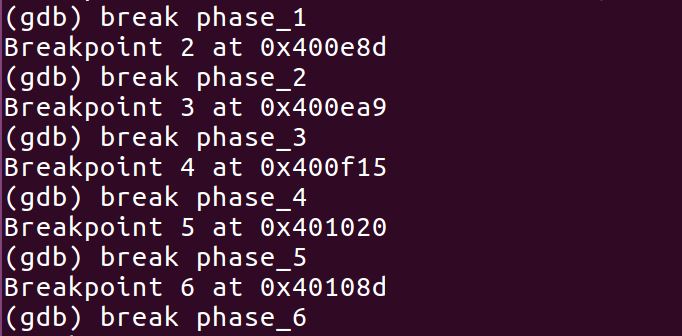
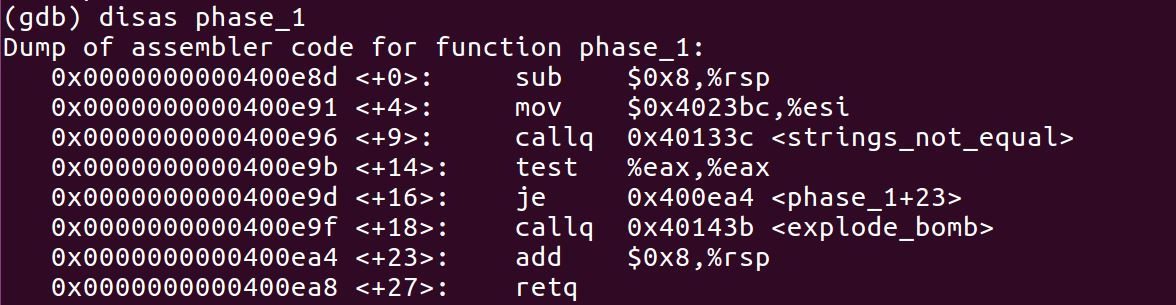
基本思路：在各个phase函数打断点，进入函数后stepi单步调试，查看寄存器。

