

**实验报告**



**题目： 拆解二进制炸弹**

**班 级：2022211313**

**学 号：2022211363**

**姓 名：谢牧航**

**学 院：计算机学院**

**2023年 10 月27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. Windows PowerShell（10.120.11.12）
3. Linux
4. Objdump命令反汇编
5. GDB调试工具

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。

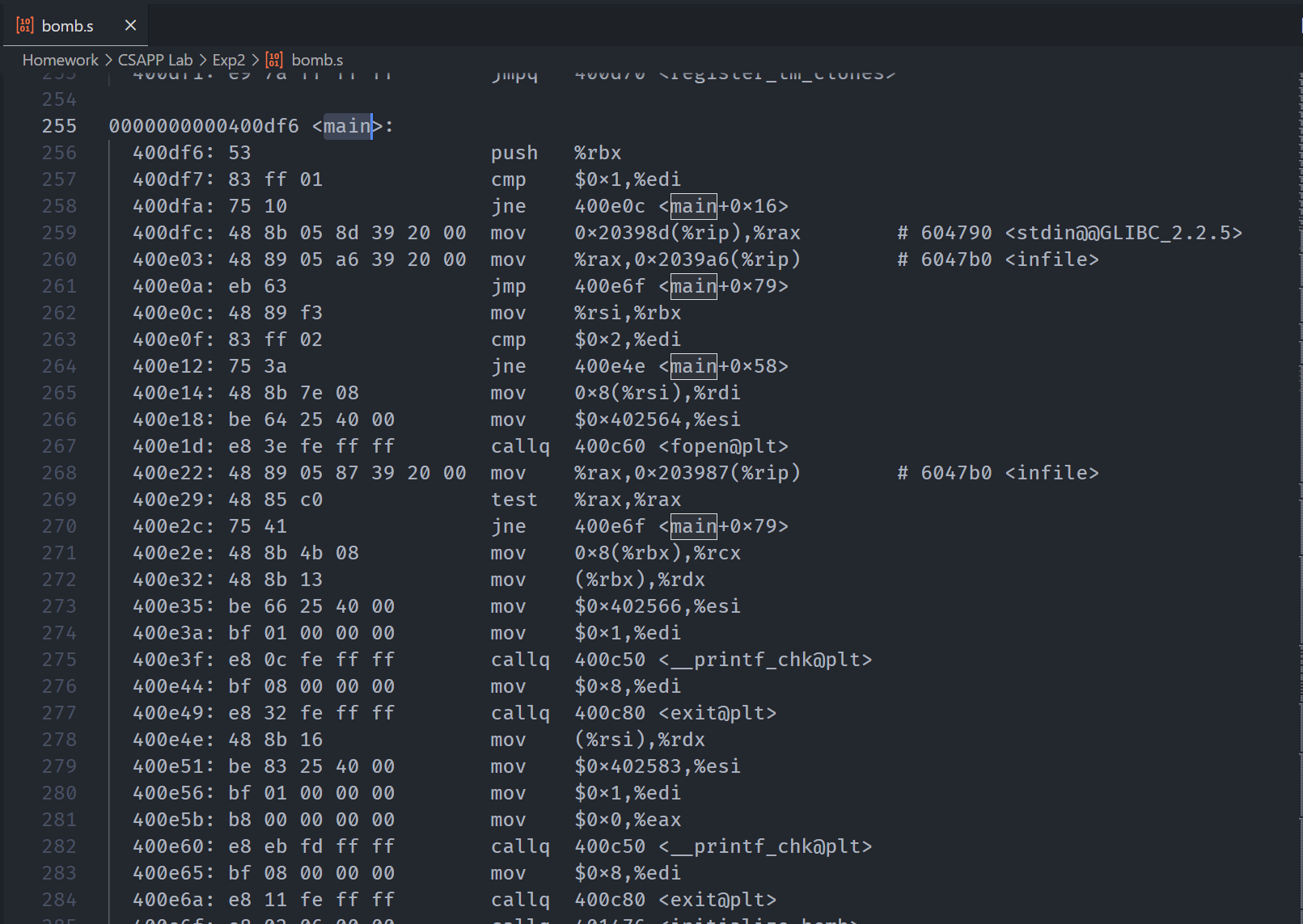
四、实验步骤及实验分析

建议按照：准备工作、阶段1、阶段2、…等来组织内容

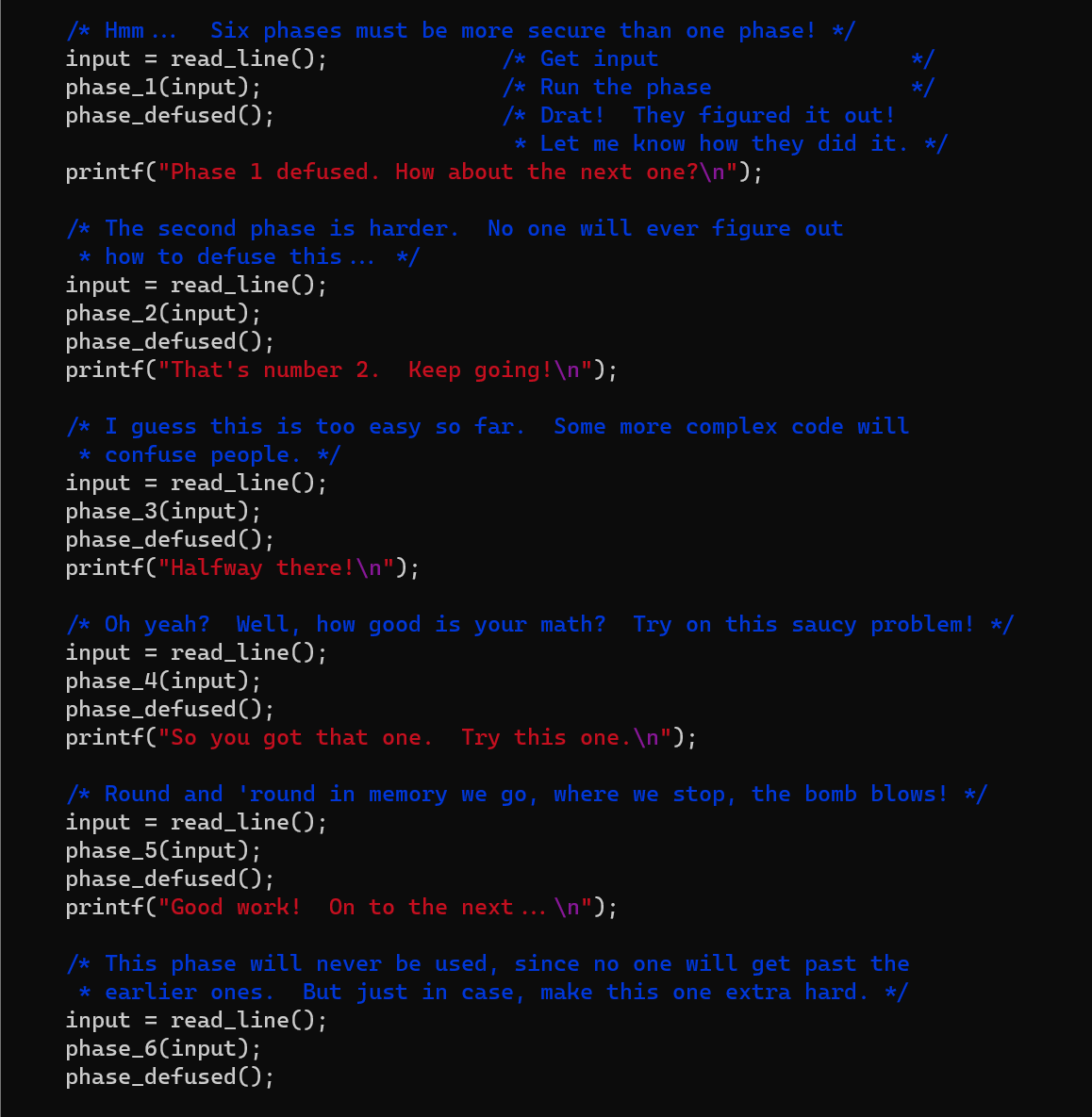
各阶段需要有操作步骤、运行截图、分析过程的内容

## 准备工作

解压 bomb.tar，使用objdump生成二进制文件bomb的反汇编文件。

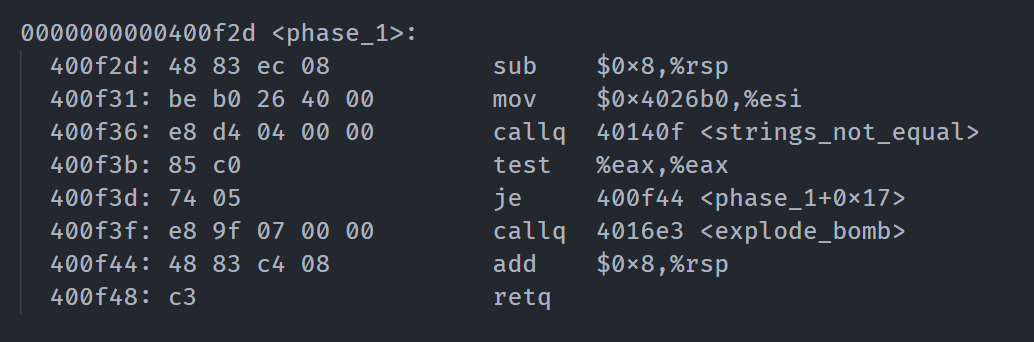


阅读bomb.c，得程序的main函数结构。



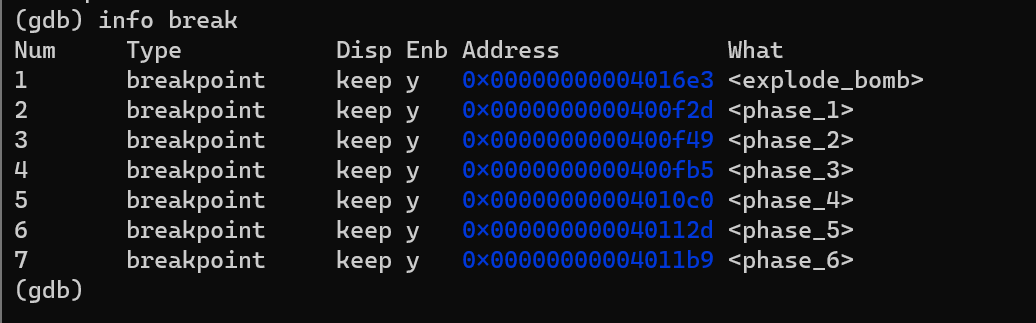
通过phase\_i函数来判断输入答案字符串的对错。还有一个phase\_defused函数作用未知。

