**中国海洋大学 计算机科学与技术系**

**实验报告**

**姓名：陈岳阳 学号：21020007009** **专业：计算机科学与技术**

**科目：**计算机系统基础 **题目：lab2**

**实验时间: 2022/11/4**

**实验成绩: 实验教师: 范浩**

1. **实验目的**
2. **深刻理解汇编代码，并掌握将其翻译成伪代码或C代码的能力。**
3. **学习使用gdb反编译和调试程序的技能。**
4. **实验要求**

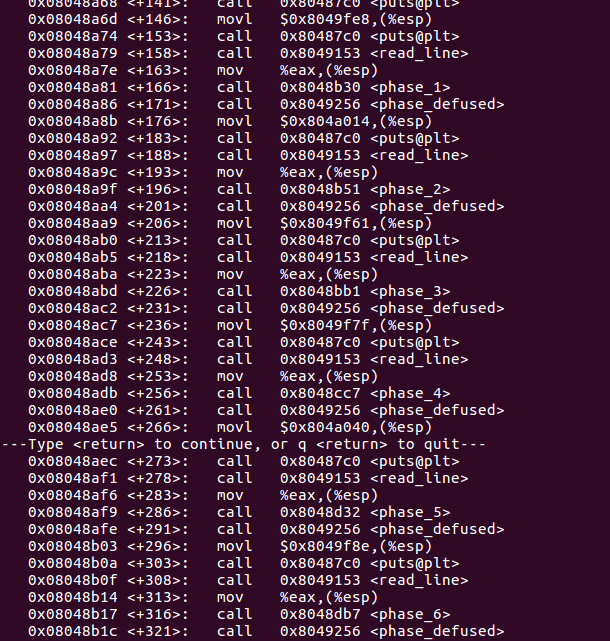
**反汇编bomb文件。看懂其汇编代码后，解决6个phase和secret\_phase。**

1. **实验内容（所修改函数代码，功能以及重要代码的解释）：**

# 实验结果图片19

Lab2  
Main  
Phase 1  
 答案：He is evil and fits easily into most overhead storage bins.  
Phase 2  
 read\_six\_numbers  
 phase\_2  
 答案：1 2 4 8 16 32  
Phase 3  
 答案：0 -92 或 1 -865 或 2 -30 或 3 -789 或 4 0 或 5 -789  
Phase 4  
 phase\_4  
 func4  
 答案：108 2 或 162 3 或216 4 (DrEvil)  
Phase 5  
 答案：5 115  
Phase 6  
 答案：2 4 6 3 5 1  
为什么还是没有结束？  
 Phase\_defused  
 Secret Phase  
 答案：35

## Main

第一件事当然是反汇编main。

* [图片1]main函数

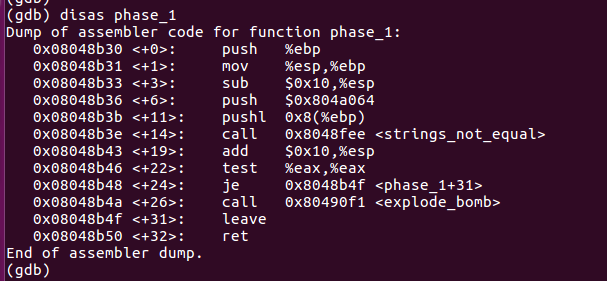
可以看到，main函数使用read\_line读入数据(实际上是猜的)，并且调用了6个phase(实际有7个)，并在每个phase结束后调用phase\_defused函数(实际上这6个phase\_defused都没有defuse bomb)。我们的目标是解决这6个bomb。