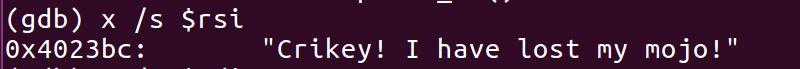


Phase 1：通过disas phase\_1查看汇编代码：



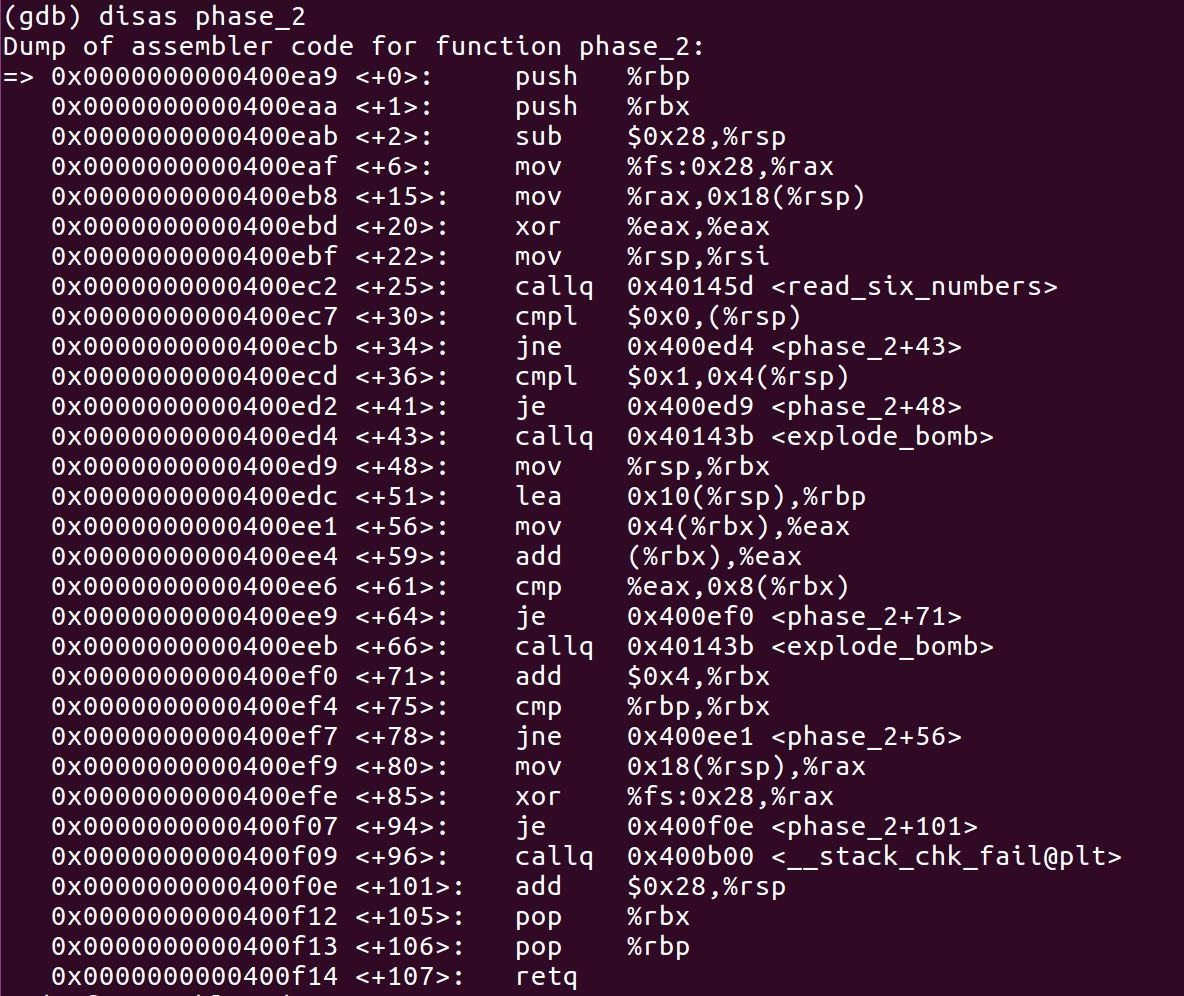
注意到用于跳转的判断条件为调用strings\_not\_equal函数的返回值。