查看phase\_1函数汇编代码：



发现通过函数explode\_bomb控制炸弹爆炸。

用gdb运行bomb程序，在六个phase函数和explode\_bomb函数处设置断点，防止炸弹爆炸。



注：前置知识：

使用examine命令（缩写为x)可以查看指定内存地址的值。语法如下：

x(hex) 按十六进制格式显示变量。

d(decimal) 按十进制格式显示变量。

u(unsigned decimal) 按十进制格式显示无符号整型。

o(octal) 按八进制格式显示变量。

t(binary) 按二进制格式显示变量。

a(address) 按十六进制格式显示变量。

c(char) 按字符格式显示变量。

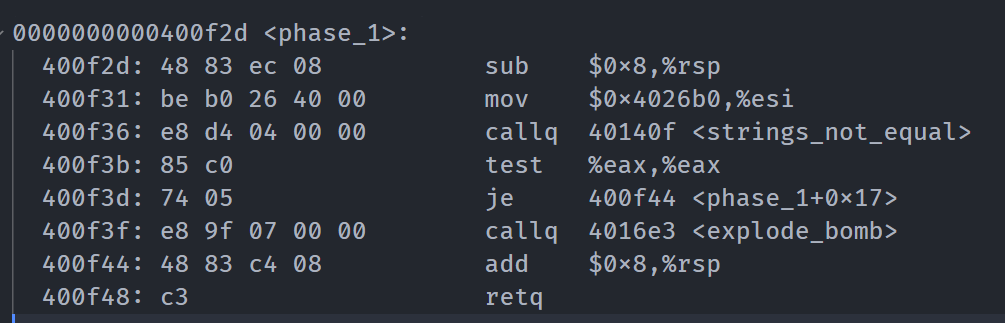
f(float) 按浮点数格式显示变量

i(instruction) 按指令方式显示指定地址处的内容

s(string) 按字符串方式显示变量

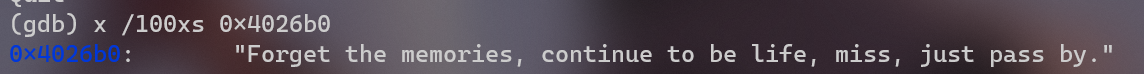
## 阶段一

阅读phase\_1汇编代码如下。



string\_not\_equal若不为0则会执行爆炸函数。而由第二行得到比较的字符串保存在0x4026b0中。

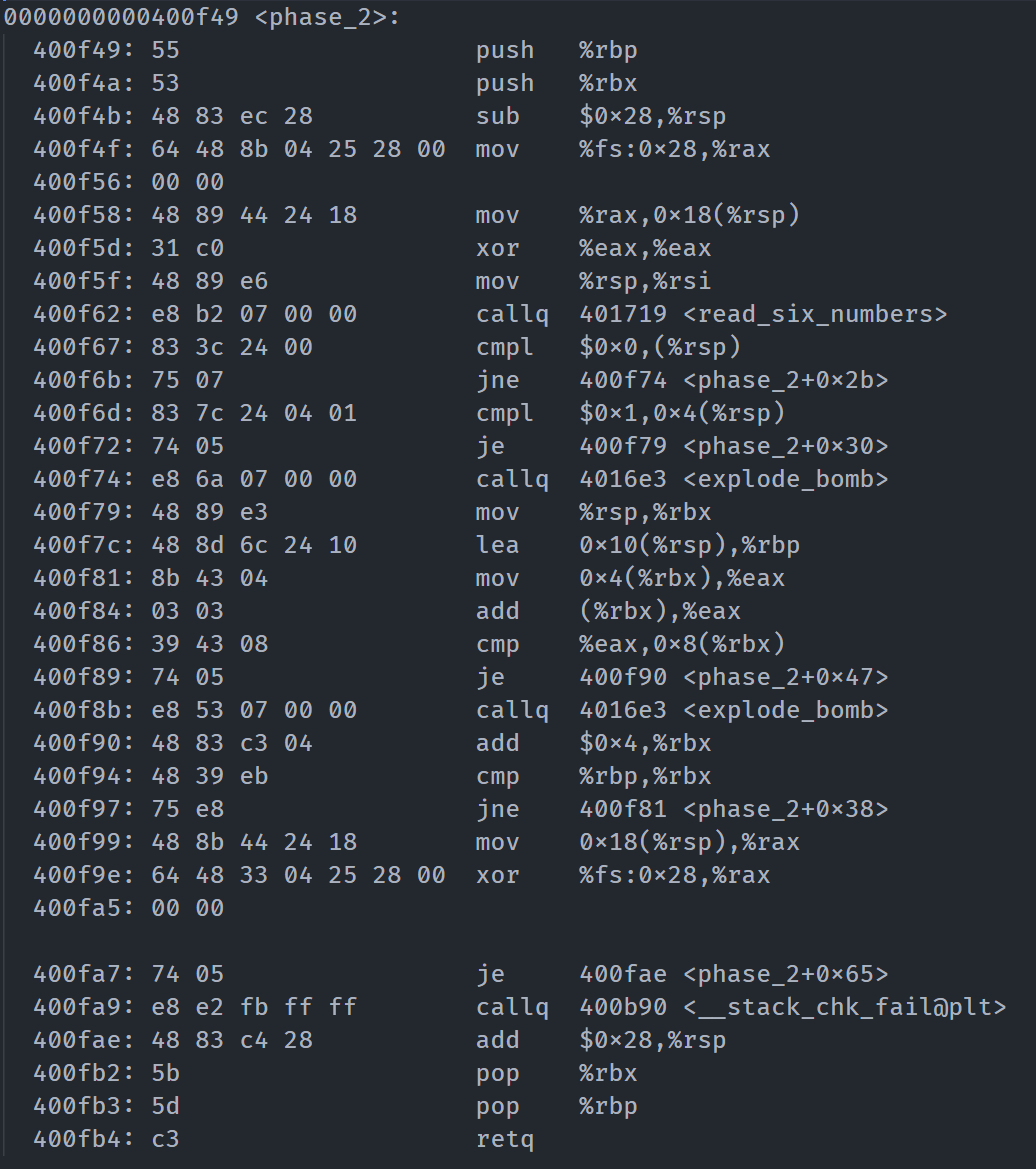
使用指令查看地址0x4026b0处的值。



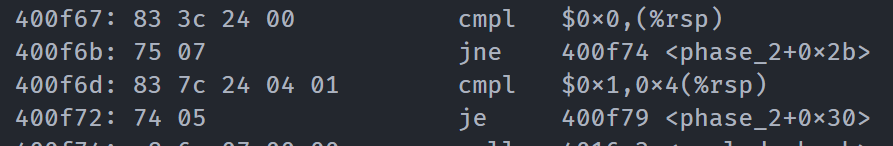
输入答案Forget the memories, continue to be life, miss, just pass by.阶段一结束。

## 阶段二

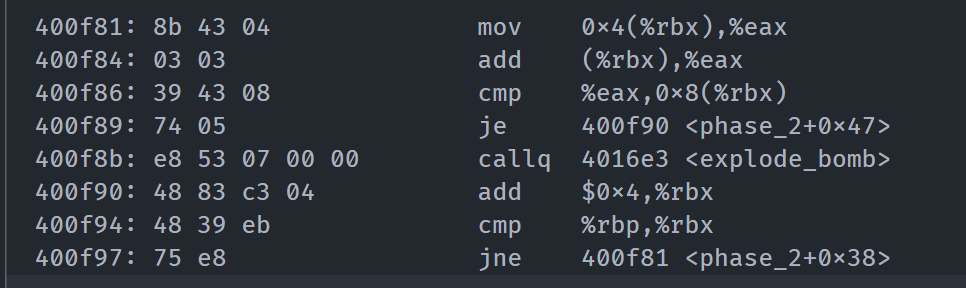
阅读phase\_2汇编代码如下。



由read\_six\_numbers知读六个整数进来。数组首地址存在%rsp中。



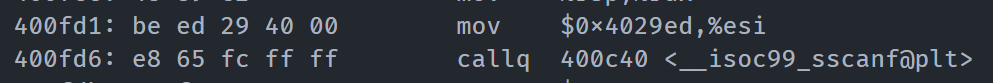
表示如果v[0]为0或v[1]不为1会爆炸，则v[0] == 0, v[1] == 1。



这部分是一个循环。前三句表示%eax存的是v[i]+v[i + 1]，而比较的对象是v[i + 2]。就是说v[i + 2]==v[i]+v[i+1]才不会爆炸。结构类似斐波那契数列。由v[1]==0,v[2]==1，得六个数为0 1 1 2 3 5。