只要给explode打上断点，就可以一命通关。（只要没炸就不算死，太对了！）

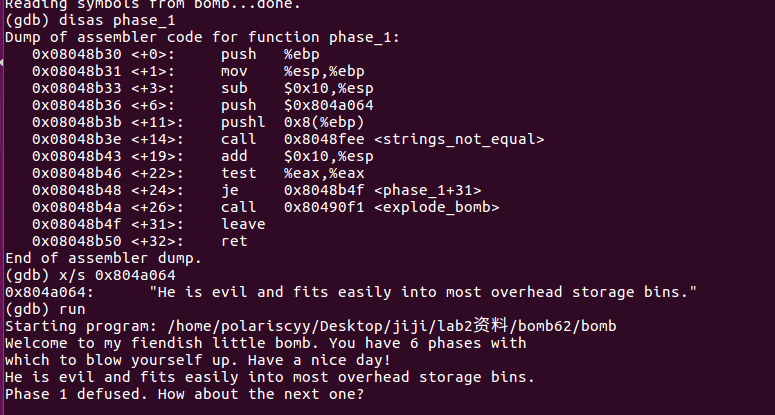
## 

## Phase 1

假设我已经会使用gdb了。我确实略知一二。

* [图片2]phase\_1反编译

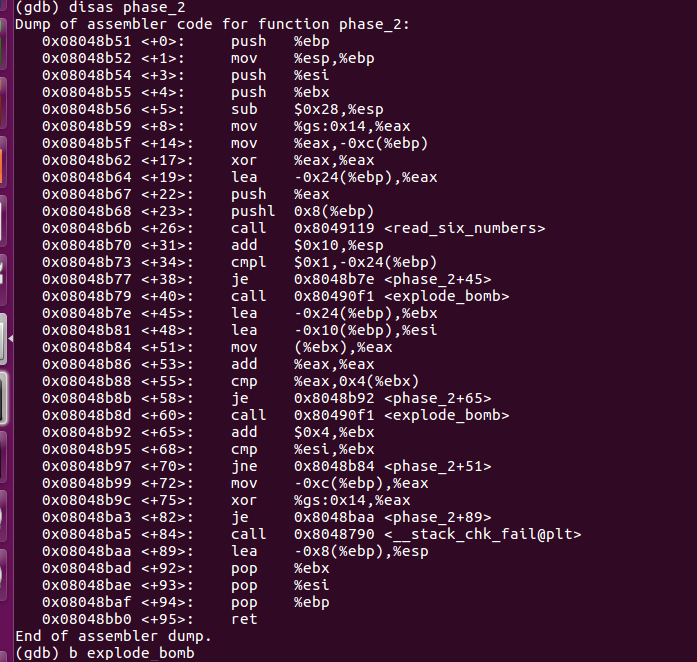
从图片中可以看到，phase1调用了stringsnotequal函数，并在函数返回值(eax中)为0时explode。stringsnot\_equal的效果就和函数名一样直白：将一个string与0x804a064处的字符串比较，相等则返回1。因此，我们看一下0x804a064处的字符串，尝试输入，通过。