且函数的一个参数存储在%esi寄存器中，则

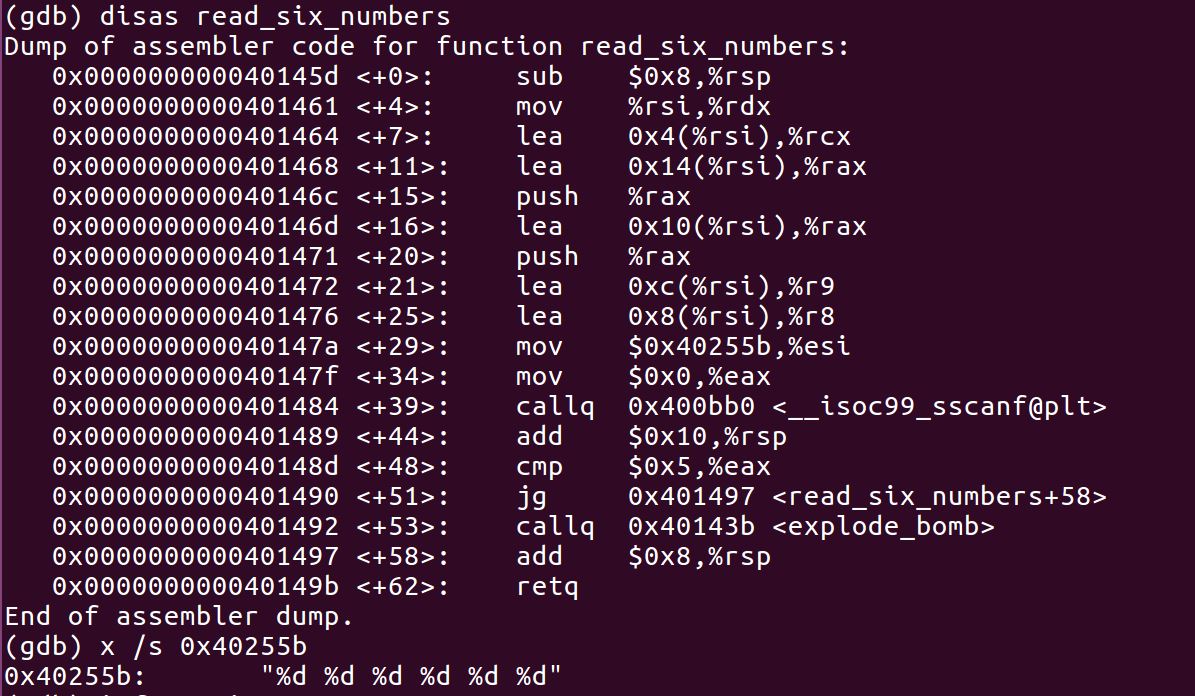


可见phase 1的答案是Crikey! I have lost my mojo! 经检验该输入可以通过。

Phase 2：通过disas phase\_2查看汇编代码：



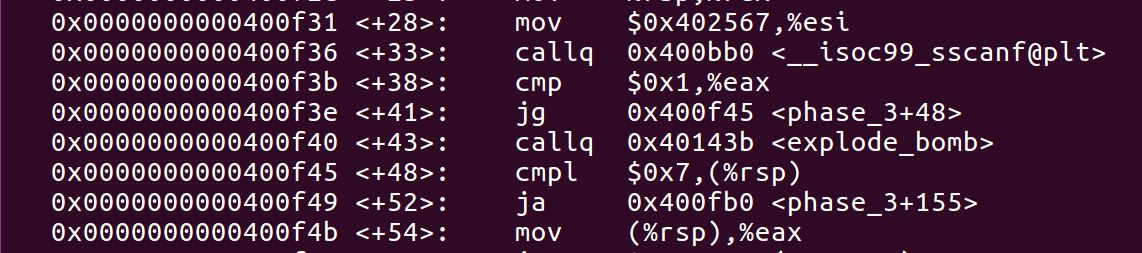
由函数名read\_six\_numbers及汇编代码，认为要输入6个整数。



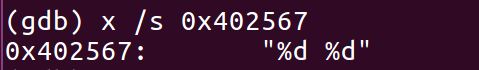
由cmpl $0x0,(%rsp)，cmpl $0x1,0x4(%rsp)及后续跳转判断第一个数字为0，第二个为1。

单步运行info registers查看寄存器，%rsp地址不断移动，每个地址存储的数据与前两个之和比较，不相等则跳转到explode\_bomb，判断斐波那契数列0 1 1 2 3 5，经检验符合。

