Unsort\_bin\_attack理解

Unsorted bin Attack: When we allocate a chunk, it would process the unsorted bin first. It would remove the chunk in unsorted bin whether or not the size matches. However, it does not check the completeness of the linked list. Before the unsorted chunk is removed from the unsorted bin, we can overwrite the bk pointer with any address-0x10. And then the address will be overwritten with the address of unsorted bin. (source) We decide to use it to overwrite \_IO\_list\_all with the address of unsorted bin

unsigned long stack\_var=0;

首先，我们想rewrite

0x7ffee993da10: 0

然后声明第一个chunk，

0x1a91010

再声明一个，防止free第一个的时候，其与topchunk混合

malloc(500);

free第一个chunk

free(p)

结果是该chunk被插入到unsorted bin，且其bk指向0x7fd9ef9fd7b8

//p[1]的确是其bk指针，但是不知道为什么是0x7fd9ef9fd7b8

人工模拟一个漏洞，可让该unsorted bin上的chunk的bk指针指向&stack\_var-16

其bk被改为0x7ffee993da00

再申请一个适当大小的chunk

malloc(400);

此时，&stack\_var，(void\*)stack\_var)，后者被修改了：

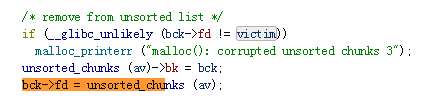
0x7ffee993da10: 0x7fd9ef9fd7b8

意思是:

当unsorted bin上的chunk块被分配时，该块的后一个chunk(bk)的前一个fd(+0x10)上放置该块的后一个指针(bk)。

总结:

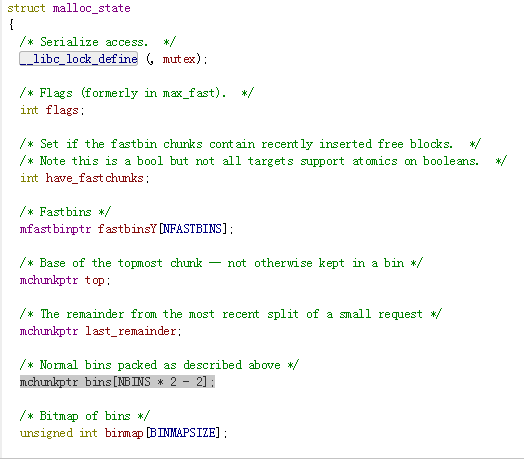
fastchunk < malloc\_size < unsort chunk时，会先将unsort chunk拆卸下来，过程中不会检查fd和bk是否被篡改，导致在unsort bin尾部(FILO)的chunk的bk+0x10处写入unsort\_bin地址。



如果我们可以溢出到unsort\_chunk的bk,那么unsort\_bin\_addr(main\_area+0x58,第12个bin，10fast bin+top\_chunk\_addr)就可以被写到任意地址了。

注意：双向链表的chunk的bins在main中占用两个地址长度。

Main\_area结构：



Main\_area:偏移0x58对应unsort\_bin地址；

Unsort\_bin\_addr+0对应unsort\_chunk\_bin

Unsort\_bin\_addr+10对应不知道

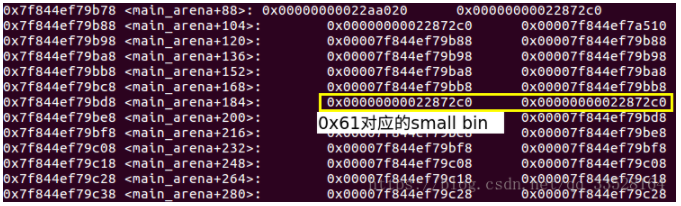
Unsort\_bin\_addr+20对应small\_chunk\_bin\_size20

Unsort\_bin\_addr+30对应small\_chunk\_bin\_size30

Unsort\_bin\_addr+40对应small\_chunk\_bin\_size40

Unsort\_bin\_addr+50对应small\_chunk\_bin\_size50

Unsort\_bin\_addr+60对应small\_chunk\_bin\_size60



<https://bbs.pediy.com/thread-222718.htm>

https://blog.csdn.net/weixin\_40850881/article/details/80043934