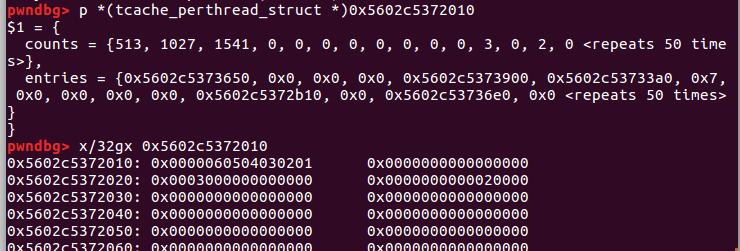
在本题使用2.31版本的setcontext并没有直接的博客参考，主要是参考

<https://blog.csdn.net/yongbaoii/article/details/119544590>

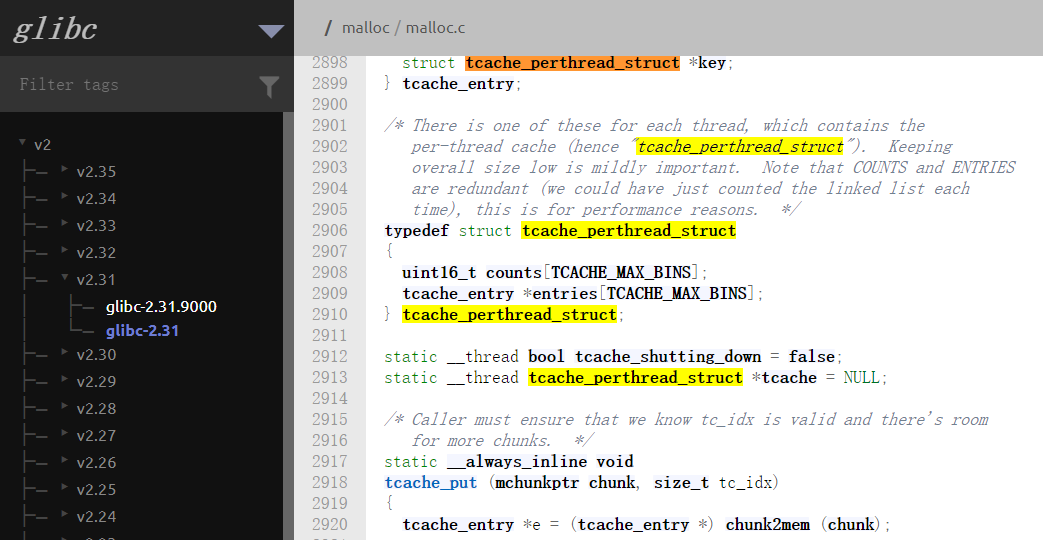
<https://blog.csdn.net/carol2358/article/details/108351308>

其中第一个师傅利用的是2.29的libc但使用的是一般在2.31中使用的magic\_gadget而非一般2.29中使用的magic\_gadget，实际上在第二个师傅的博客里面详细讲解了2.27，2.29，2.31三个版本libc利用setcontext的区别，2.27无需gadget，2.29和2.31都需要gadget设置rdx，而因为2.29的gadget很简单并不需要进行过大调整，只需要将最后的stack chunk适配gadget即可；2.31则较为特别，经过多次调试后进行较多调整完成攻击，故记录下：

首先前半部分攻击[tcache\_perthread\_struct](https://ctf-wiki.org/pwn/linux/user-mode/heap/ptmalloc2/implementation/tcache/" \l "tcache_perthread_struct)完成泄露libc的部分要进行修改，原本的payload会造成count区域乱码，下面是通过写入0x1 0x2 0x3 0x4 0x5 0x6的调试，可以发现本来counts应该是0x1 0x2 0x3 0x4 0x5 0x6但实际上不是



