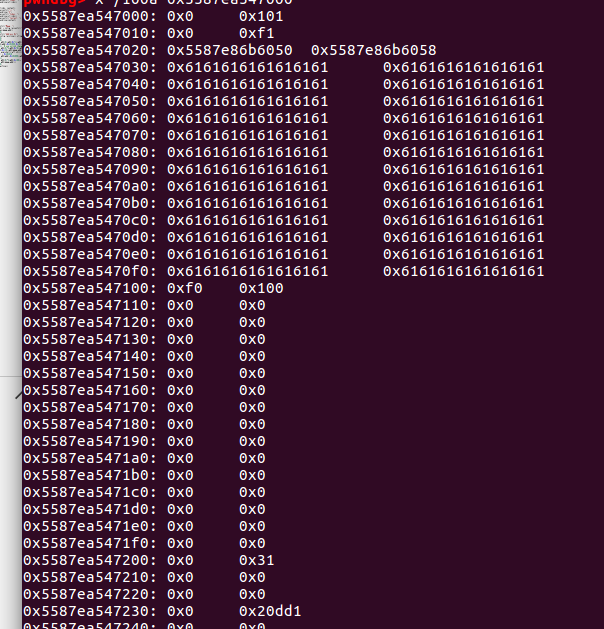
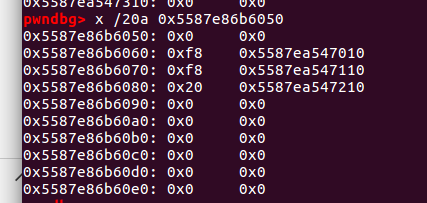
初始化这个heap：

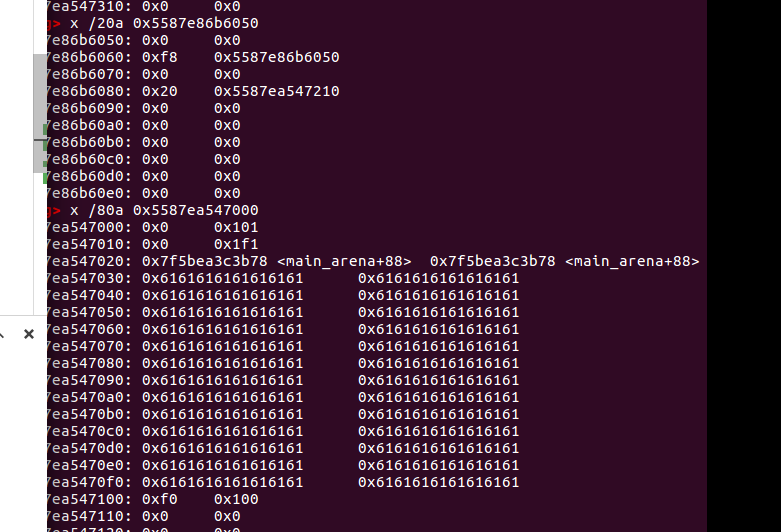




将chunk1伪造出fd和bk，其中fd指向chunktab的上一个位置，p1的地址-0x18，希望通过unlink来修改p1的值？

通过null byte off by onb修改p2的size低位为0，伪造一个已经free的假象，同时修改了p2的presize，这里的值与在p1中伪造的chunksize一致，0xf0

接着free掉chunk2



这里主要有两处变换“

1. 在chunk1中伪造的fakeChunk由于chunk2的free，导致其FD和BK指向了main\_arena，因此获得了libc
2. 在chunkTab中由于上一次指针指向0x50那里，unlink后0x50+0x18=0x68的地方被修改成chunk1中fakeChunk的fd

为什么会有这样的效果？

首先null byte溢出，使得chunk2的size位低位为0，使得上一个fakeChunk为free态，从而在free chunk2时，可以与fakeChunk合并，从而实现unlink的目的。

在unlink时，首先有两个检查：

1. size检查，查看当前这个chunk的size和下一个chunk的pre\_size是否一致，

这里要unlink的是fakeChunk，fakeChunk的size为0xf0，而在chunk2的pre\_size位也通过溢出修改成了0xf0，check pass

1. 链表完整性检查，这里是检查fakeChunk的双向列表完整性，

FD：fakeChunk->fd=0x50，BK：fakeChunk->bk=0x58

在检查时，会有FD->bk == P ? :fakeChunk+8(0x50)+0x18 = 0x68 = fakeChunk\_Pointer

还有BK->fd==P？：fakeChunk+12(0x58)+0x10=0x68=fakeChunk\_Pointer

这里的0x68中存放的是malloc得到的chunk1的mem地址，刚好是指向fakeChunk的开头，从而正好绕过这个检查，实现了unlink

进行unlink操作，会引发chunk2和fakeChunk的合并，从而对fakeChunk进行unlink

在chunkTab中修改了0x68位置是由于unlink造成的，那么在合并后的chunk中为什么会遗留main\_arena？

‘’‘

方法二：使用main\_arena获取libc基址

上面我们已经介绍了一种leak地址的方式，下面我们介绍另一种方式获取地址的方式，那就是通过main\_arena来获取，在fastbin为空时，unsortbin的fd和bk指向自身main\_arena，而main\_arena存储在libc.so.6文件的.data段，通过这个偏移我们就可以获取libc的基址，这里我讲一下怎么找到main\_arena的地址，首先使用IDA打开libc文件，然后搜索函数malloc\_trim()，具体如下图所示。

‘’‘

注意这里unsortBin在fastBin为空的时候会将自身的fd和bk指向main\_arena，因此这里遗留了main\_arena的地址