* [图片3]输入字符串

##### 答案：He is evil and fits easily into most overhead storage bins.

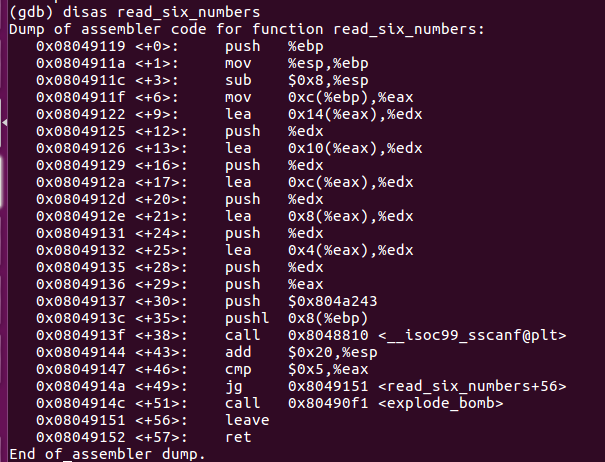
## Phase 2

上一问的答案已经存在ans.txt中了。

* [图片4]phase\_2反编译

%gs应该是Canary，不用管。我们还不会对栈做什么。

### read\_*six\_*numbers

可以看到调用了一个叫做read\_six\_numbers的函数。虽然它的用处也很明显，但还是反编译一下，因为后面会出现类似的坑爹情况。

[图片5]read\_*six\_*numbers

在read\_six\_numbers内部调用了一个\_isoc99\_sscanf@plt函数。这是一个信号函数，用于接受输入，并返回输入个数。但最坑爹的地方就在这里：这个函数并没有限制输入个数的上限（由0x804a243的格式串决定。下限则由explode\_bomb约束）。这个phase还用不到这点。

同时，我们可以看出，输入的六个数被存在栈中。

### phase\_2

让我们回到phase\_2主体。

实际上有效代码在+26~+70这些行中，伪代码如下：

for i = 0 to 5:  
 scanf("%d", &num[i]);  
  
if num[0] != 1:  
 explode\_bomb();  
else:  
 ebx = num;  
 esi = num + 5;  
 eax = \*ebx;  
 eax += eax;  
 if eax != \*(ebx + 1):  
 explode\_bomb();  
 else:  
 ebx = ebx + 1;  
 if esi == ebx:  
 You defused it!

进一步简化：

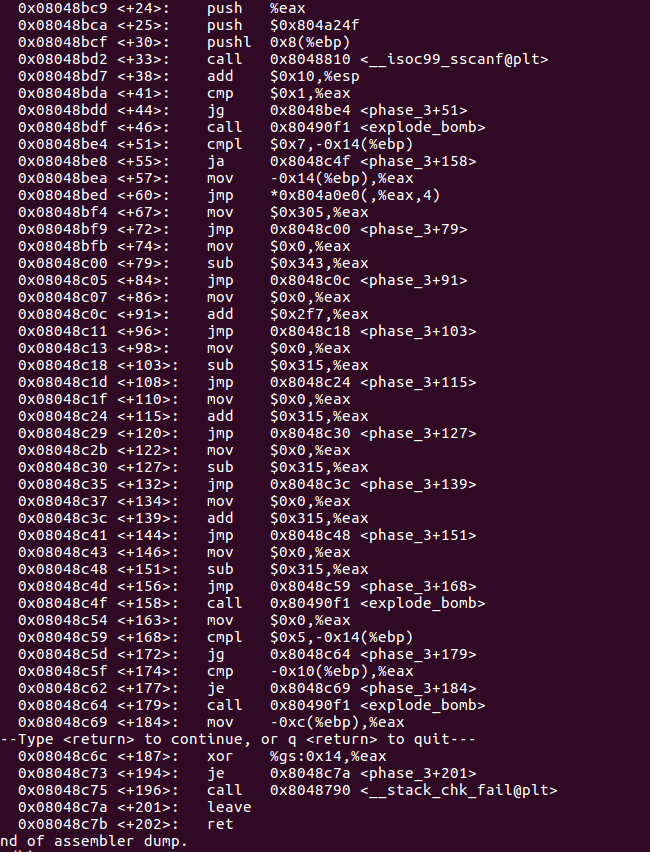
for i = 0 to 5:  
 scanf("%d", &num[i]);  
  
ebx = num;  
eax = 1;  
if (\*ebx != eax)  
 explode\_bomb();  
  
while(ebx++ != num + 5){  
 eax <<= 1;//已经保证了\*ebx=eax  
 if(\*ebx != eax)  
 explode\_bomb();  
}