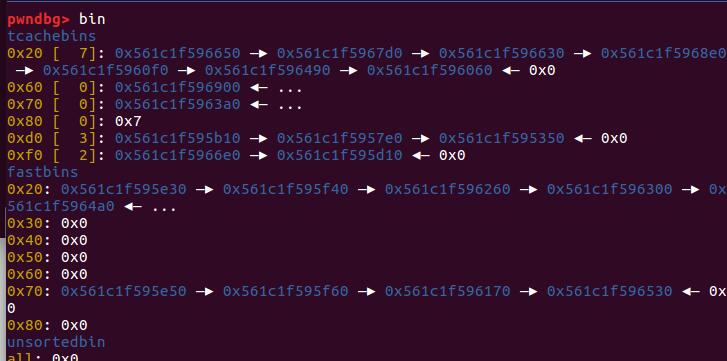
本来都一筹莫展找不到原因，后来突然想起啦看一下源码发现了问题所在，在2.31中counts里的元素是16位也就是两个字节，以前调试的2.27和2.29都是char类型也就是一个字节，所以原本的\x07要换成\x07\x00才行



修改后为：

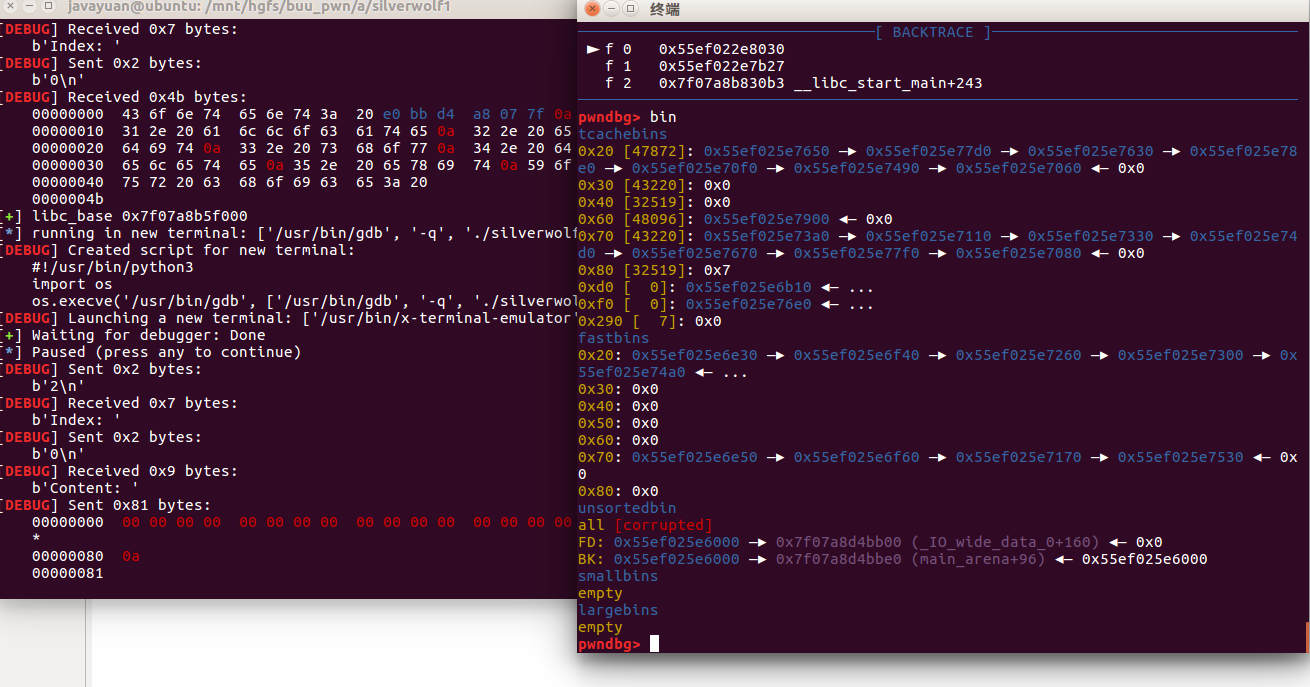


接下来这个问题根本原因其实也是上面这个问题的答案，即在再次污染[tcache\_perthread\_struct](https://ctf-wiki.org/pwn/linux/user-mode/heap/ptmalloc2/implementation/tcache/" \l "tcache_perthread_struct)完成tcache设置的时候因为count区域增加为原本的两倍长所以0x78的大小写不下完整的payload，payload在edit的时候被截断（本题限制了size大小），本来一开始意识到这点的时候我的第一反应就是edit完后再次add从unsorted bin中分割出后面的[tcache\_perthread\_struct](https://ctf-wiki.org/pwn/linux/user-mode/heap/ptmalloc2/implementation/tcache/" \l "tcache_perthread_struct)继续污染，但如果直接利用未置零的指针edit而后再次add的时候发现此时unsorted bin中没有chunk导致程序崩溃



但如果先add回次指针即可正常分割unsorted bin，一开始还是百思不得其解，还是动调调试发现了原因，下面这张图是edit后read了第一个字节\x00后的bin图，可以发现unsorted bin 报错，原因是unsorted bin 正常指向了[tcache\_perthread\_struct](https://ctf-wiki.org/pwn/linux/user-mode/heap/ptmalloc2/implementation/tcache/" \l "tcache_perthread_struct)，但

