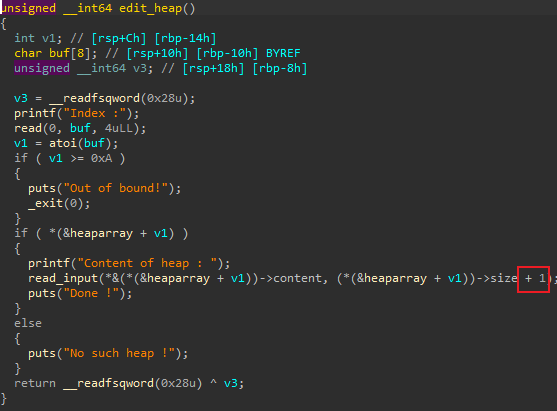
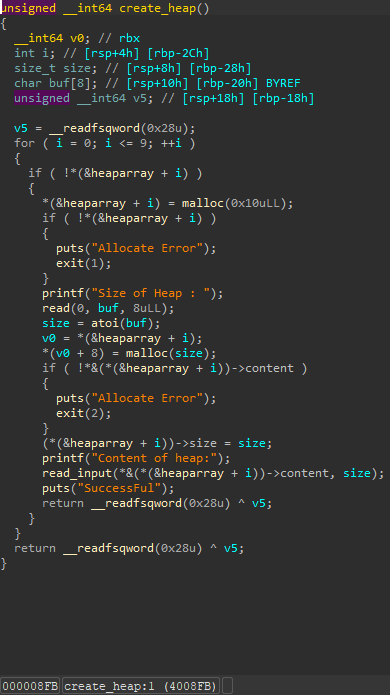
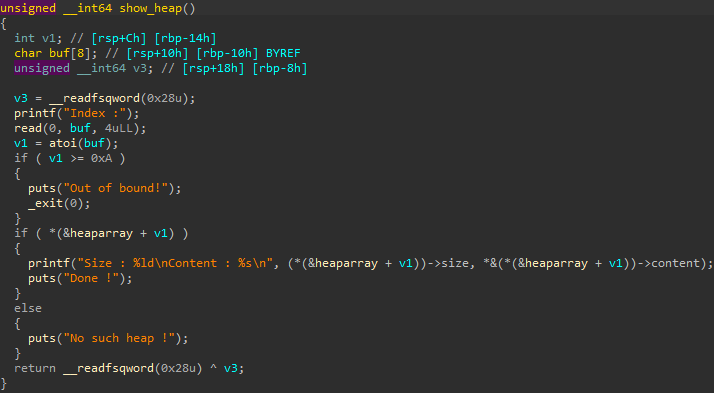
**off\_by\_one\_2**