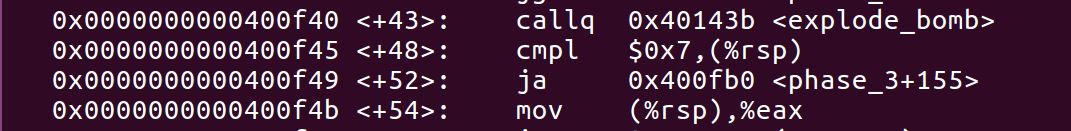
Phase 3：通过disas phase\_3查看汇编代码：

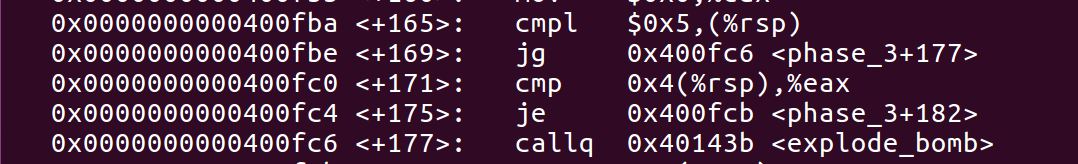


可见输入的数量需大于1，否则执行explode\_bomb。



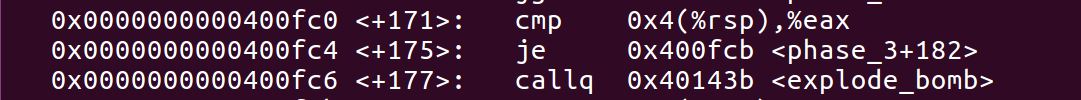
可见输入为两个整数。



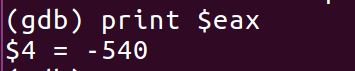


可见第一个数字低于或等于5（无符号），该数一系列操作后得到的值储存在%eax中，该值应等于第二个数，否则将跳转执行explode\_bomb。

观察中间代码可见语句之间的跳转与第一个数字有关，难以直接推出算式。



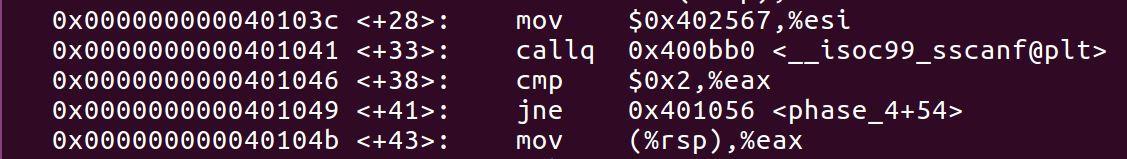
在\*(0x400fc0)处断点，尝试输入0和一随机数字，直接跳转到该断点，查看%eax。

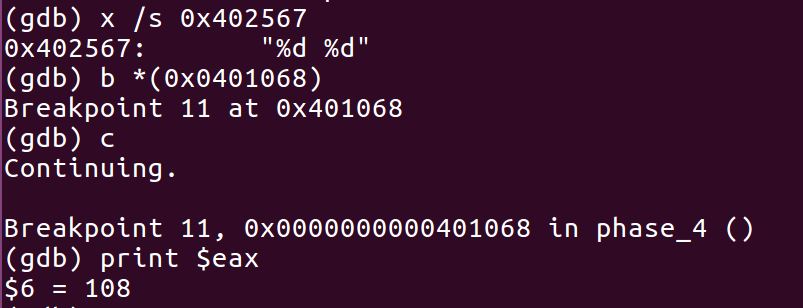


则判断0 -540为一组符合要求的输入，经检验通过。

同理可得另外四组解，分别为1 -591，2 -41，3 -914，4 0和5 -914。

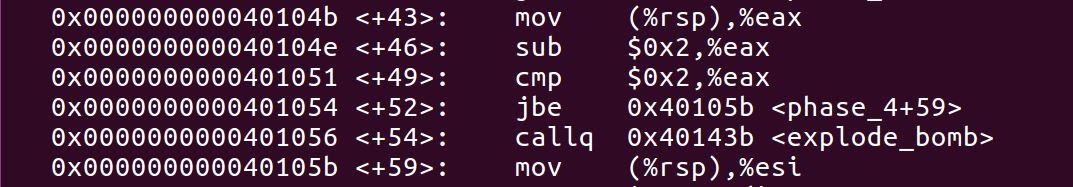
Phase 4：通过disas phase\_4查看汇编代码：



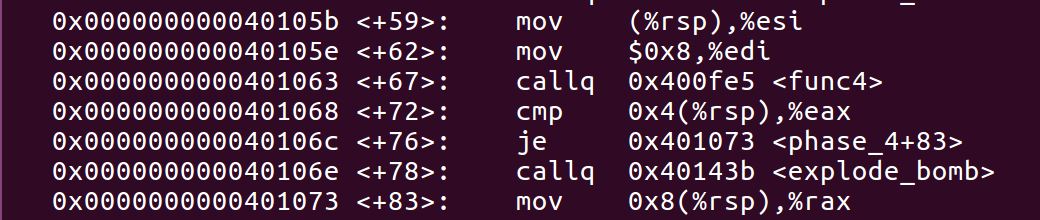


可见需要输入两个整数。

随机输入两个数字，运行到phase\_4，分别查看(%rsp)和0x4(%rsp)的值，发现第一个数字储存在0x4(%rsp)中而第二个数字储存在(%rsp)中。