输入0 1 1 2 3 5，阶段二结束。

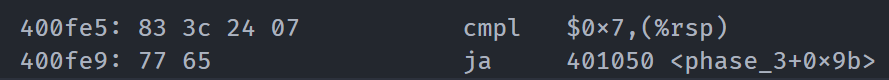
## 阶段三



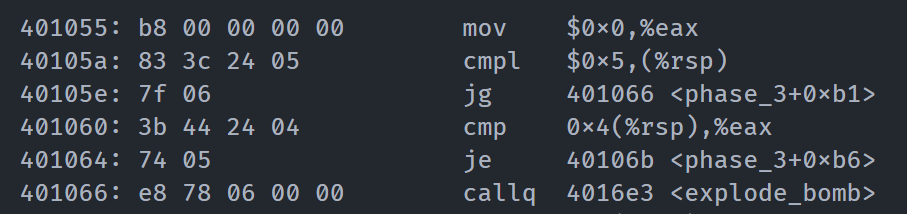
首先读入，查看格式字符串：



可知输入为两个整数。我们不妨称为v1和v2。

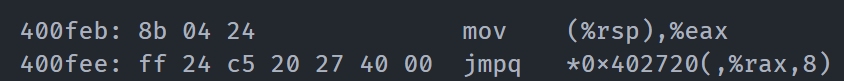


v1在无符号下大于7会爆炸。这里我们选择让他为0。



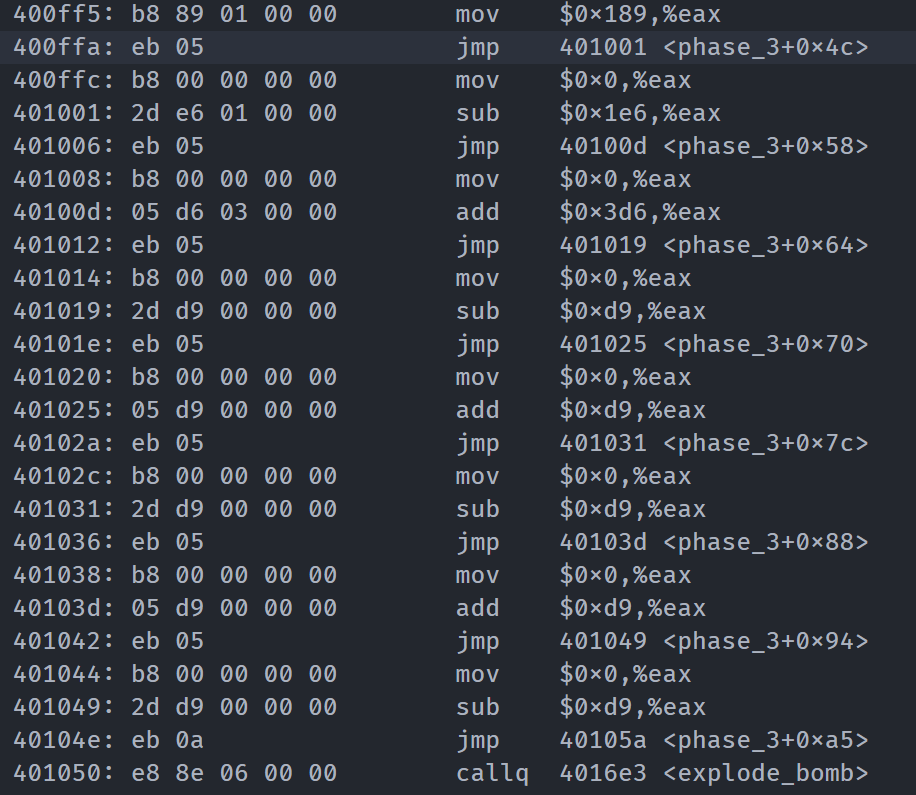
若v1大于5或v2不等于%eax的值会发生爆炸。

观察中间的代码，类似一个switch结构。我们看输入0会如何跳转。



v1为0，跳转的位置即为\*0x402720，查看此地址处的值：



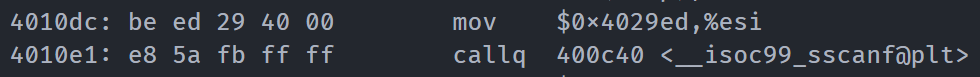


接下来会分别跳转0x400ff5，0x401001，0x40100d，0x401019，0x401025，0x401031，0x40103d，0x401049。根据计算得，%rax的变化分别（十进制下）为393，-93，889，672，889，672，889，672，889，672。

所以v1==0，v2==672。输入0 672，第三阶段结束。

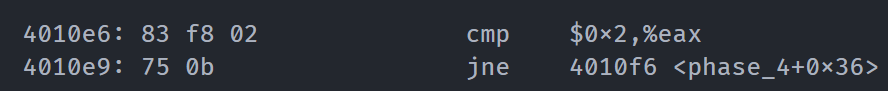
## 阶段四

查看sscanf格式化字符串

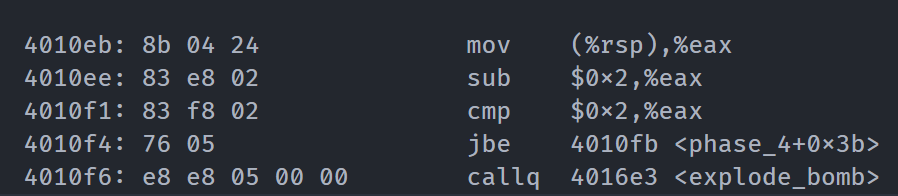




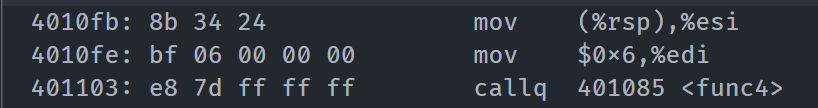
可知输入为两个整数。称为v1，v2。



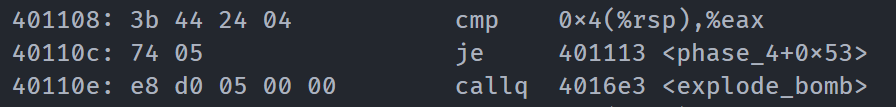
输入数不为两个会爆炸。



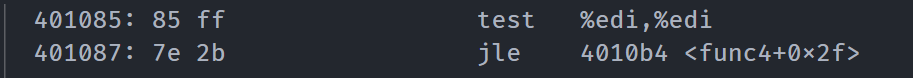
无符号下v2-2>2会导致爆炸，所以我们令v2=3。

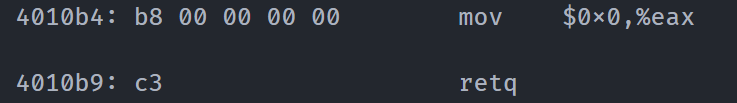


调用func4(6, v2)

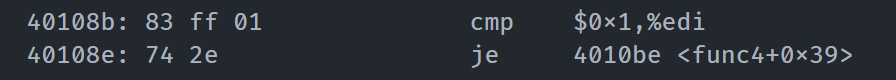


如果返回值和v1不等则会发生爆炸。所以f接下来我们分析func4函数。



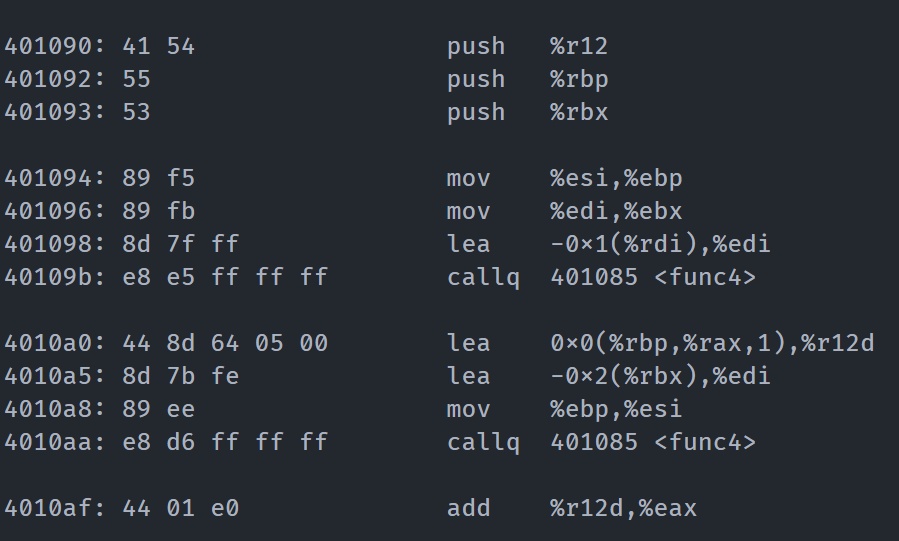


如果a1<=0，return 0。





a1=1，也会直接返回0，不为0，则进入下面的部分。



这是一个递归函数。第一个调用func4(a1-1, a2)。

%r12d中存了a2和第一次调用的返回值的和。然后再调用func4(a1-2,a2)。所有加起来。

写成C语言伪代码结构大概如下：

return a2 + func4(a1 - 1, a2) + func4(a1 - 2, a2);

列表如下

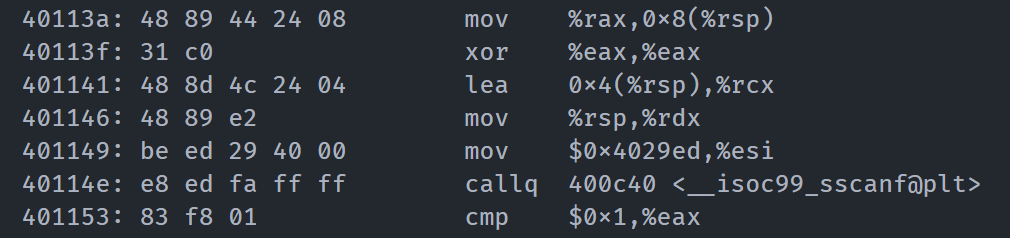
0   1   2   3   4   5    6

0   v2  2v2 4v2 7v2 12v2 20v2

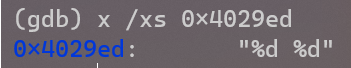
v2=3，则v1=60。

输入60 3，第四阶段结束。

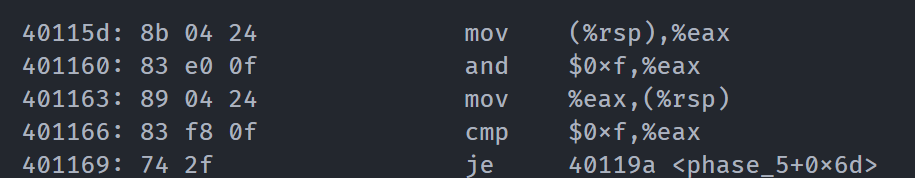
## 阶段五



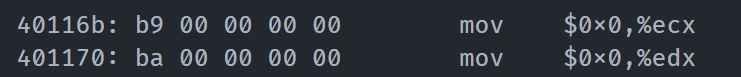
查看sscanf格式化字符串



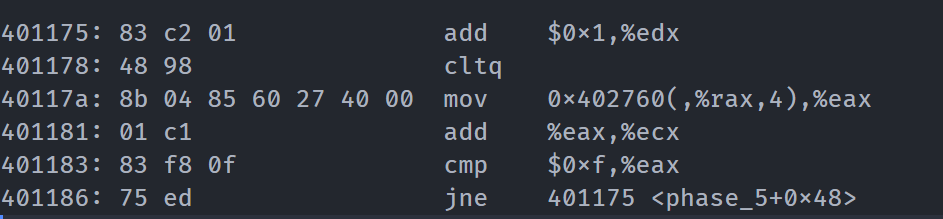
输入两个整数v1，v2。如果输入个数小于等于1，则发生爆炸。



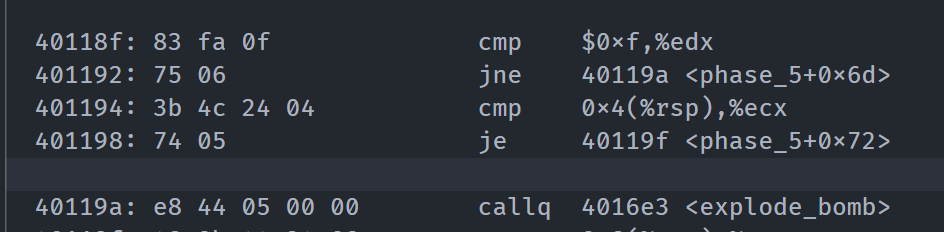
令v3=v1&15，若v3为15，则发生爆炸。则v1一定是一个小于15的非负数。



