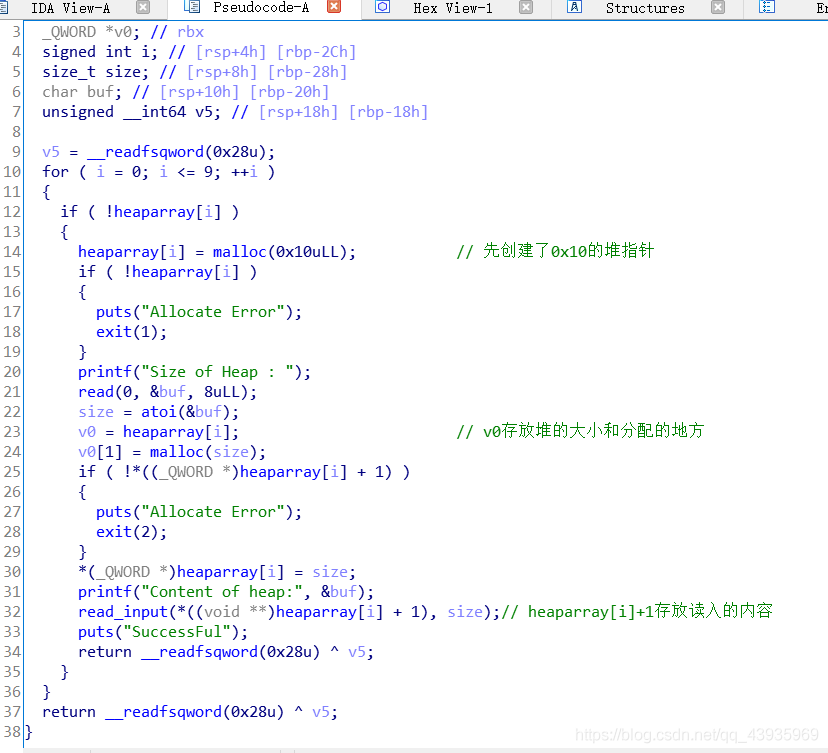
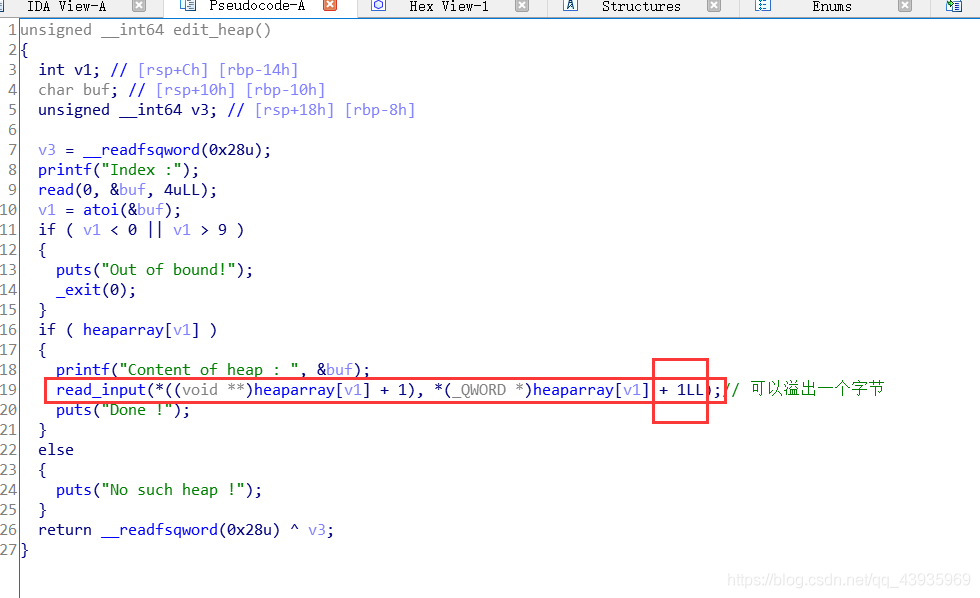
<https://blog.csdn.net/qq_43935969/article/details/105486593>

分析和漏洞点：

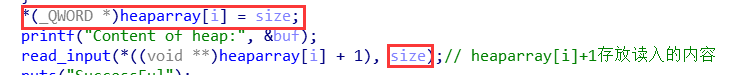
创建的时候，每次都会先创建0x10的heaparray，然后再创建堆



对比建堆，可以看到size大了一个



对比：



程序基本功能就这样了

因为edit的时候可以多写入一个字节，存在off-by-one，我们可以通过溢出覆盖下一个字节，为了覆盖到下一个字节的size位，因为程序存在复用，即如果都inuse的话，上一个chunk可以使用到下一个chunk的prev\_size，那么我们再溢出，就可以到size位了。

步骤：

（1）我们是首先申请了0x18字节（实际就给了0x10+下一个chunk的prev\_size）

（2）然后再申请0x10

（3）edit第一个chunk，并且输入‘/bin/sh\x00’，通过栈溢出，将第二个chunk的size设置为0x41【解释1】

（4）free掉第二个chunk

（5）重新申请0x30的chunk大小【解释2】

（6）将free\_got的地址输入chunk，并且打印出来

（7）计算出基地址、system地址

（8）覆盖free\_got表为system地址

（9）删除第一个堆块即可

create(0x30,p64(0)\*4 +p64(0x30) + p64(free\_got)) #1

重新申请堆块，最重要的一块，重新申请的堆块大小为0x30，也就是我们之前释放的0x41，这样子，就成功的将heaparray和chunk换了位置，，，

我们知道heaparray的第二个放的是函数指针，，那么如果我们申请到的地址被系统认为是heaparray，让我们在chunk里面布置上的地址就会被认为是函数地址，会去执行，我们也就可以进行一系列操作了

因为heaparray的大小是0x10，所以会去链表里找被释放有没有符合大小的，因为有，所以会把之前的chunk给他，而现在要的chunk大小是0x30，会把之前更改后释放的0x41的heaparray的地址给他们，成功实现互换，，，

之前我们已经将指向chunk2地址的指针覆写成了free的got表，也就是说chunk2里的内容现在是free函数的地址，现在往chnk2里写入system函数的地址，覆写掉了free函数的地址，也就是说现在free的got表地址指向的是system函数