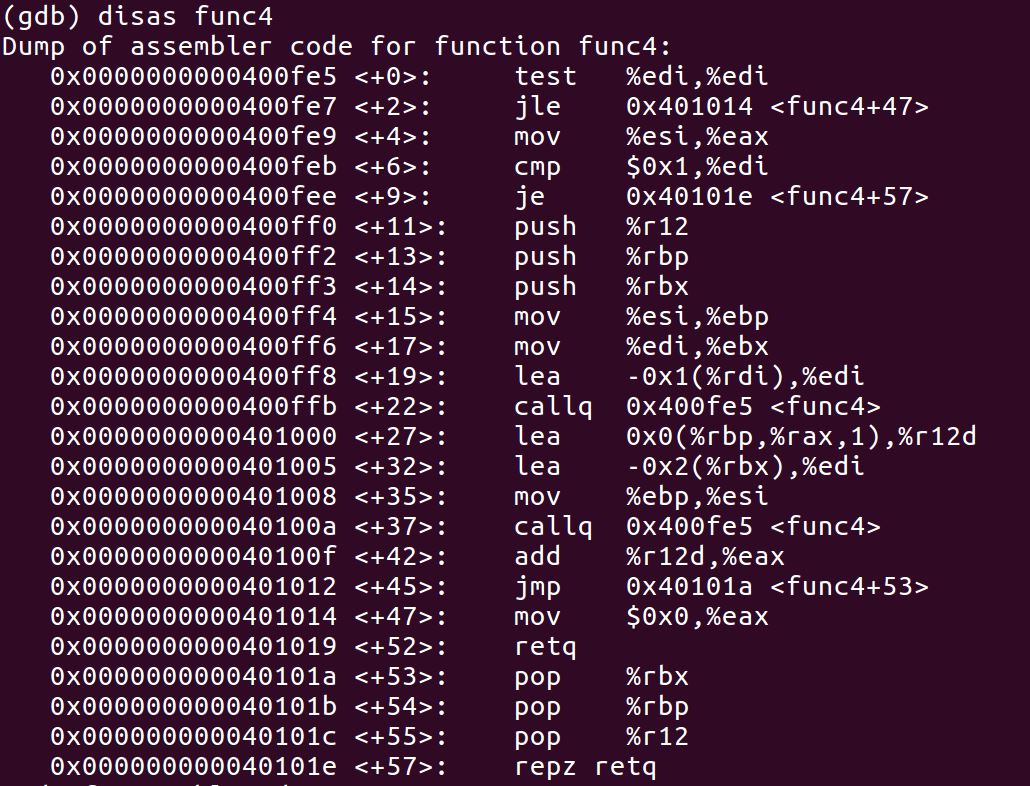


通过上面这段代码可知第二个数字满足0 <= x – 2 <= 2，即2 <= x <= 4。



可以看到此处调用func4，两个参数分别是第二个数和8，返回值需等于第一个数。

通过disas func4查看汇编代码：



观察汇编代码可知该函数含有递归，其对应的C代码应为：

int func4(int x, int y) {

if (y <= 0)

return 0;

else if (y == 1)

return x;

else

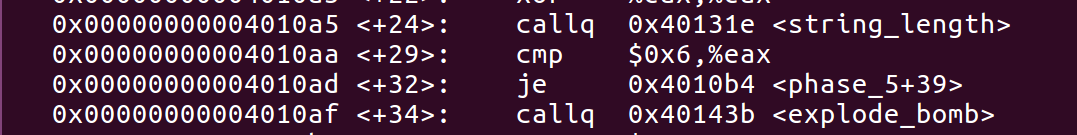
return func4(x, y-1) + func4(x, y-2) + x;