可以看出，只要第一个输入为1，之后每一个输入为前一个的2倍即可。

答案：1 2 4 8 16 32

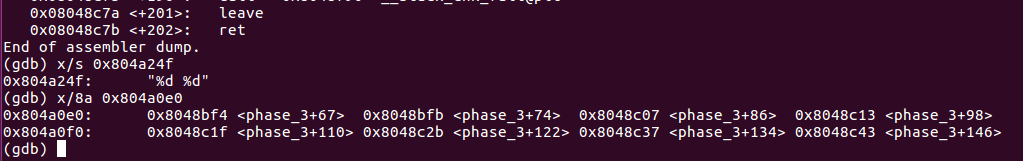
## 

## Phase 3



[图片6]phase\_3

当我看到\*和一个奇怪的数时，鉴定为：跳转表。

[图片7]跳转！

输入num val。如果num > 7或num < 0，调用explode\_bomb()；否则，进入跳转表。伪代码如下：

switch(num){  
 case 0: num = 0x305;  
 case 1: num -= 0x343;  
 case 2: num += 0x2f7;  
 case 3: num -= 0x315;  
 case 4: num += 0x315;  
 case 5: num -= 0x315;  
 case 6: num += 0x315;  
 case 7: num -= 0x315; break;  
 default: explode\_bomb();  
}  
if (num > 5)  
 explode\_bomb();

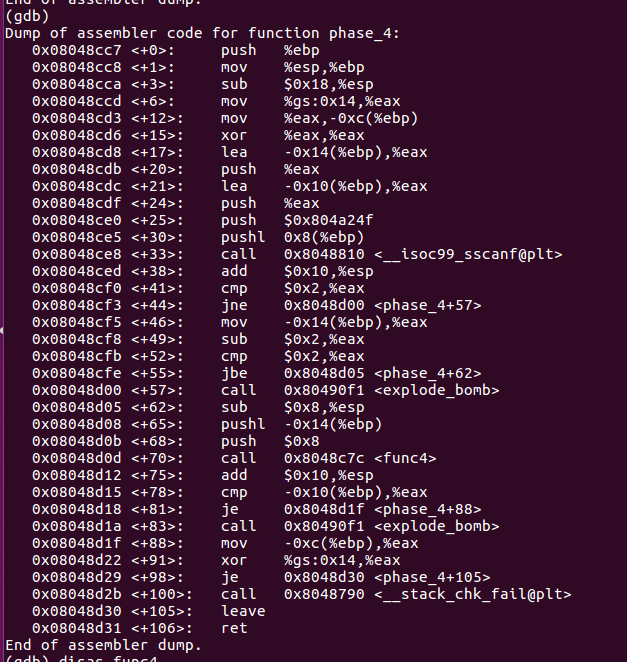
化简如下：

if(num < 0 || num > 5)  
 explode\_bomb();  
  
switch(num){  
 case 0: num = -92;break;  
 case 1: num = -865;break;  
 case 2: num = -30;break;  
 case 3: num = -789;break;  
 case 4: num = 0;break;  
 case 5: num = -789;break;  
}