新建两个变量v4，v5，初始化为0，分别存在%rcx和%rdx寄存器中。

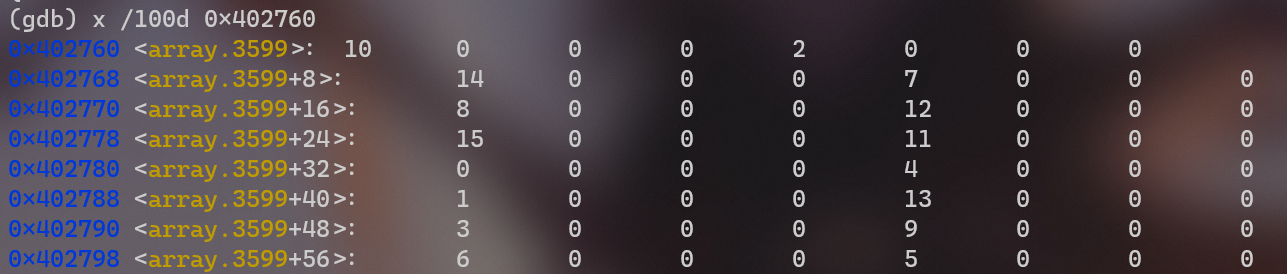


这是一个循环，当v3=15的时候会停止循环。循环内部，v5每次加一，v3每次变成0x402760 + 4 \* v3。则0x402760类似一个数组array，每次v3=array[v3]。然后v4+=v3。

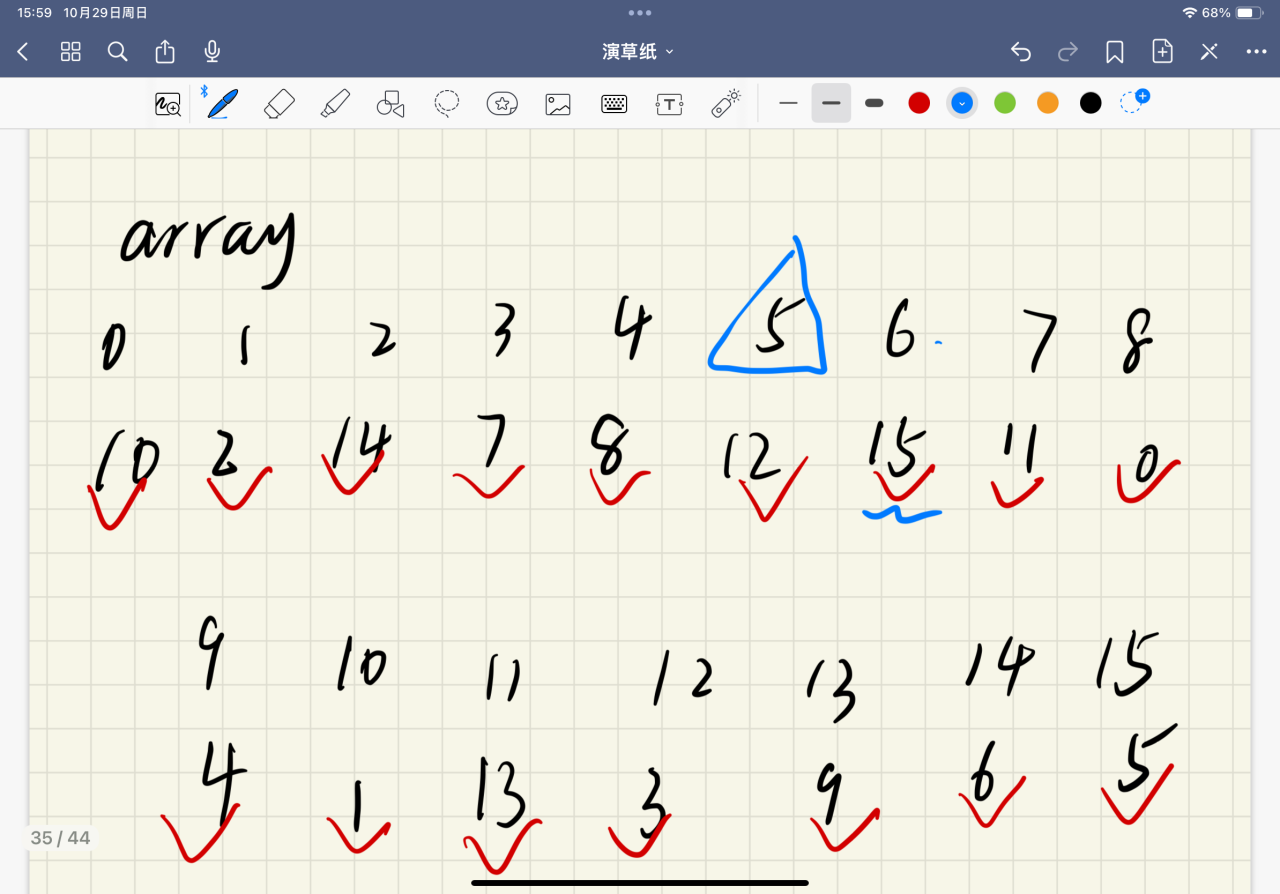


最后v3!=14或v2!=v4则发生爆炸。所以我们需要让v3最后为15,v2最后为v4。

首先查看数组0x402760的值



画图如下：



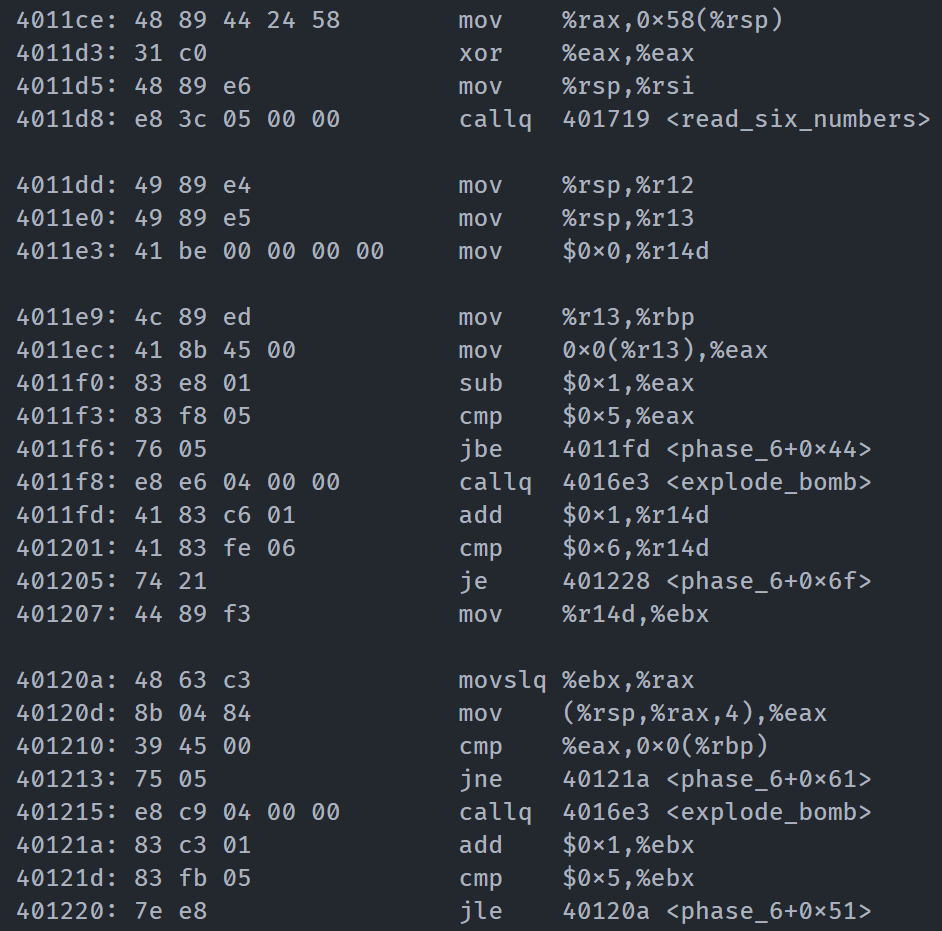
从最后v3=15反推15次循环回去，则v3最开始为5，而下面加的过程中只有5不会被加到，其他所有数加起来为115。