}

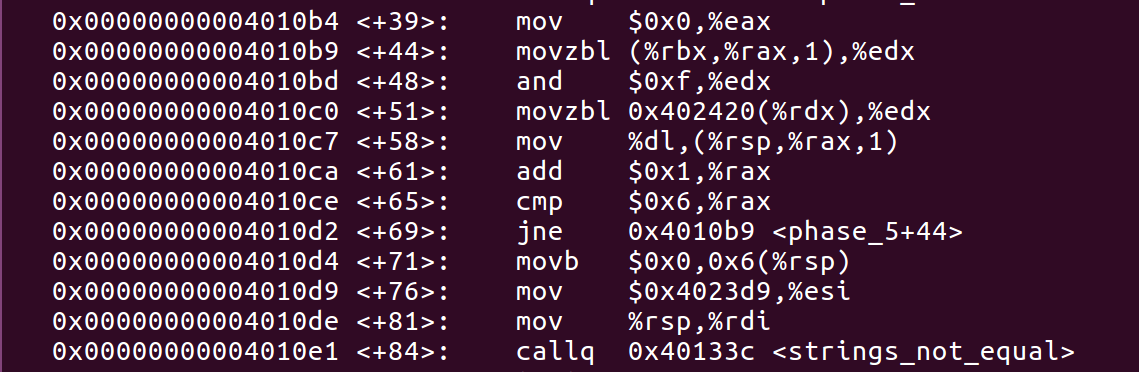
则两个数满足关系x = func4(y, 8) = 54 \* y。

则由此可推出三组解，分别为108 2，162 3和216 4，经检验均可通过。

Phase 5：通过disas phase\_5查看汇编代码：

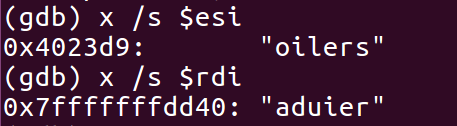


可见输入字符串的长度为6，即共有6个字母。



可见字符串比较之前通过某些规则进行了加密。

既然总共也只有26个字母，那不妨6个一组试一试吧。



在\*(0x40133c)处断点，查看寄存器%esi和%rdi的值，得知目标字符串为oilers。

abcdef –> aduier，同理ghijkl -> snfotv，mnopqr -> bylmad，stuvwx –> uiersn，yzabcd -> foadui，则根据这些转化关系可以列出26个字母的加密对照表：

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | **d** | **e** | **f** | **g** | h | i | **j** | k | l | m |
| a | d | u | **i** | **e** | **r** | **s** | n | f | **o** | t | v | b |

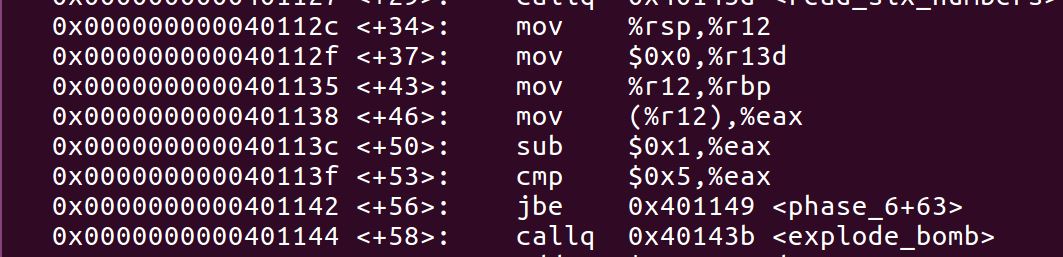
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | **o** | p | q | r | s | t | u | V | w | x | y | z |
| y | **l** | m | a | d | u | i | e | R | s | n | f | o |