##### 答案：0 -92 或 1 -865 或 2 -30 或 3 -789 或 4 0 或 5 -789

## Phase 4

### phase\_4

老规矩，先看看phase\_4。可以看到这里面没什么东西。scanf输入两个整数，且输入的第二个数减2后为0/1/2，即第二个输入为2/3/4。

此外，唯一有用的信息就是func4，和最后的cmp -0x10(%ebp), %eax。-0x10(%ebp)是存放输入的第一个数（猜的，因为后面用不到这个）。

func4的parameter 1是8，parameter 2是输入的第二个数。

* [图片8]phase\_4

### 

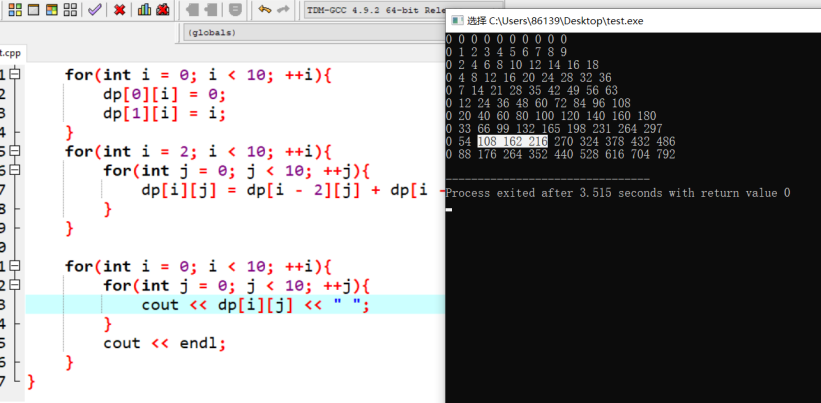
### 图片9func4

* [图片9]func4

令人头疼的递归式。我们直接让它运行到explode\_bomb，eax中存放的值就是第一个数。

完结撒花！......

好吧，我们还是认真做一下。下面是func4主体转换成的代码（经过本人化简）：

int input\_1, input\_2;  
int func4(int x, int y);  
  
//func4+9开始  
int func4(int x, int y){  
 if x <= 0:  
 return 0;  
 if x == 1:  
 return y;  
 return func4 (x - 2, y) + func4(x - 1, y) + y;  
}  
  
int main(){  
 scanf("%d %d", &input\_1, &input\_2);  
 if ( unsigned(input\_2 - 2) > 2)  
 explode\_bomb();  
 if( func4(8, input\_2) != input\_1 )  
 explode\_bomb();  
 phase\_defused();  
}

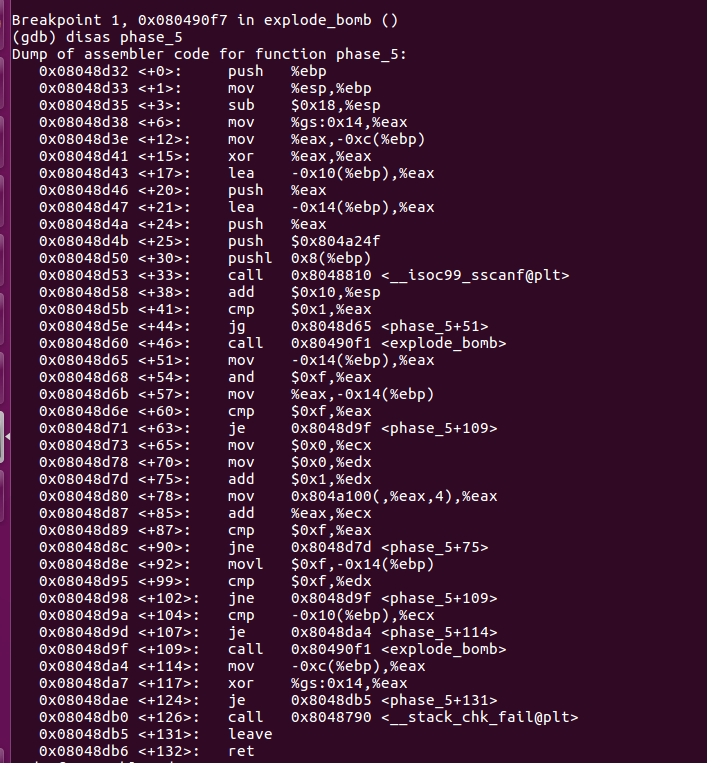
* [图片10]dp

做一个dp，dp[8][2], dp[8][3], dp[8][4]就是2/3/4对应的结果。

答案：108 2 或 162 3 或216 4 (DrEvil)

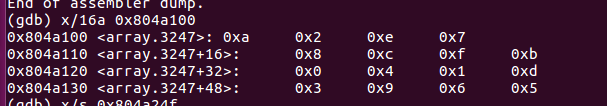
## 

## Phase 5

* [图片11]phase\_5

scanf要求输入两个十进制整数。

第一感觉像是循环。尝试将asm翻译成伪代码。

* [图片12]数组

//+60  
if (eax == 0xf)  
 explode\_bomb();  
ecx = 0;  
for(int dex = 0; dex < 15 && eax != 15; ++i){  
 eax = num[eax];  
 ecx += eax;  
}  
if (ecx != \*(ebp-0x10))  
 explode\_bomb();