所以输入5 115，阶段五结束。

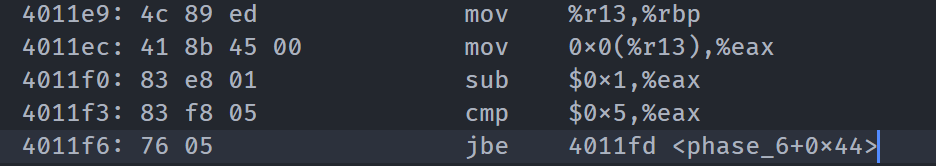
## 阶段六

阶段六代码较长，我们将分成几个部分来解析。

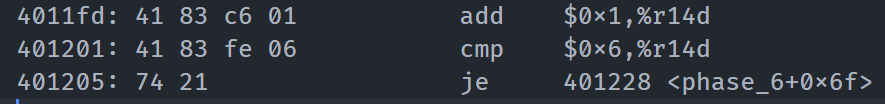
#### 输入并验证



首先输入六个数字，然后是一个双重循环。



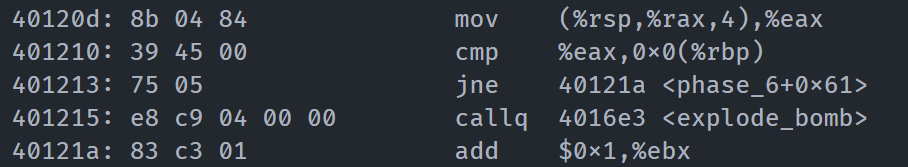
输入的每个数-1后大于5会爆炸，也就是说每个数都要小于6。



检查个数到达6之后跳出循环。也就是输入的六个数。

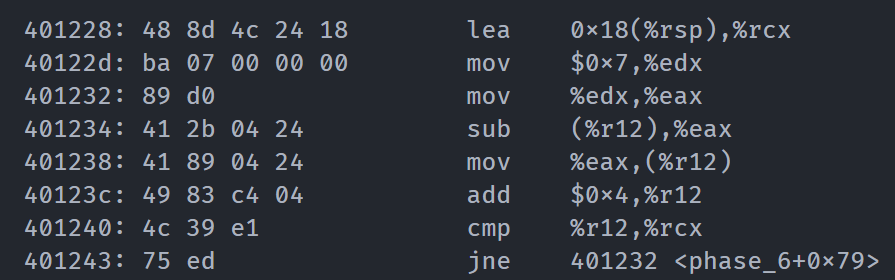


这个是这个循环的内层循环。



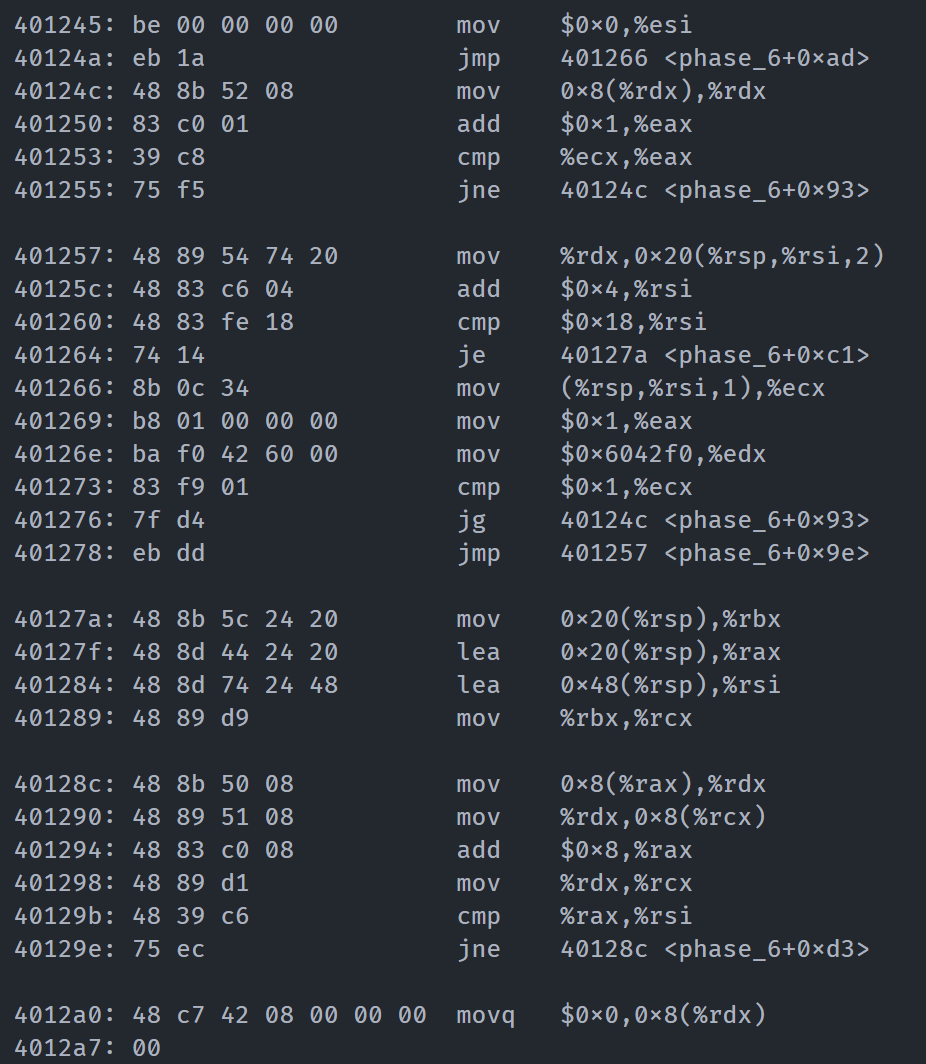
这段话实际上讲的是把当前数和后面所有数相比较，如果出现相同的就会爆炸。所以这段的意思是需要输入的六个数各不相同。

#### 转换输入

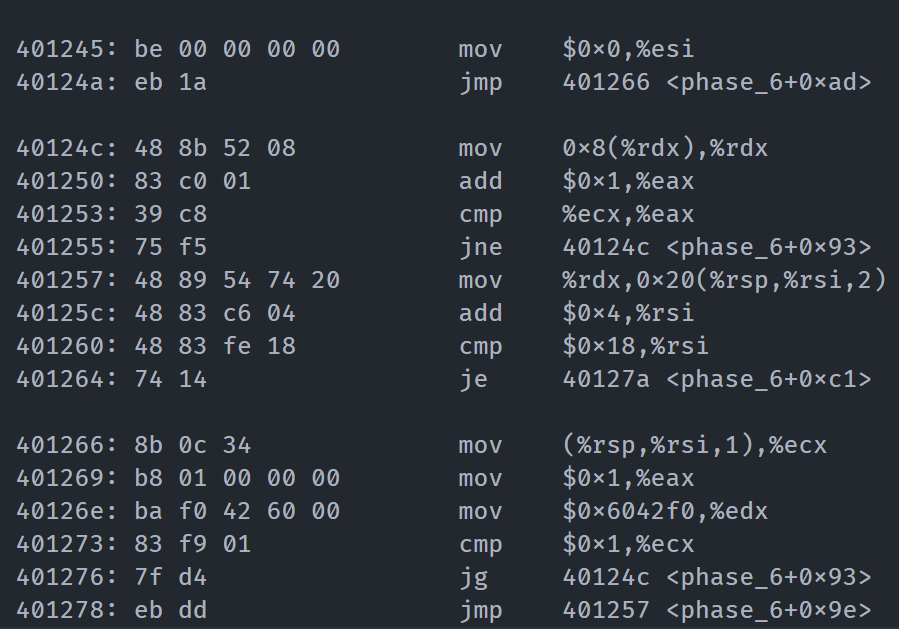


这一段的意思是把每个输入的数x都变成了7-x，直到数组结束。所以我们要注意输入的时候，应该要再反过来输入7-需要的值。

#### 重排链表



这部分分为两个循环，我们先来看第一个循环。



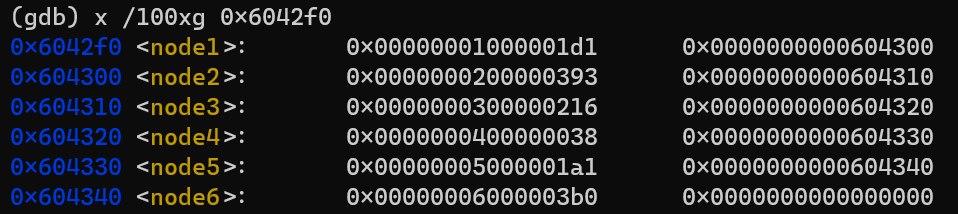
可以发现这个循环中还嵌套着另一个循环。

首先将%ecx归零，然后跳转到0x401266。把0x1赋给%eax，$0x6042f0的值赋给%edx。我们取出%rsp+%rsi的值（当前数组读到的位置）赋给%ecx，如果大于1跳转0x40124c，否则跳转0x401257。

假设小于等于1，我们跳转0x401257，我们把%rdx赋给%rsp+0x20+2\*%rsi，下一行%rsi+0x4，所以我们可以推测这是一个以%rsp+0x20为起始的数组，%rsi作为索引每次+0x4，所以数组单个元素值大小为0x8。一直到数组遍历到末尾结束。

假设大于1，我们跳转0x40124c，%rdx指向%rdx+0x8，%eax每次加一，直到%eax和%ecx相等。

首先查看0x6042f0处的值：



发现这个形成了一个链表结构，每个结点第一个数是当前的值，第二个数是下一个结点的地址。所以

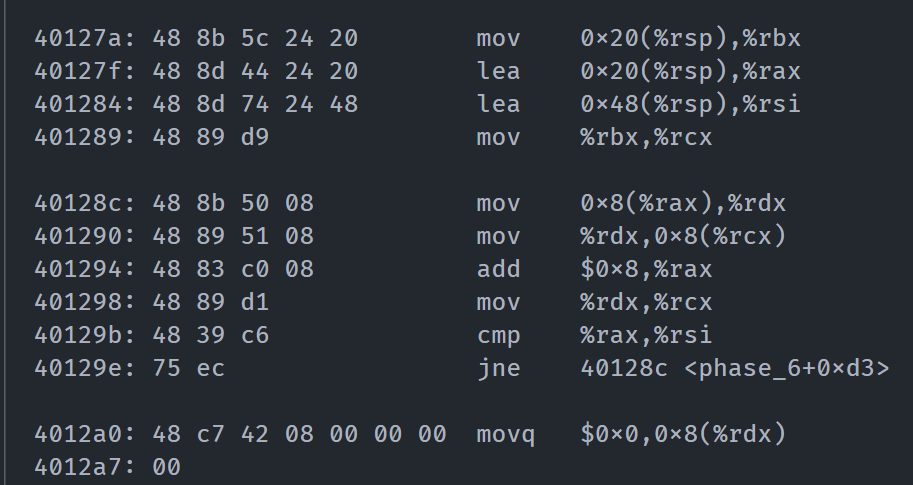


这句话就类似C语言中的p=p->next。

总结这部分的作用：取读入的数组中的v[i]，找到链表中的第v[i]个元素，并把它放在新链表数组的第i个位置。

但是现在只是改变了结点的位置，改变之后链表被打乱了，所以我们的next也需要修改，而第二个循环就是用来改变next的。

观察第二个循环的汇编代码：



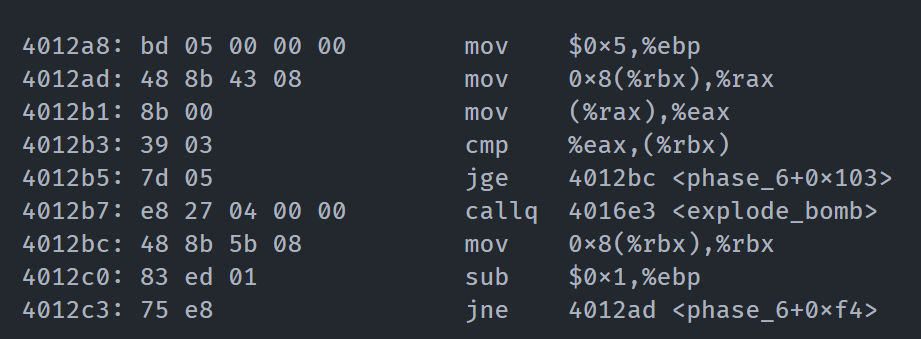
初始化阶段首先把数组首元素放在%rbx和%rcx中，首地址放在%rax中，数组末尾放在%rsi中，然后进入循环。

首先把当前数组下一位放在%rdx中，然后让$rcx+0x8赋值为%rdx，所以完成了p[i]->next=p[i+1]的操作，改变了next。然后%rax+0x8，%rdx赋给%rcx，即循环进入到下一位，直到数组末尾。最后，把链表最后元素的next设为0。

总结：这部分进行了一次链表的重构，取读入的数组中的v[i]，找到链表中的第v[i]个元素，并把它放在新链表的第i个位置。

#### 链表检验

最后这个循环进行对链表的检验。



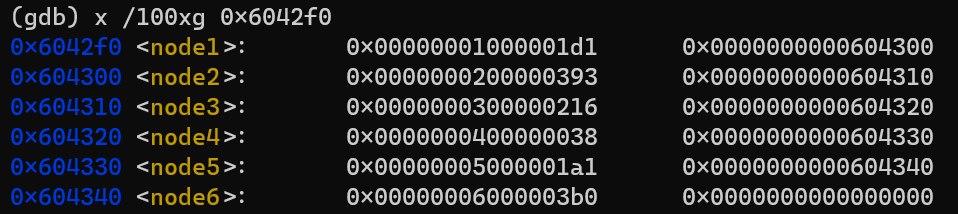
%ebp是循环次数控制变量，可知共循环五次。%rbx初始值为%rsp+0x20，则%rbx+0x8是当前的下一个元素的地址。然后再把这个地址的值赋给%eax。总的来说，就是把链表下一个结点的值赋给%eax。

