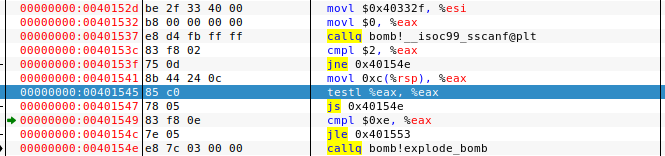
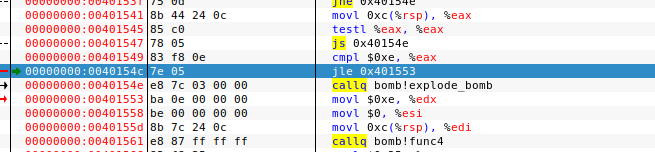


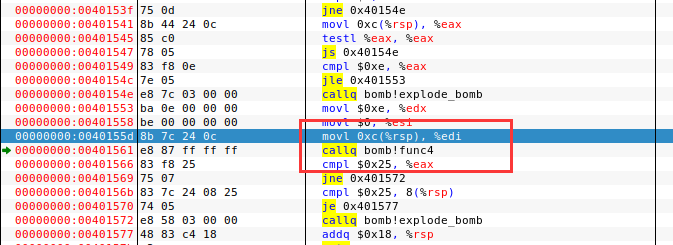
通过第4关刚开始的代码分析可知这一关还是读取了两个整数，第一个整数放在rsp偏移0xC的位置，第二个整数放在rsp偏移0x8的位置。



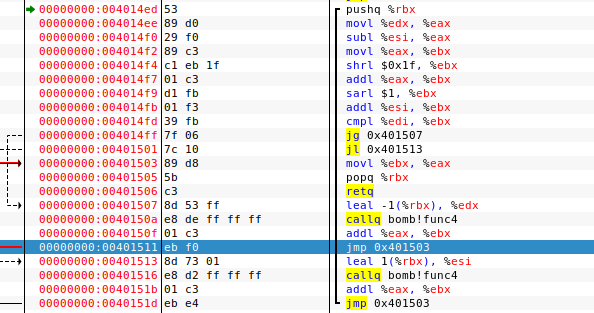
第一次比较时，将rsp偏移0xC的数放到了rax寄存器中，也就是输入的第一个数，通过比较，如果这个数是个负数就会跳转到0x40154e，也就是爆炸函数，因此输入的第一个数必须是正数或0。



第二次比较同样比较的是输入的第一个数，如果这个数小于或等于0xe，就不会执行爆炸函数，因此要求输入的第一个数不大于0xe，也就是十进制的14



第三次比较时用到了一个函数fun4，如果函数fun4的返回值不等于0x25则会导致爆炸函数，因此必须设法让fun4的返回值为0x25，也就是十进制的37。Fun4的参数为{%edi: 输入的第一个参数，%esi: 0，%edx: 0xe，也就是十进制的14}。下面分析函数fun4。



本人通过逆向工程技术，逐步分析语句，将函数fun4反编译出的结果如下：

fun4(x, y, z)

//x in %edi, y in %esi, z in %edx

{

int ret;

ret = z - y;

int n = (unsigned)ret >> 0x1f; //n in %ebx，0x1f = 31(十进制)，获取n的符号位

n += ret;

n >>= 1;

n += y;

if(n > x){

//0x401507

z = n - 1;

n += fun4(x, y, z);

}

else if(n < x){

//0x401513

y = n + 1;

n += fun4(x, y, z);

}

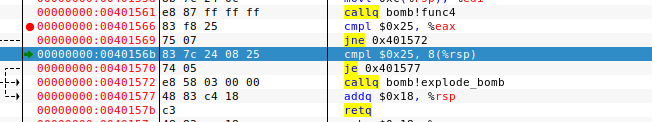
ret = n;

return ret;

}

在第4关的函数中初次调用fun4是的传参为fun4(屏幕输入的第一个参数, 0, 14)

为了让fun4函数返回37，只需要控制fun4的第一个参数即可(即在第4关最初输入的第一个参数)，而由上面的分析可知第一个参数只能取0~14之间的某个数，通过逐一尝试输入0~14发现，当输入的第一个参数为10(十六进制的0xa)时，可以让fun4函数返回37(0x25)，因此第一个数必须输入10。



执行完fun4之后，输入的第2个参数与0x25比较，如果不是0x25则执行爆炸函数，因此输入的第2个参数必须是0x25，也就是十进制的37。