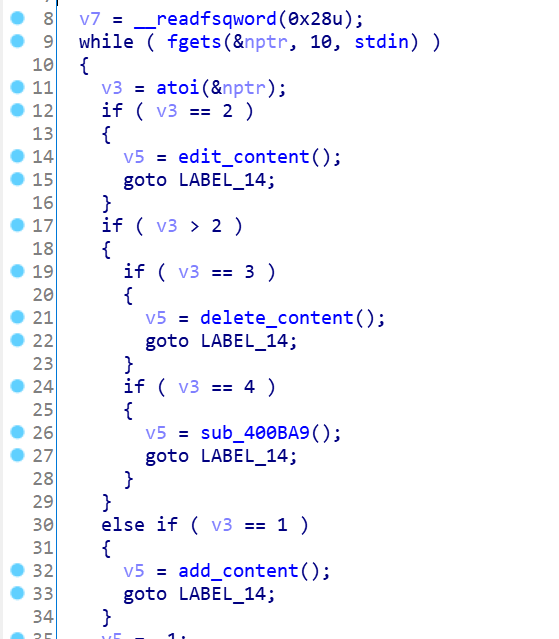


程序没有提示，但是呢，也是一个菜单题



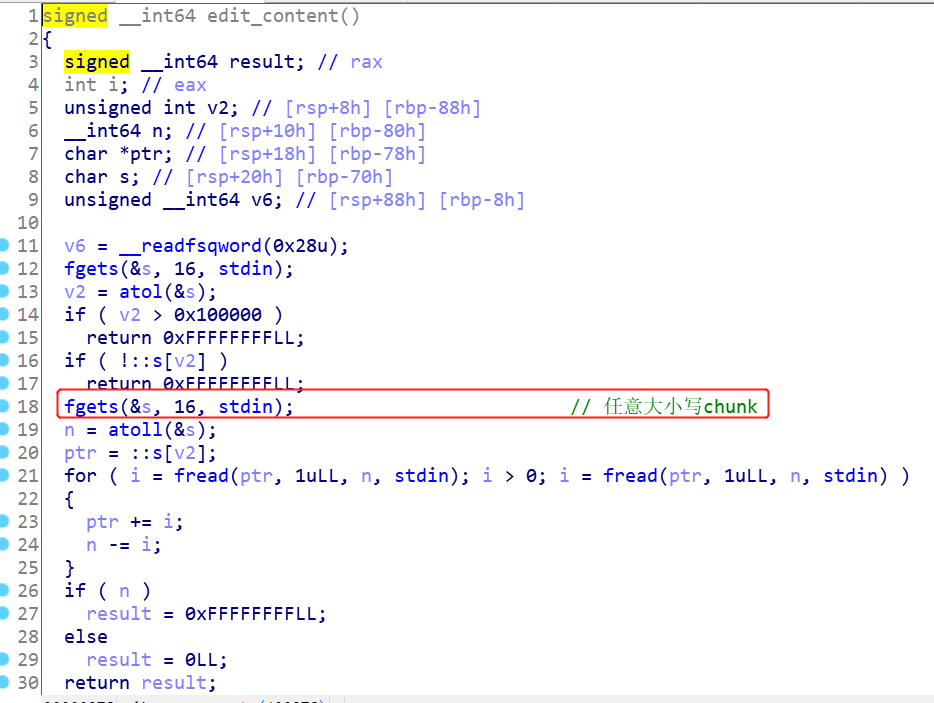
1 add

2 edit

3 delete

4 这个没啥用

然后，发现漏洞在：



可以任意大小写chunk。

程序没有可以打印的函数，因此，考虑利用unlink将某一些函数的got地址改为puts的plt地址。用来实现打印功能。

这里需要复习一下unlink的过程。

FD = P->fd

BK = P->bk

FD->bk = BK

BK->fd = FD

这里需要绕过校验：

FD->bk == P and BK->fd == P

假如有个指针ptr指向P(amd64下)。

可以覆盖P的fd = &ptr – 0x18

P的bk = &ptr – 0x10

然后想办法触发对P的unlink。可以使得ptr指向&ptr-0x18。

如果是用的一个数组来管理堆指针，那么可以修改堆指针数组某一个指针指向自己的低0x18字节，那么就可以控制前两个堆的指针指向。结合edit功能，可以泄露地址，或者往地址里面写入内容。

本题就是这样，具体思路如下：

1. 首先伪造unlink

2. 然后修改前两块堆指针分别为got[‘free’]和got[‘atoi’]