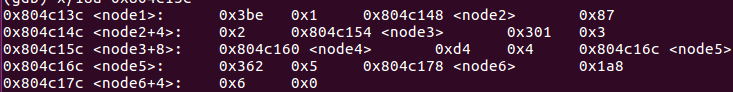
其中，num是0x804a100处的数组，长度为16sizeof(int)，存储0~15的整数各一。从某个数开始，访问15个数，到达0xf，即除了num[0xf]外全部访问，总和ecx=120-num[0xf]=115，是输入的第二个数。&num[num[0xf]] - num = 5, 是输入的第一个数。

##### 答案：5 115

## 

## Phase 6

太长了，图片截不下来。直接翻译成伪代码好了。其中包含注解。

* [图片13]链表

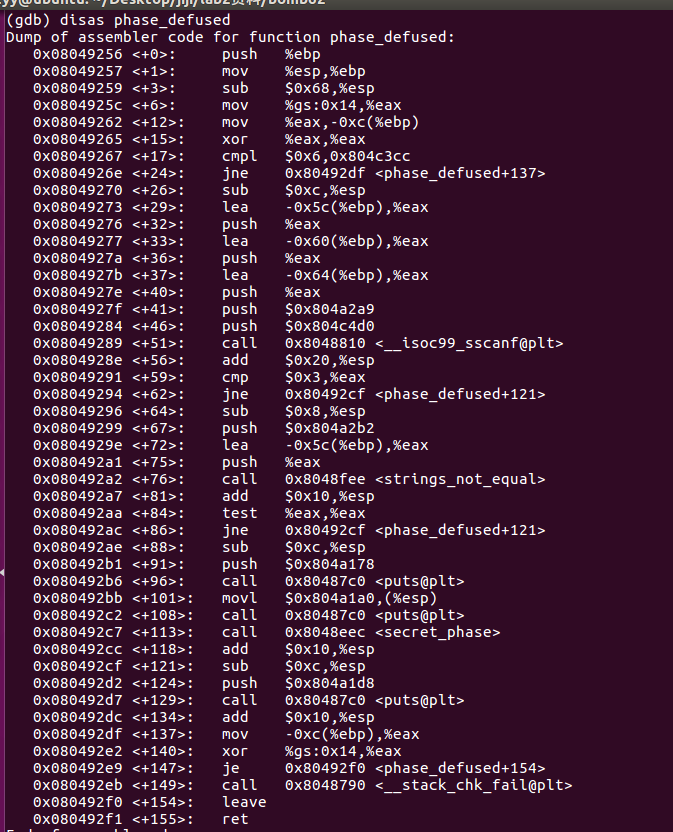
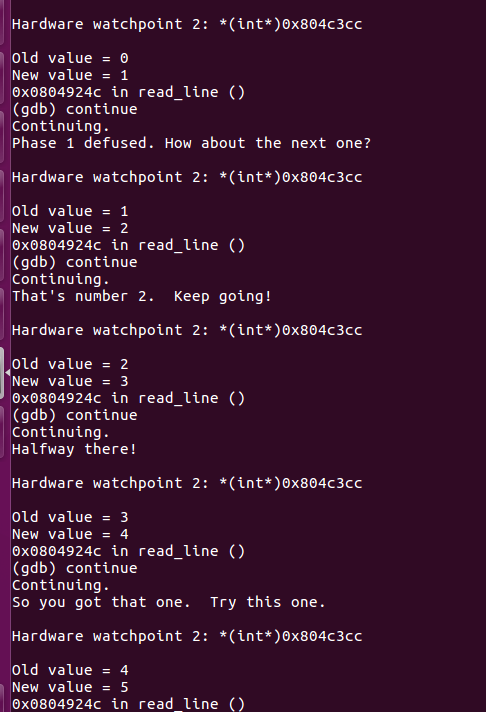
//+26  
int num[6];  
for i in range(6):  
 scanf("%d", &num[i]);  
   
