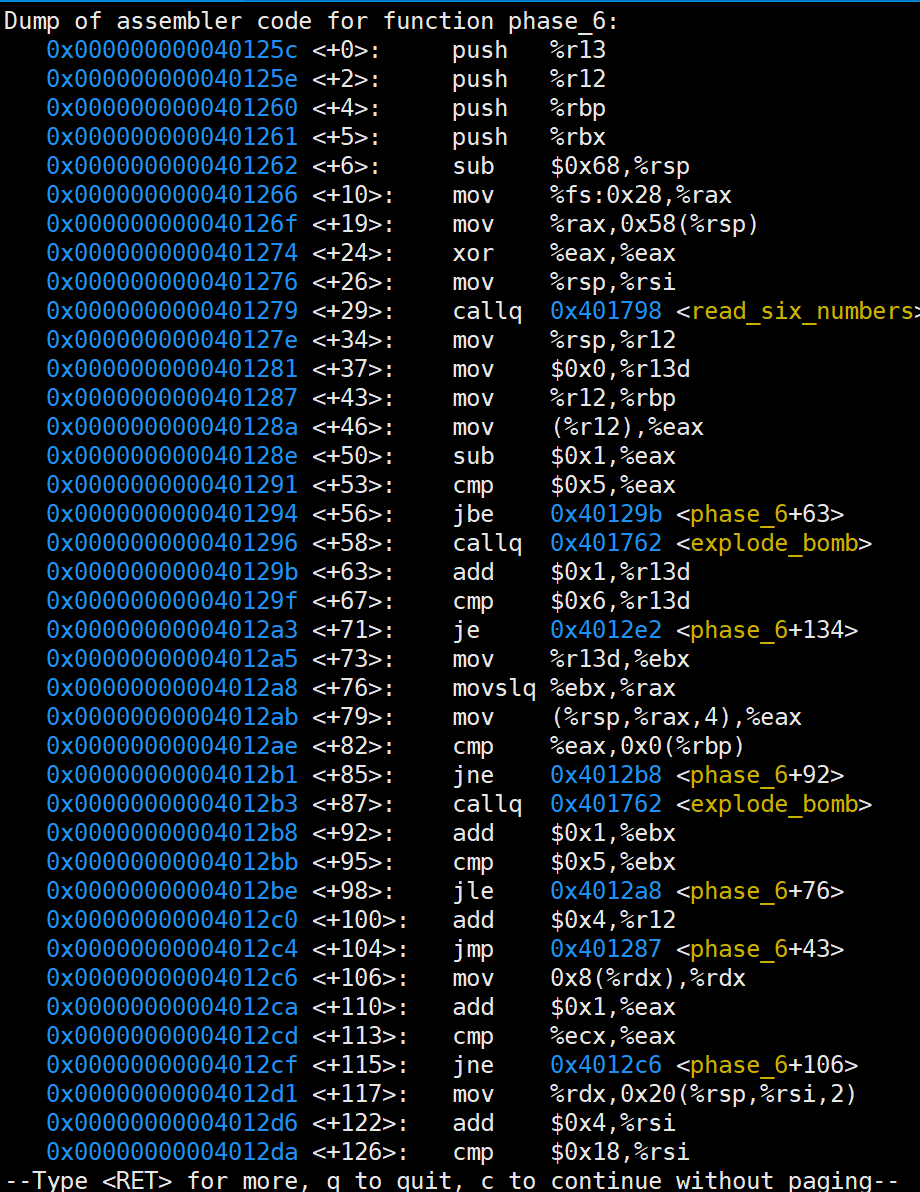
由+48可以得知第一个%rax存储的是输入的第一个值，即5，又因为在循环过程中将得到的%eax一次次累加%ecx得值为115，由+103行可知，只有%ecx内值等于输入的第二个数的时候能拆除炸弹，所以第二个数为115.所以输入的两个数为5 115.