[tcache\_perthread\_struct](https://ctf-wiki.org/pwn/linux/user-mode/heap/ptmalloc2/implementation/tcache/" \l "tcache_perthread_struct)作为unsorted bin chunk 的fd指针被我们修改成了\x00，导致这个链表结构被破坏了以至于我们找不到unsorted bin chunk

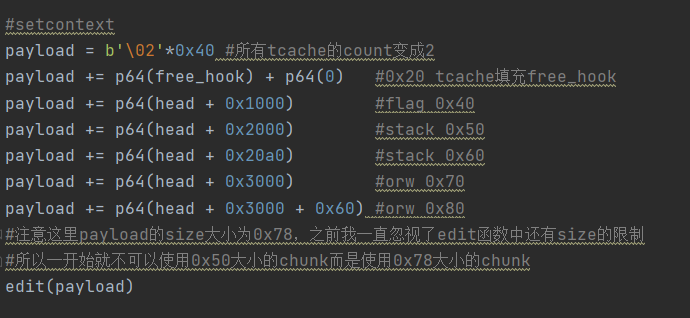


知道了原因就好办了，接下来就是先add再add已到达分割unsorted bin chunk的目的，但是这里就需要动调了，因为这里add的大小可不是随便选择的，我们查看下add之前的tcache，可以发现从0x20到0x80(这就是我们能申请的size大小)中只有0x50这个size的tcache没有值，也就是说只有申请0x50大小的chunk不会从tcache中分配而是切割unsorted bin chunk来分配chunk。



另外在第一次分配chunk的时候还要为第二次和有可能的第三次chunk的size的tcache的count覆盖为0以便继续从unsorted bin chunk中来切割。

这里就要提一下第二个chunk怎么确定size了，我们看一下我们原本的payload，长度为0x40+0x38,而现在变成0x40\*2+0x38,而本来的0x38中其实是有个p64(0)也就是0x30的tcache我们并没有用到，在2.27和2.29中我们不在意，但是2.31里能分配的空间很有限，我们需要关注每一块能利用的空间。