for(esi = 0; esi < 6; ++esi):  
 eax = \*(ebp + 4 \* esi - 0x3c) - 1  
 if eax > 5:  
 explode\_bomb();  
 esi++;  
 if esi is 6:  
 break;  
 for(ebx = esi; ebx < 6; ++ebx):  
 eax = \*(ebp + 4 \* ebx - 0x3c);  
 if eax is \*(ebp + 4 \* ebx - 0x40):  
 explode\_bomb();  
   
/\* 第一个部分用来检查输入。输入存在ebp - 0x3c, ebp-0x38, ..., ebp-0x28六个地址中。  
 对于每个输入，val＞5+1=6，或val≤0，则explode；内部嵌套的循环用来判断当前的val是否与后面的输入相同。如果相同，则explode。因此，可以判断，输入的六个数是123456的一种排列。  
 \*/  
   
  
for(ebx = 0; ebx < 6; ++ebx):  
 ecx = \*(ebp + 4 \* ebx - 0x3c);  
 for(eax = 1, edx = 0x804c13c; eax != ecx; ++eax):  
 edx = \*(edx + 0x8); // edx = edx->next  
 \*(ebp + 4 \* ebx - 0x24) = edx  
/\* ebp - 0x24, ebp - 0x20, ..., ebp - 0x10指向的数是0x804c13c + 0 \* 12, ..., 0x804c13c + 5 \* 12的一个排列。排列的顺序和输入有关。顺序:   
node[num[0]-1], ..., node[num[5] - 1]  
 使用x/18a 0x804c13c可以看到，这其实是一个链表。每个结点存储了一个value，一个序号，和指向下一个结点的next指针。  
 这个循环将六个结点的首地址在一个新的空间存储。  
\*/  
  
ebx = \*(ebp - 0x24)  
esi = ebp - 0x10  
ecx = ebx  
for(eax = ebp - 0x24; eax != esi; eax += 4):  
 edx = \*(eax + 0x4)  
 \*(ecx + 0x8) = edx  
 ecx = edx  
/\* 按上一个循环的顺序重排链表 \*/  
   
\*(edx + 0x8) = 0  
/\*赋空指针\*/  
for(esi = 5; esi != 0; --esi):  
 eax = \*(ebx + 0x8)  
 eax = \* eax;  
 if \*ebx > eax:  
 explode\_bomb();  
 ebx = \*(ebx + 0x8)  
/\*如果结点不是升序排列，则explode\_bomb\*/  
phase\_defused()