下一行，将当前节点的值和下一个结点的值进行比较，如果小于，则发生爆炸。

总结：这部分要求新链表中的值是递减的，否则发生爆炸。

#### 反推答案

反观node链表

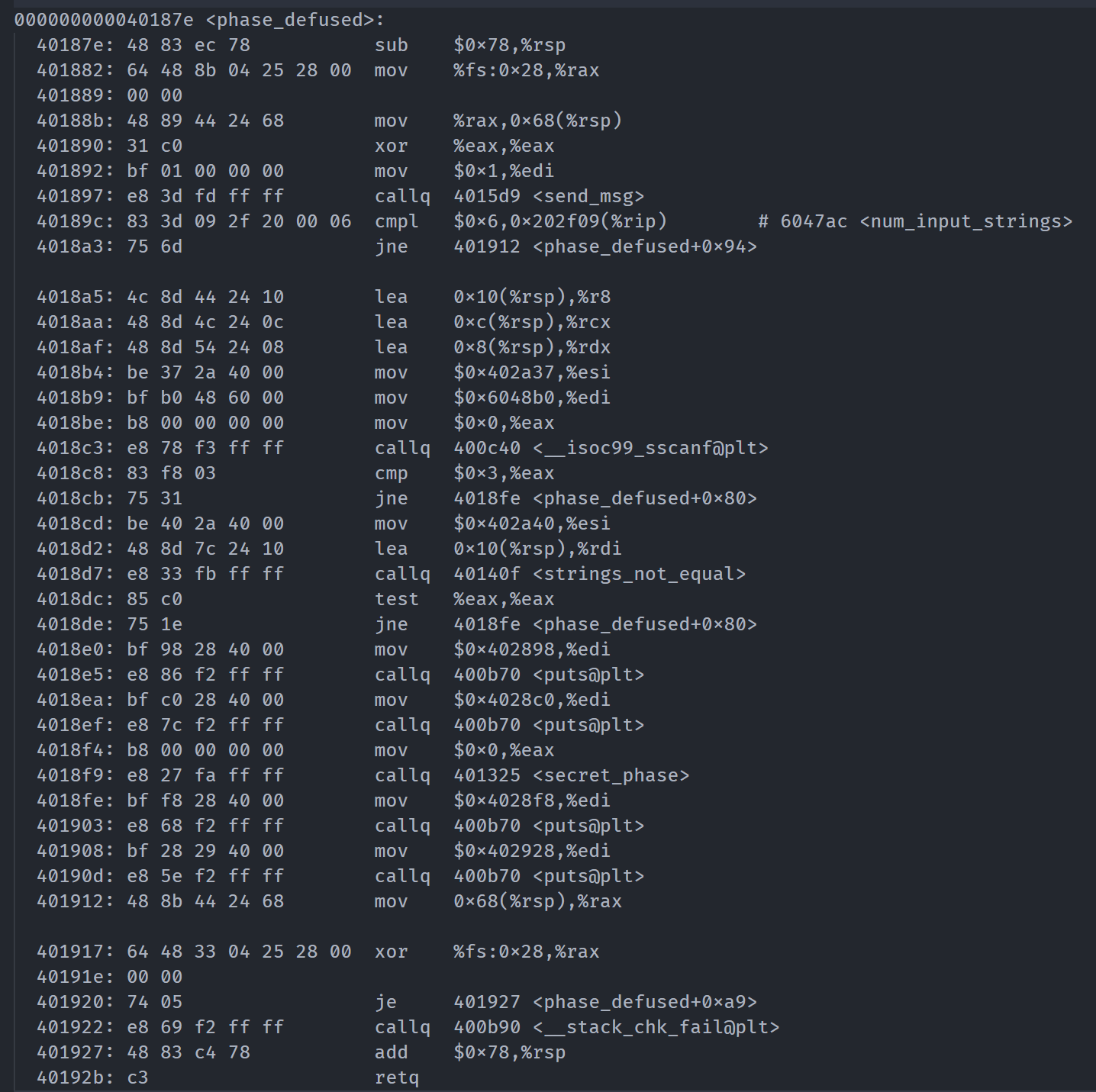


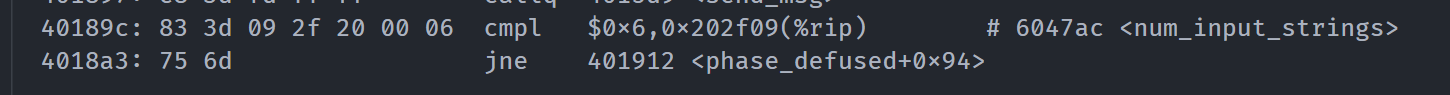
从大到小排列为6 2 3 1 5 4。

但是注意到我们开始的时候有一个x=7–x的操作，所以答案为1 5 4 6 2 3。阶段六结束

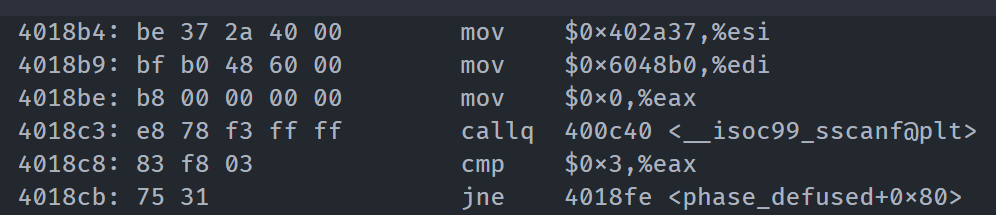
#### 阶段七

我们注意到phase\_defused函数还未用到，查看其汇编代码：





如果输入字符串不为6则直接返回。所以这个函数只有在我们解决了前六个阶段之后才会运行。



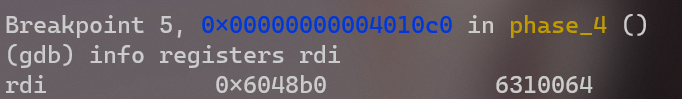
sscanf需要输入三个元素，查看格式化字符串：



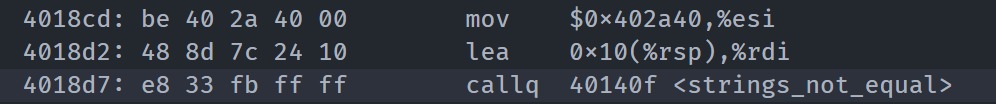
为两个数字和一个字符串。

但是不同的是其下面对%edi也进行了赋值（$0x6048b0），正常来说%edi中的值应该来自read\_line函数。说明这次没有再次从输入流中读入，而是使用了之前读入的字符串。