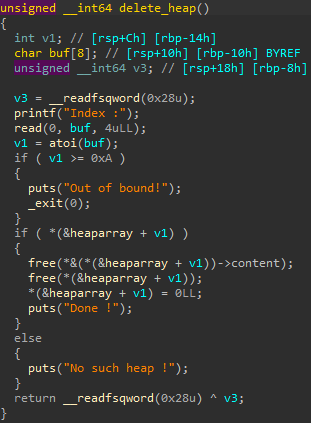


Buu的一题



创建的时候会在最前面创建一个0x10大小的chunk用来存用户所创的chunk的地址，edit的时候可以编辑size+1大小，经典的off by one





show和delete都很正常，没有利用点

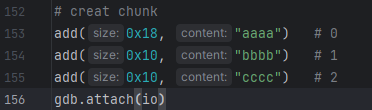
利用思路：

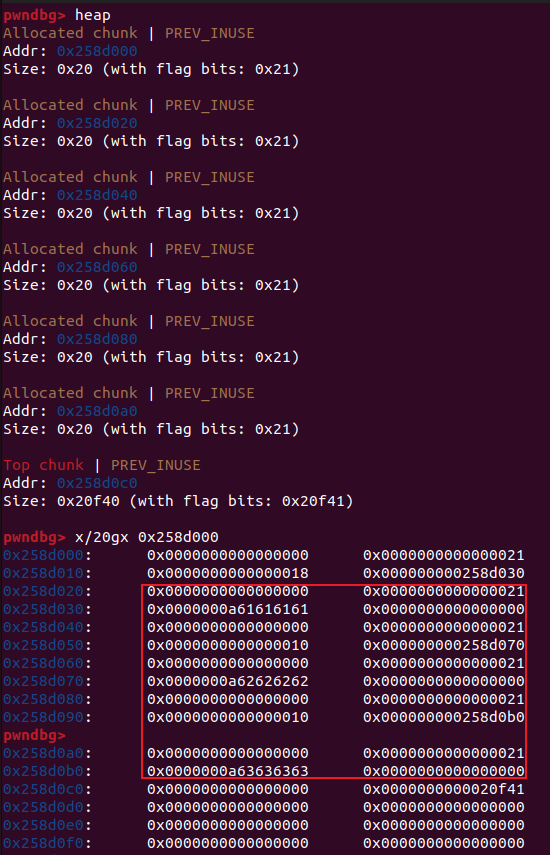
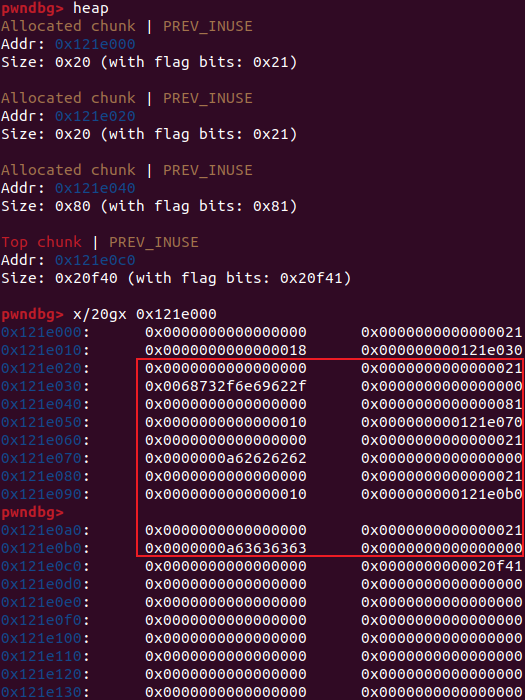
1. 申请三个堆块012
2. edit chunk0时利用off by one修改chunk1的size能覆盖到chunk2
3. free掉chunk1后，再申请一个chunk跟修改后的chunk1的size相同的堆块（有啥用？）
4. 修改chunk1，这时就能修改chunk2的内容，在chunk2中写一个free函数的got表的值
5. 这时show一下chunk2就能泄露libc
6. 再edit一下chunk2就能修改free函数的值，改为system的plt表的值就能实现free chunk0的时候执行system(‘/bin/sh’)，其中binsh是第二步顺便写进去的

利用过程：

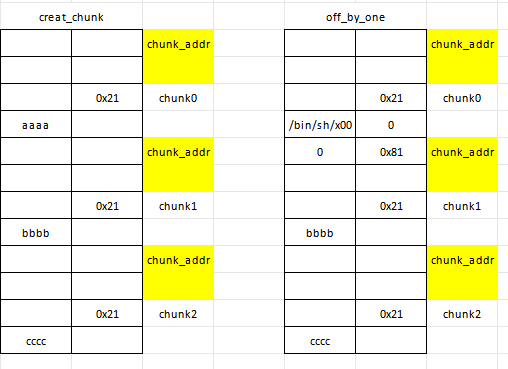
1.->2.:

申请->off by one:

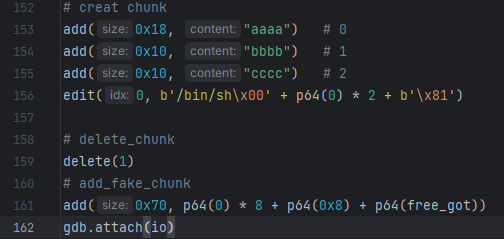
 

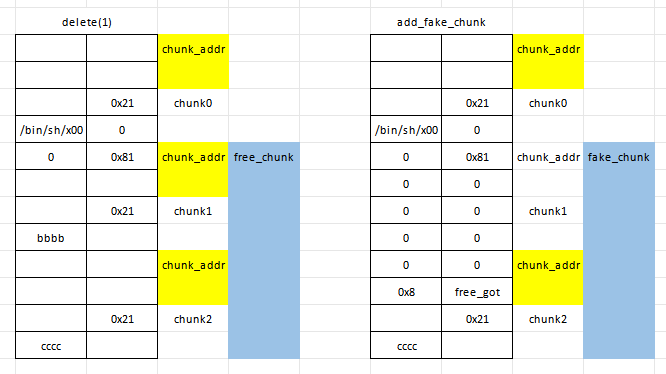
 

每个chunk之前的chunk\_addr就是用来保存我们申请的chunk的地址的chunk，在这里根据gdb或者栈示图就可以计算出要覆盖0x80个字节，在edit完chunk1之后，gdb就不会识别到有chunk2的存在（上：右下图），但是不影响chunk1的使用

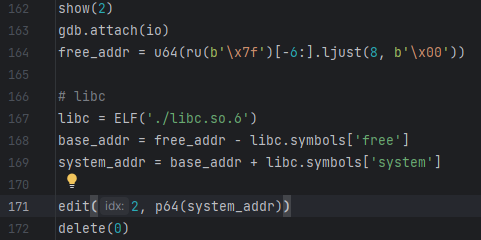


下一步，我们把伪造的这个堆块申请出来，申请的同时修改chunk2的chunk\_addr中存放chunk2地址的部分为free的got表值





这样再show(chunk2)的话就能打印出free函数的真实地址，就能算出libc基址



而edit(chunk2)就能改free函数地址的值为system函数的地址，再delete一下就能执行system(‘/bin/sh’)了



本题属于off\_by\_one的第二种利用方法：拓展已分配块，该方法的成功依赖于free函数完全依赖size域来判断一个即将被释放堆块的大小