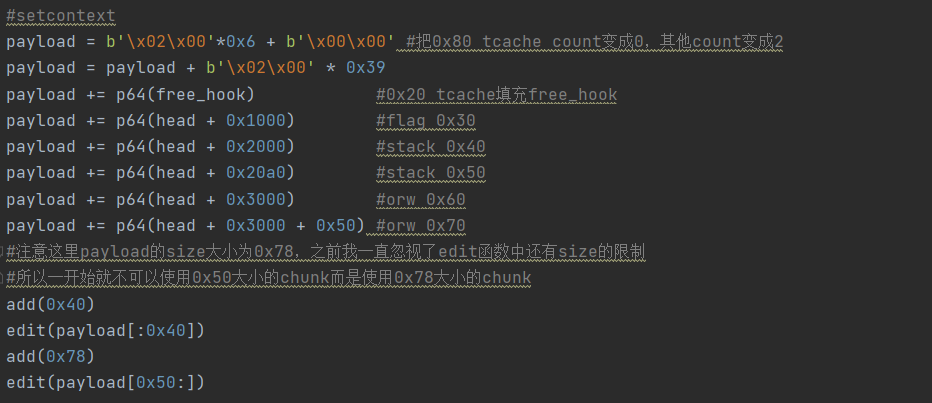


首先我们使用0x50的tcache第一次切割unsorted bin chunk，将前0x40的payload写入，剩下0x40 + 0x38的payload。

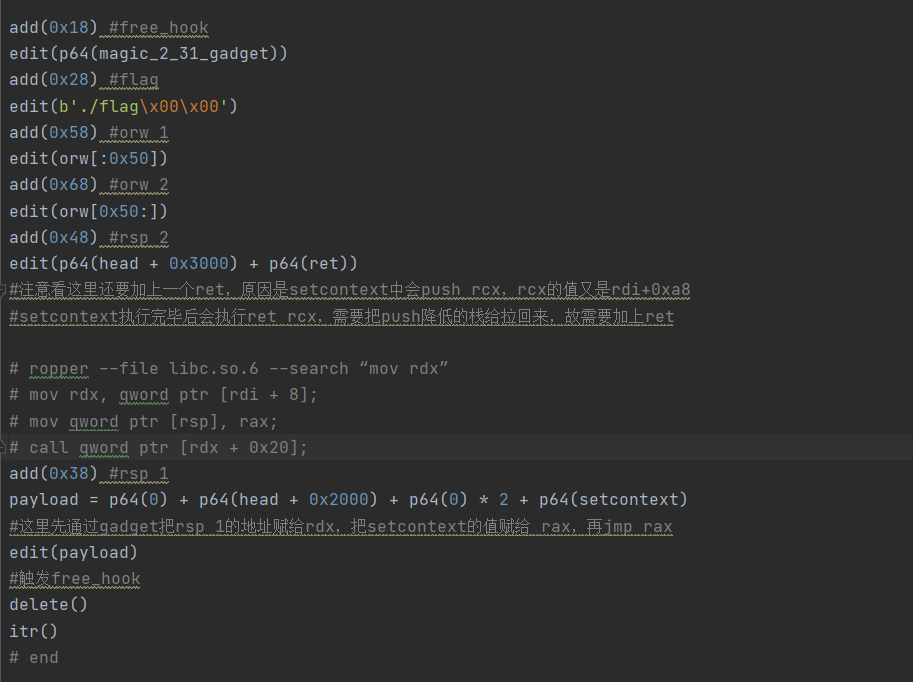
我们在新分配一个chunk（即第二次切割时）的时候，其prev\_size和size各占8个字节，因为上面payload剩下的0x40部分是一些非常大tcache的counts，故我们不用关心prev\_size和size占据了0x10字节，只剩下了0x30 + 0x38的payload需要写入，所以需要利用0x70或0x80的chunk进行写入

注意我们上面有提到第一次分配chunk的时候还要为第二次和有可能的第三次chunk的size的tcache的count覆盖为0以便继续从unsorted bin chunk中来切割，这里我们采用0x80的chunk来写入第二次切割unsorted bin chunk，这样一来0x80的tcache的count需要我们在第一次将其覆盖为0，故orw不能再使用0x80大小的tcache，我们依次将其size前移即可

具体payload如下：



最后的分配攻击的size也需要进行相应的修改



最后是简单的gadget适配，这部分和2.29类似，上面已经把payload和对应的注释给出来了就不再赘述