之前每次字符串都被保存在%rdi寄存器保存的地址中，我们检查每次%rdi中的内容。

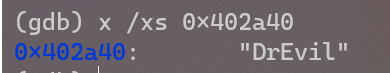


发现在进行phase\_4时，%rdi的值为0x6048b0。所以在这里我们需要再输入一个字符串。

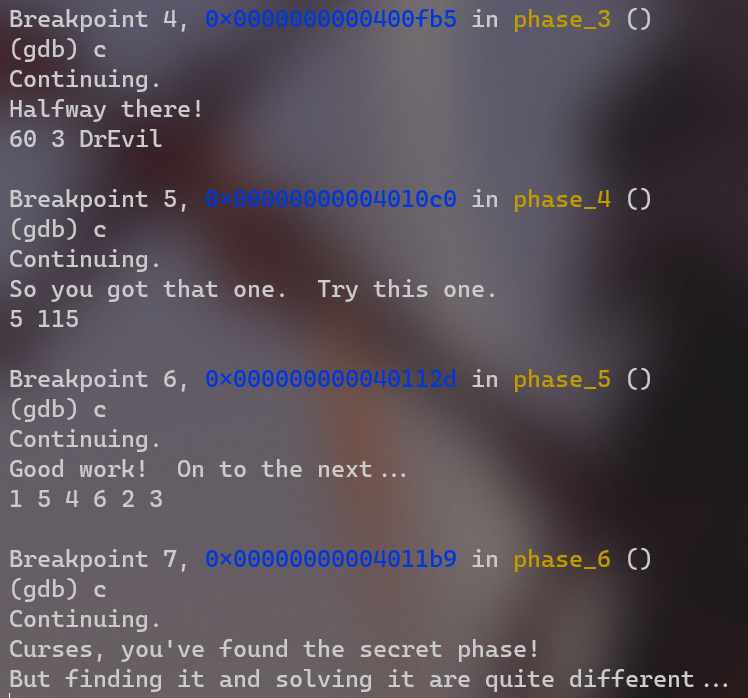


其实这里0x10(%rsp)的值也暗示了是第四次。

查看所需字符串



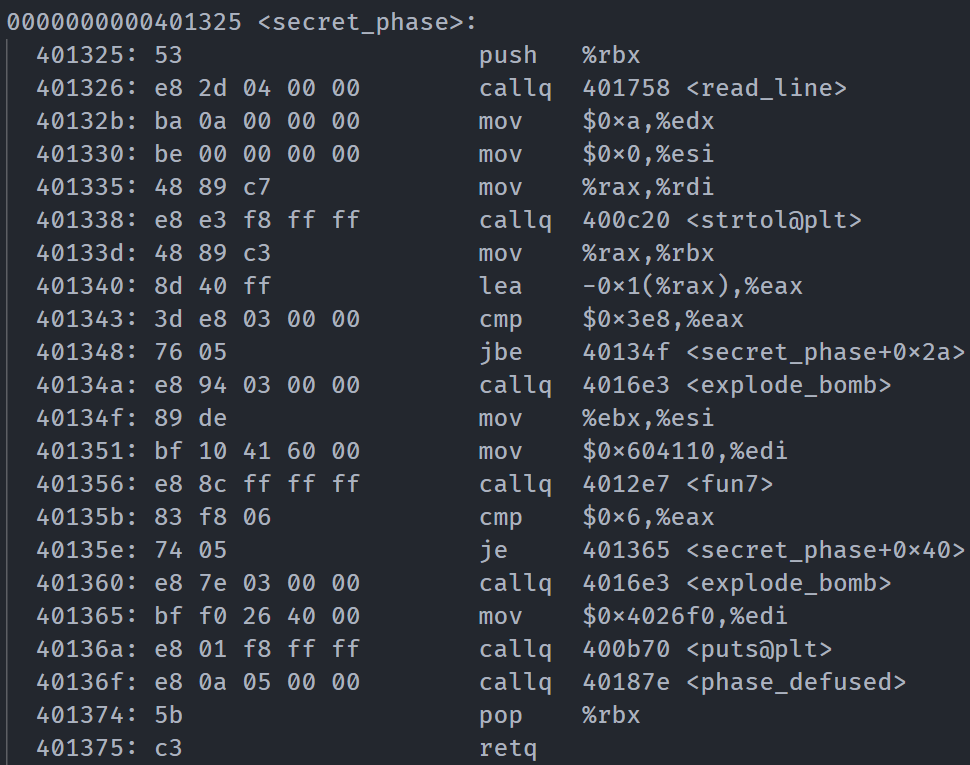
为DrEvil。所以我们在输入第四次时后面加一个DrEvil即可进入隐藏关。





进入隐藏关，接下来是secret\_phase函数。

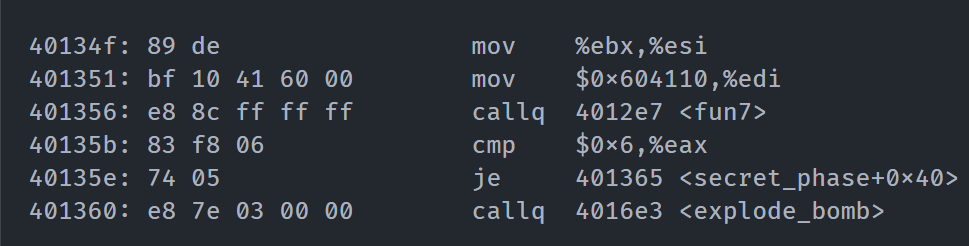
查看secret\_phase的汇编代码：



首先调用了一个strtol函数，这个函数用来把输入字符串转换为一个十进制数v1。

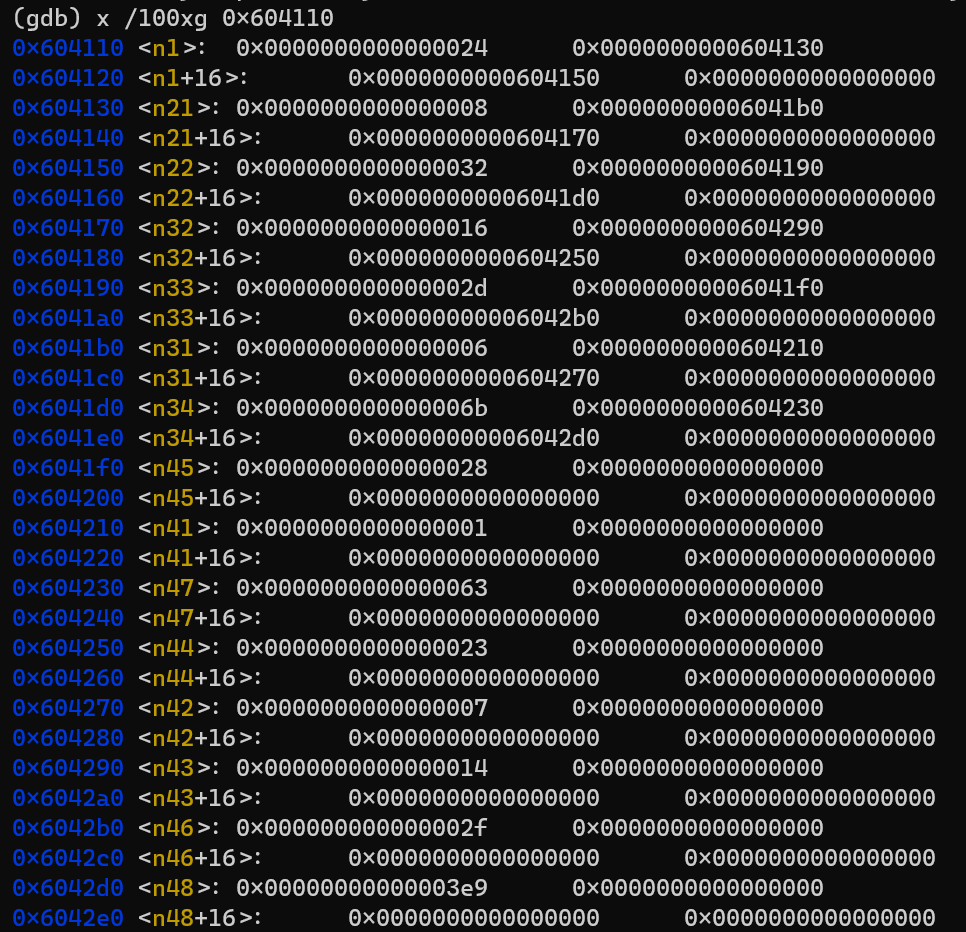


如果v1-1>0x3e8就会爆炸。所以v1<=1000。

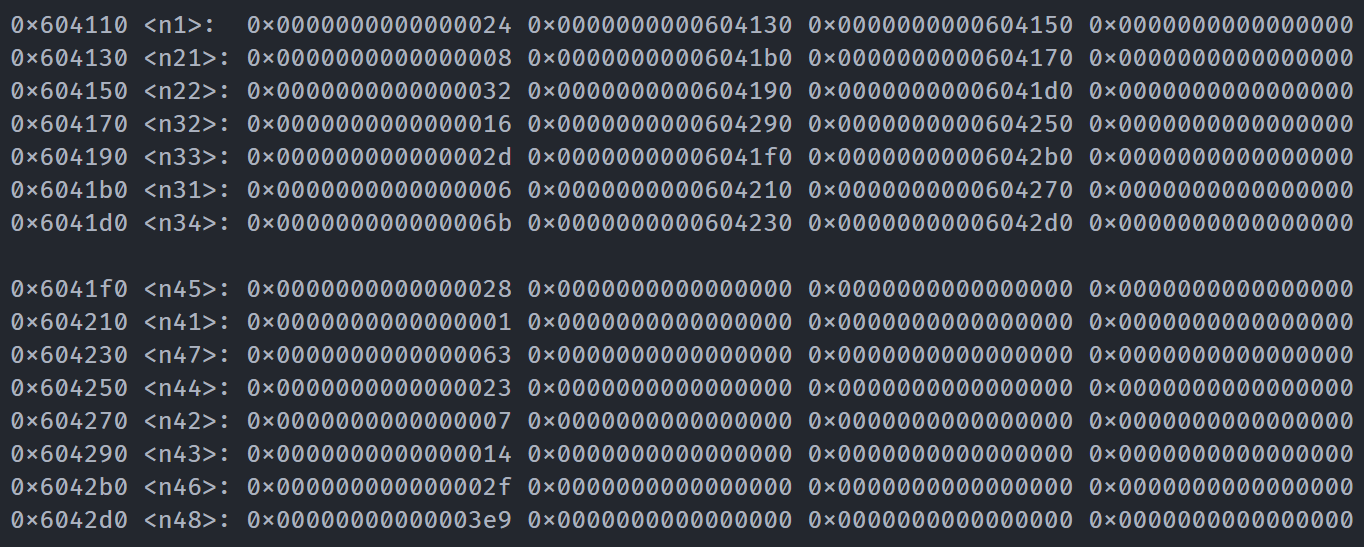


调用fun7，若返回值不为0x6则爆炸。第一个参数为$604110，第二个参数为v1。

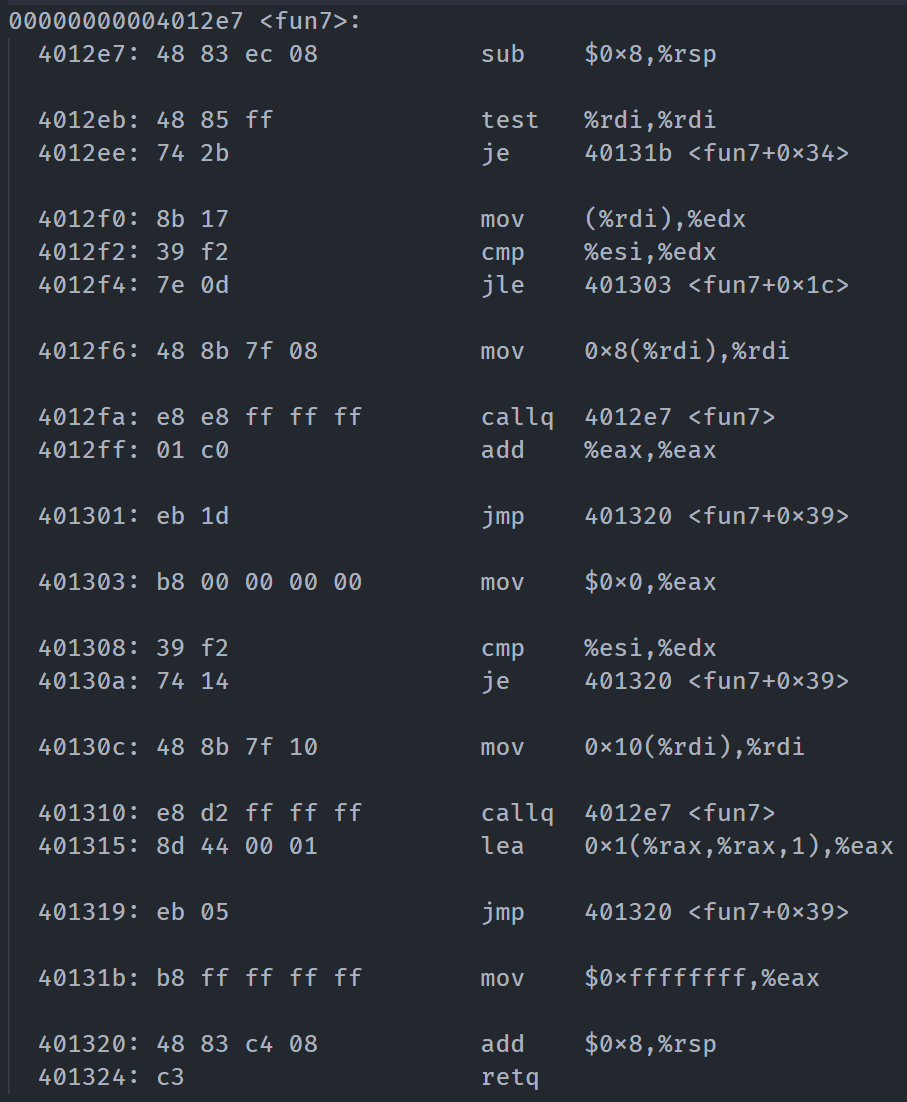
查看$604110处的值：



整理得到



含义未知，我们先看fun7函数的汇编代码。



发现对一个结点%rdi，接下来的操作由%rdi+0x8和%rdi+0x10，即上面的第二列和第三列。

到这里，我们可以猜测这是一个二叉树的结构，并发现这是一颗四层的满二叉树，数据结构如下：

struct Node {

    int val;