##### 答案：2 4 6 3 5 1

## 为什么还是没有结束

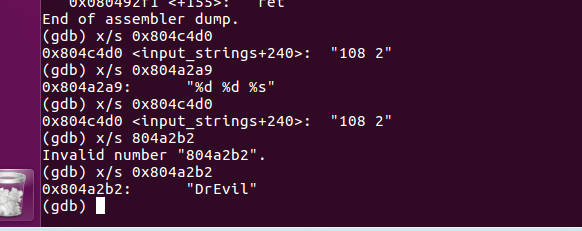
### Phase\_defused

bomb迟迟没有拆掉，那一定是defuse函数有问题。反汇编如下：

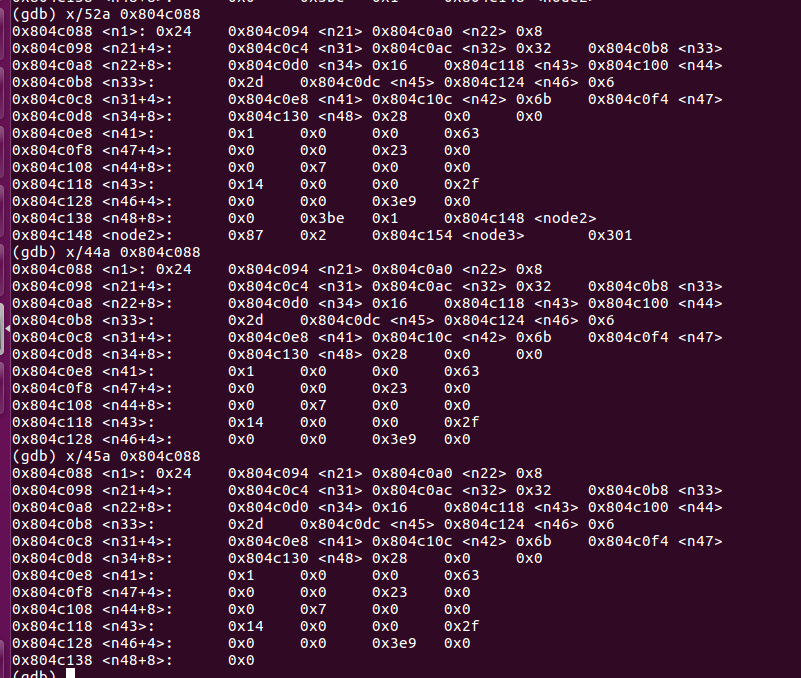
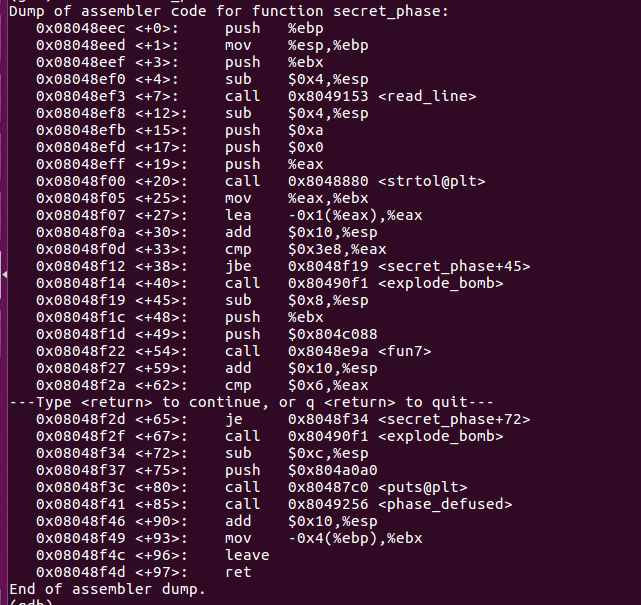
* [图片14]phase\_defused（左图）
* [图片15]0x804c3cc（右图）

可以看到，phase\_*defused中有一个secret\_*phase。而判断能否进入secret\_*phase有两个条件。其中一个要求0x804c3cc的值等于6。使用watch \*(int\*)0x804c3cc，运行bomb可以发现，每进入一关，这个值会+1。即进入secret\_*phase的条件之一是解决前6个phase。

另一个奇怪的是scanf。上面出现了两个常数，分别查看，可以看到：前者是格式串，而后者是phase4的输入。因此，我们需要在phase4中再输入一个字符串。输入的字符串与0x804a2b2处的字符串相同，是DrEvil。

* [图片16]DrEvil

## Secret Phase

* [图片17]secret\_phase和fun7（左图
* [图片18]完全二叉搜索（右图）

其实就是一棵二叉搜索树，翻译出来是这样的：

int value, eax;  
node \*root;  
  
int fun7(int val, node\* n):  
if val < 0:  
 explode\_bomb();  
if n->val == val:  
 return 0;  
elif n->val > val:  
 eax = 2 \* fun7(val, n->lchild)  
else:  
 eax = 2 \* fun7(val, n->rchild) + 1  
  
if eax == 6:  
 phase\_defused();

总共要搜索不超过三次。唯一可能是向左一次，再向右两次，结果是。结点的值是0x23，即35.

##### 答案：35