由oilers逆推可得到其中一组符合条件的输入为jdoefg，经检验通过。

Phase 6：通过disas phase\_6查看汇编代码：

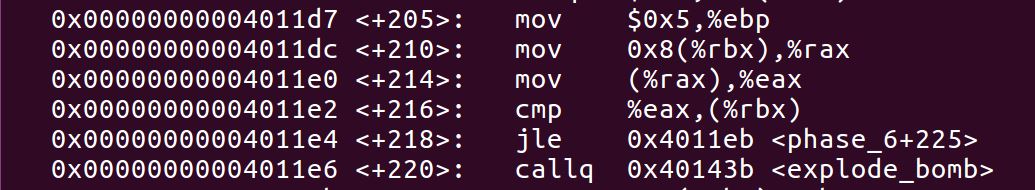


可见输入为6个整数，尝试1 2 3 4 5 6。

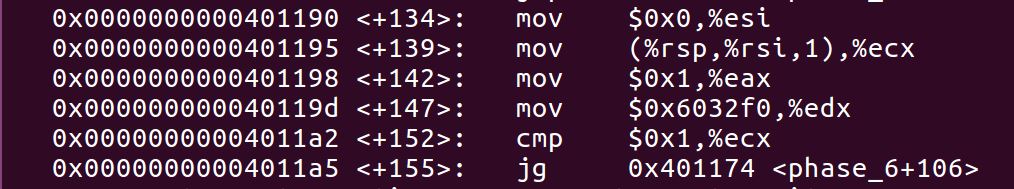


则0 <= a[0] – 1 <= 5。

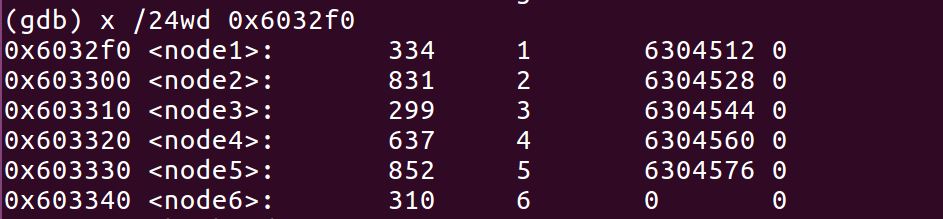
中间较长一段代码未见目标为explode\_bomb的跳转，直到：



通过观察可知这是一个循环，链表中的每一个数字需比上一个大，否则爆炸。

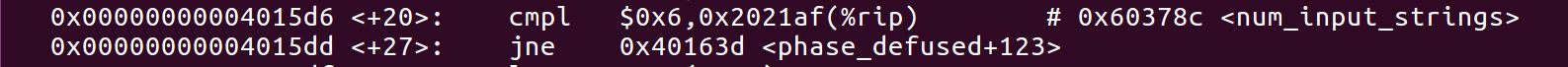


链表储存在地址0x6032f0处，从该地址开始查看链表：

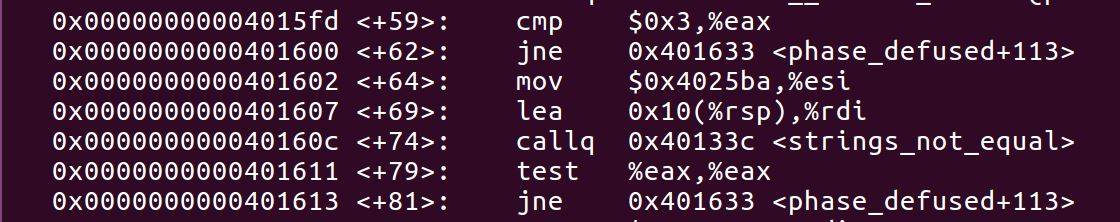


对该链表进行从小到大重新排序，应为3 6 1 4 2 5，经检验通过。

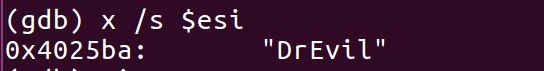
Secret Phase：观察提供的bomb.c代码，注意到phase\_6(input)与phase\_defused后没有其他操作了，进一步观察汇编代码可见secret phase隐藏在phase\_defused中。



进一步确认phase 6的defuse过程与其他关卡不同，可能是开启隐藏关卡的入口。

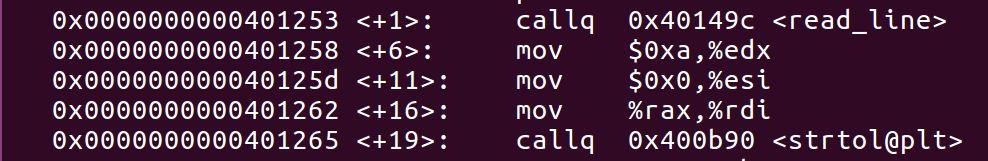


猜测此处的strings\_not\_equal用于测试phase 4的附加字符串是否正确。phase 4附加随机字符串，在\*(0x40160c)处断点，运行到该处查看寄存器%esi和%rdi。

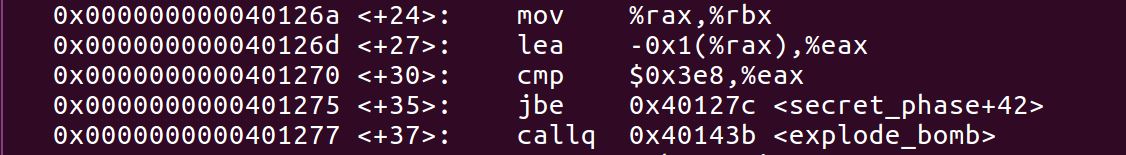


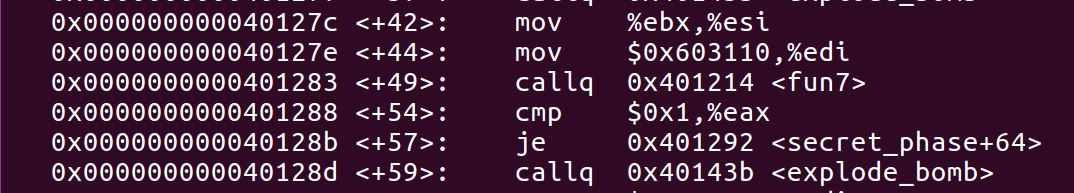
说明想要开启secret phase，须在phase 4后附加的字符串是DrEvil。