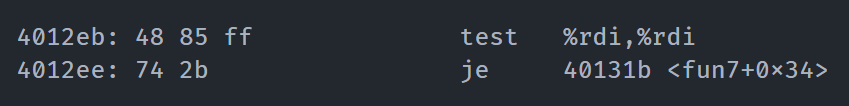
    int lson, rson;

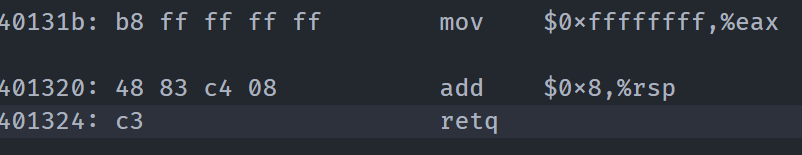
    int temp;

};

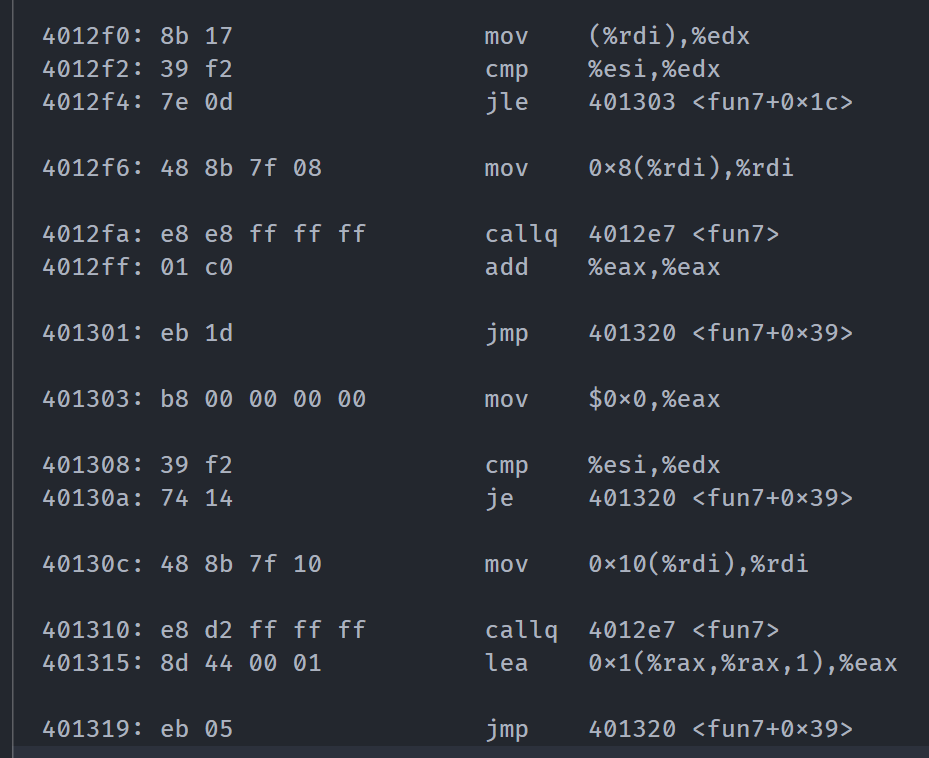
temp的值都设为0，这个元素的含义未知，猜测是为了使Node的大小为32byte而凑出来的变量。

先看fun7的结构：





如果当前结点为0（空），则返回-1。



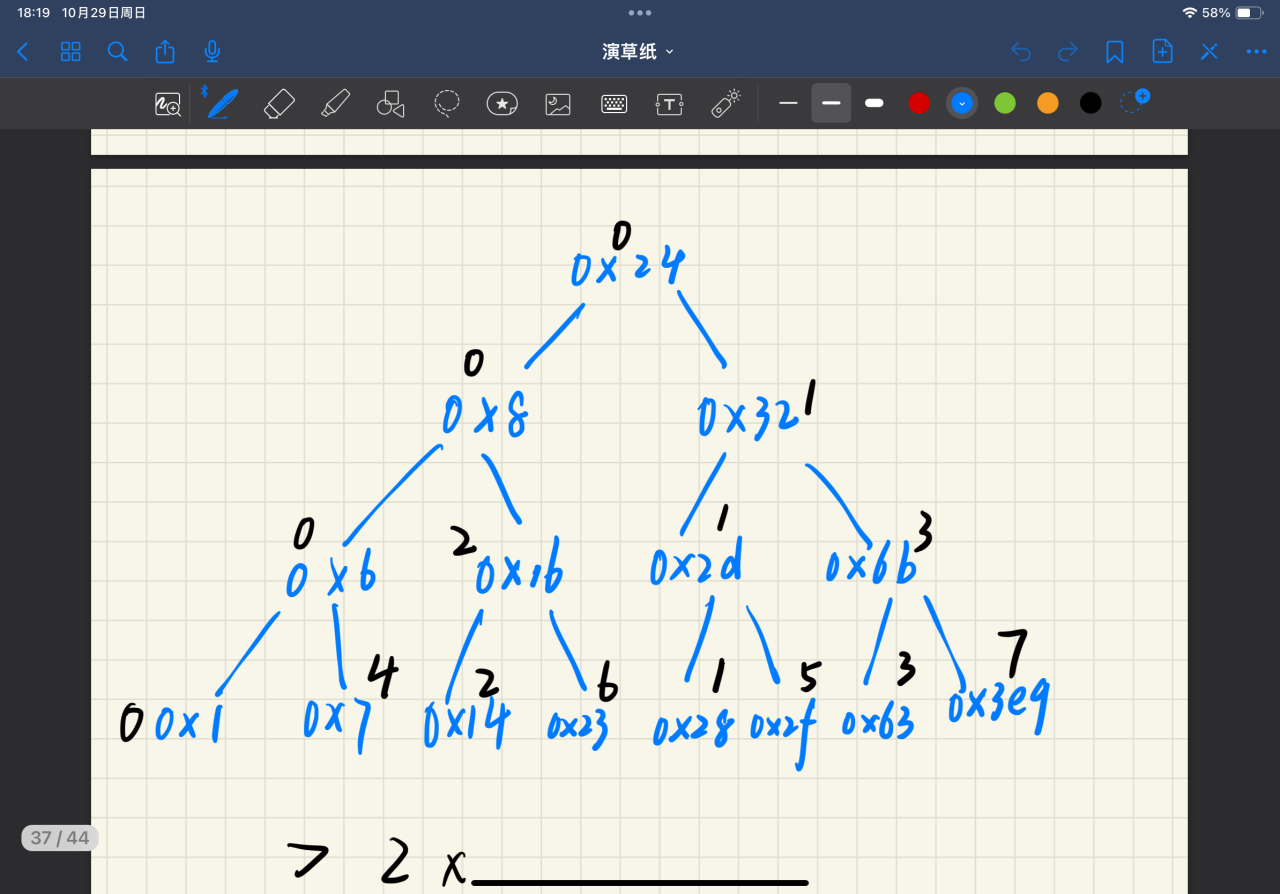
这部分为递归。设当前结点值为val，那么这部分是在说：

若val==v1，则返回0。

若val>v1，则返回2\*fun7(lson, v1)

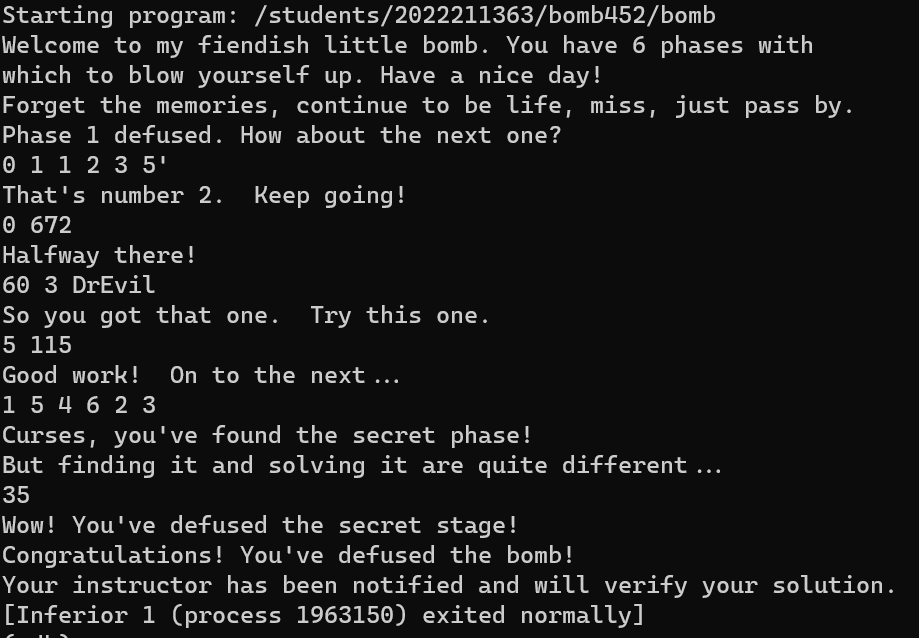
若val<v1，则返回2\*fun7(rson, v1)+1

画出之前的二叉树，我们发现其确实呈现出一棵二叉搜索树的形状：



把每个结点经过手动模拟后得到的fun7值标在了图上。可以看到返回值为6的是结点0x23，即35。阶段七结束。

## 通关截图



五、总结体会

总结心得（包括实验过程中遇到的问题、如何解决的、过关或挫败的感受、实验投入的时间和精力、意见和建议等）

本次实验做题时间约5小时，每解决一个阶段的炸弹，都会有一种过关斩将的成就感。特别是在解决了一些复杂阶段的炸弹后，这种感觉更是明显。通过这个实验，我对计算机系统、汇编语言和程序调试有了更深刻的理解。这种实践性很强的学习方式让我感到非常充实和有成就感。

阶段六确实难度很大，而且代码量很大。所以我把它分段查看，一个个读懂每一个循环的意思，最终终于通关。

隐藏关的难度主要是找到入口，一开始并没有头绪，后来发现本来应该通过read\_line得到的%rdi中的内容被直接赋值了，所以猜测是直接使用的之前某次已输入的字符串。

建议：可以在文档中提供一些详细的gdb命令所用到的参数，比如本次实验examine命令用到的各种参数如x,s,d,g等。

总之，Bomb Lab是一个非常有价值的实验，它不仅提高了我的编程和调试技能，也让我更加了解了计算机系统的运作方式。