将函数运行到爆炸函数，输入5 115 炸弹5成功被拆除：



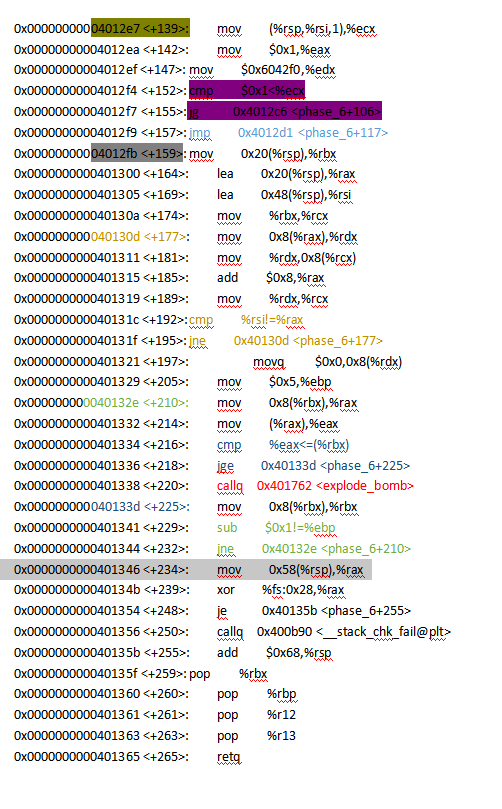
**炸弹6：**

查看phase\_6的源码：

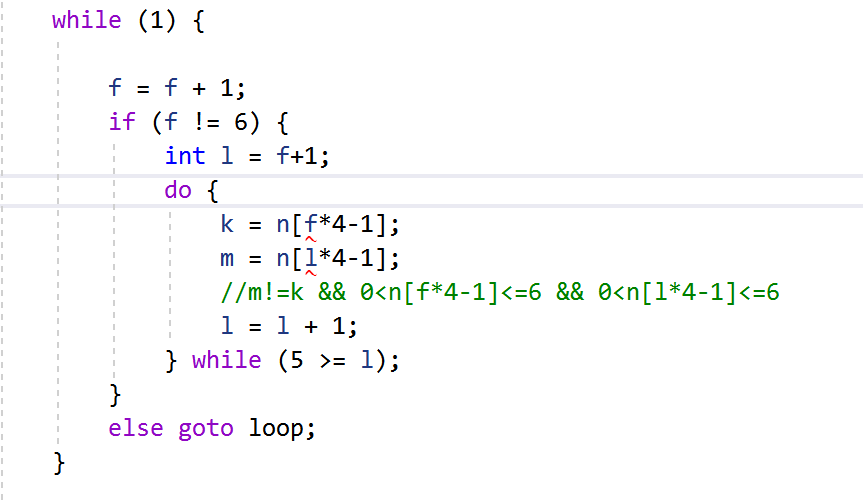


从+29的函数read\_six\_number可以判断出，输入值为6个数字

该函数循环较多，通过颜色标记来看每一次循环的开始与结束：

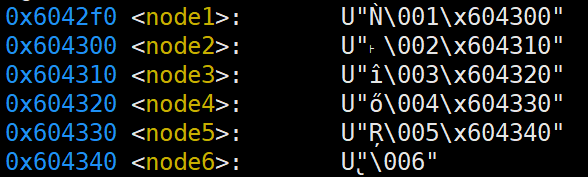


选取+43行到+104行作为一个循环，翻译为c语言代码：

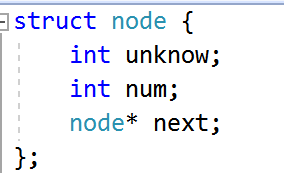


该循环的目的是判断输入的六个数是否都大于0小于等6，且各不相等（即6个数为1，2，3，4，5，6的排列组合）

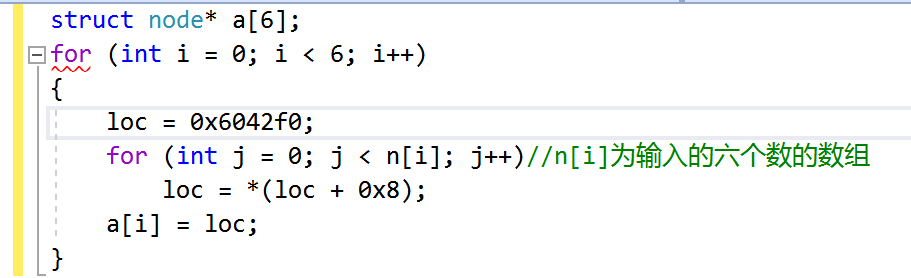
再选取+106到+152作为一个循环,观察到+147行有地址赋予：将0x6042f0赋给%edx，查看0x6042f0:



显然该地址为一个链表的开始位置，且该链表储存了未知数据+编号+下一个数据的位置。

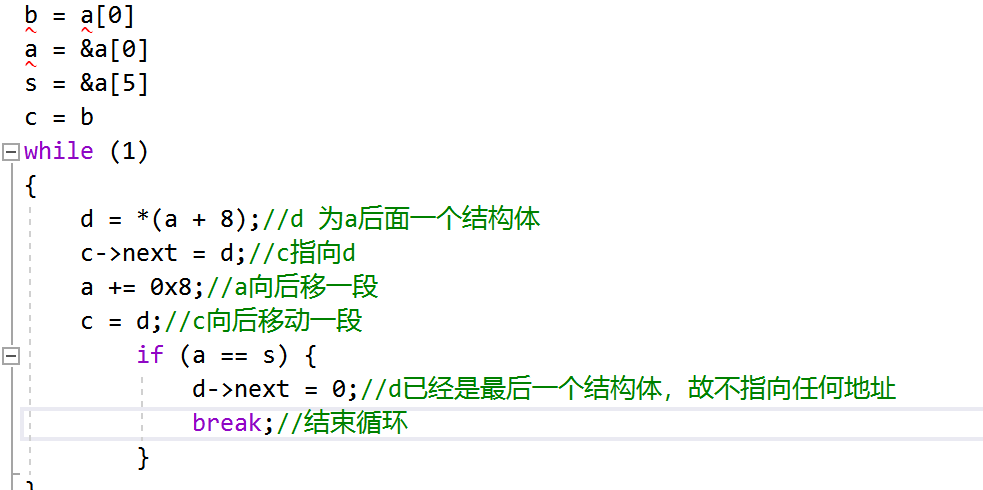
该结构体为：

翻译为c语言代码



即：将6个结构体按次序排列到以0x6042f0为开头的地址上去.

将+159到+197作为一个整体分析,翻译为c语言代码(作用如标注所示）为：

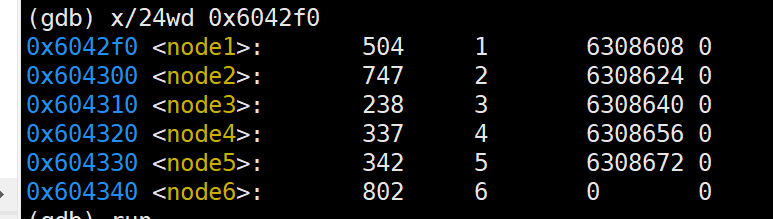


即：将结构体按顺序串成链表。

将+205到+232看作一段:

+205让%rax保存%rbx的下一个指针，+214行让%eax保存%rax中的unknow值，+216行对前后的unknow值进行比较，如果前者小于后者则爆炸，如果不爆炸则循环移动指针，直到最后一个指针被比较完，由此可知6个数是按unknow的降序排列的。

查看0x6042f0，根据unknow大小对1 2 3 4 5 6进行排序：



可知顺序为：6 2 1 5 4 3

运行到爆炸函数输入该顺序，第六个炸弹被拆除，至此，所有炸弹都被拆除。

`I[OK0DHMC{PF0AL4[GWX7W

五、总结体会

1.一个程序跳转过多时可以利用颜色对每一个跳转出入口进行标注，不然容易看错

2.要多关注call指令前的地址，通过查看这些指令可以判断输入值类型以及数量

3.要注意函数返回值通常储存在%eax中，在phase\_4中我就判断错了返回值，误以为返回的是%ecx，导致我无法求出正确答案。

4.输入值常常储存在栈%rsp中，通过Imm（%rax）来访问栈的不同地址。

5.可以在explode\_bomb处设置断点来防止爆炸

6.\017对应ascii码不是17是15，可以自己写一个函数来打印，不能被表面现象所迷惑！

7.汇编代码中要敢于设置变量，虽然这些变量在后期可以整合在一起，但在前期分开写有利于进行理解每一步操作的目的。

8.对于部分较为复杂的汇编代码，可以先翻译为c语言再进行分析，较为容易

9.要充分理解直接调用寄存器和先（寄存器）再调用的区别，一个是传地址，一个是传地址内的值。以及mov和lea的区别。

10.x/s可以直接查看某地址的储存的值,同时x也有很多不同的用法，可以根据个人要求使用不同的x指令。

六、诚信声明（不签扣10分）

在完成本次实验过程中，我曾分别与以下各位同学就以下方面做过交流：

无

此外，我还参考了以下资料：

1. <https://www.cnblogs.com/findumars/p/4126893.html>（查询ASCII码
2. https://blog.csdn.net/u013985241/article/details/89169007?spm=1001.2101.3001.6661.1&utm\_medium=distribute.pc\_relevant\_t0.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1.essearch\_pc\_relevant&depth\_1-utm\_source=distribute.pc\_relevant\_t0.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1.essearch\_pc\_relevant（如何使用x指令）

在我提交的程序中，还在对应的位置以注释形式记录了具体的参考内容。

我独立完成了本次实验除以上方面之外的所有工作，包括分析、设计、编码、调试与测试。

我清楚地知道，从以上方面获得的信息在一定程度上降低了实验的难度，可能影响起评分。

我从未使用他人代码，不管是原封不动地复制，还是经过某些等价转换。

我未曾也不会向同一课程（包括此后各届）的同学复制或公开我这份程序的代码，我有义务妥善保管好它们。

我编写这个程序无意于破坏或妨碍任何计算机系统的正常运行。

我清楚地知道，以上情况均为本课程纪律所禁止，若违反，对应的实验成绩将按照0分计。



（签名）