通过disas secret\_phase查看汇编代码：



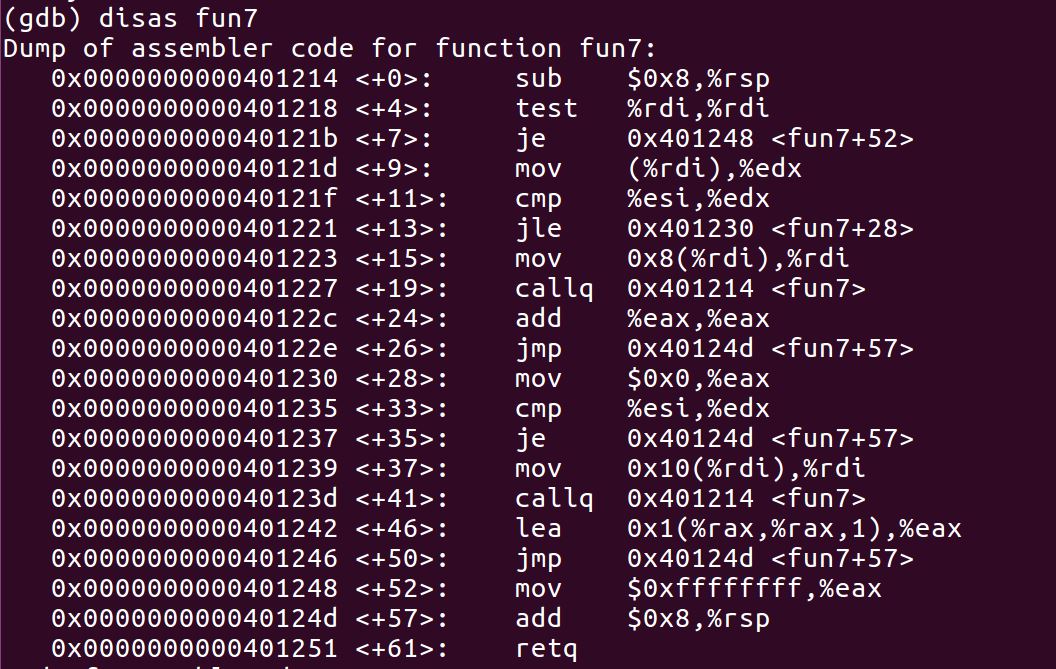
可见输入的字符串会被转化成十进制数。





可见这个数满足0 <= x – 1 <= 0x3e8，即1 <= x <= 1001，fun7的返回值为1。

通过disas fun7查看汇编代码，为一递归函数：



如果是已知参数，求返回值，就不一定需要推出函数的具体内容，可以直接通过寄存器获得特定输入的返回值，已知返回值倒退参数，就必须知道函数的具体实现了。

int fun7(int val, Node \*root) {

if (val == 0)

return -1;

else if (root->val == val)

return 0;

else if (root->val < val)

return 2 \* fun7(val, root->right) + 1;

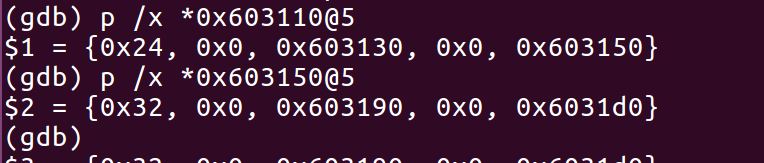
else

return 2 \* fun7(val, root->left);

}

返回值为1，推测上一次的返回值为0，由右子树返回，其val的值为0。

在\*(0x401221)处断点，通过根节点的地址0x603110，依次遍历该树。



由此可得val的值为0x32即50，满足val == root->right->val且val >root->val。