3. 修改got[‘free’]的地址为puts@plt地址。

4. delete前一块堆，这样就会打印出atio的实际地址，可以计算出system地址

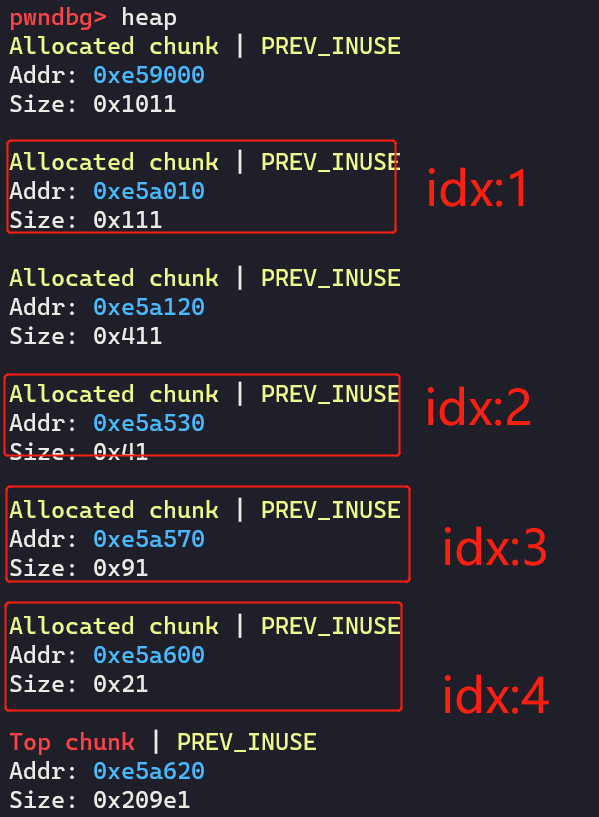
5. 这里需要注意，deete会置为0，所以要重复一下第2步

6. 回到main函数，输入/bin/sh;就能出发system(/bin/sh\x00)

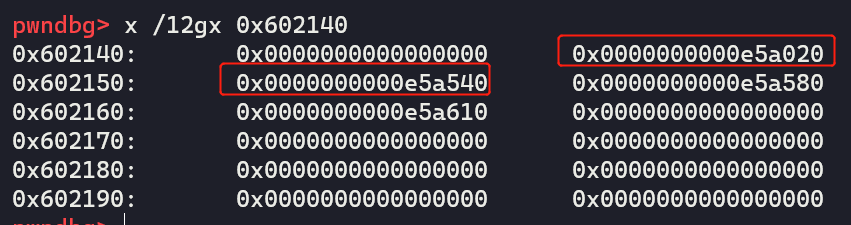
调试过程如下：



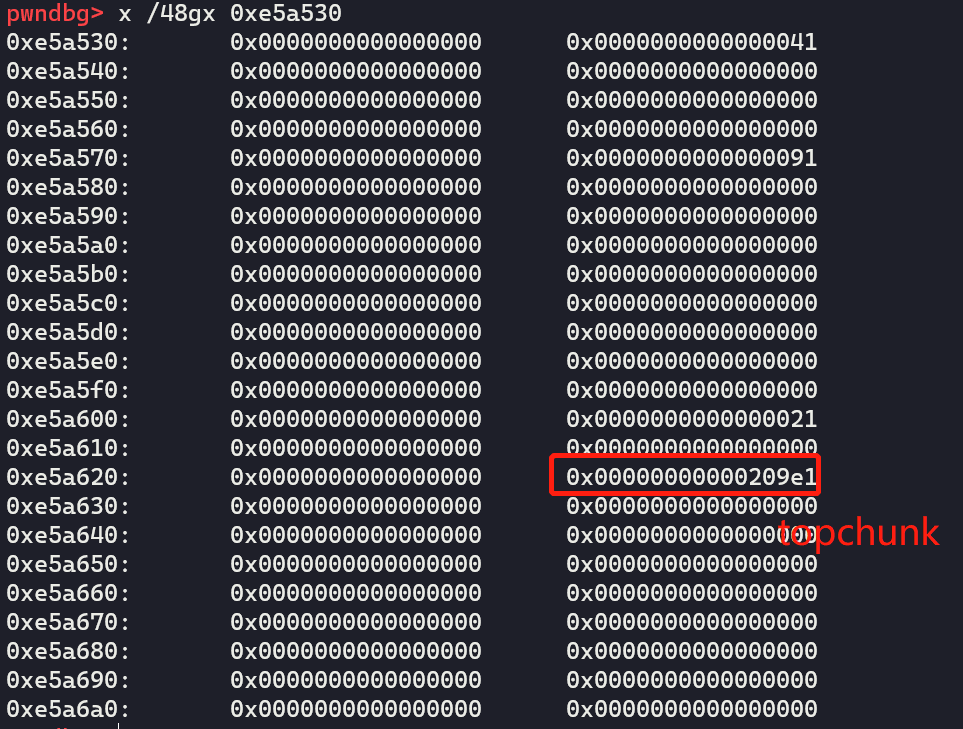
这里要注意，程序没有setbuf，所以需要手动出发一下缓冲区分配。此时的堆：



然后看下堆数组：

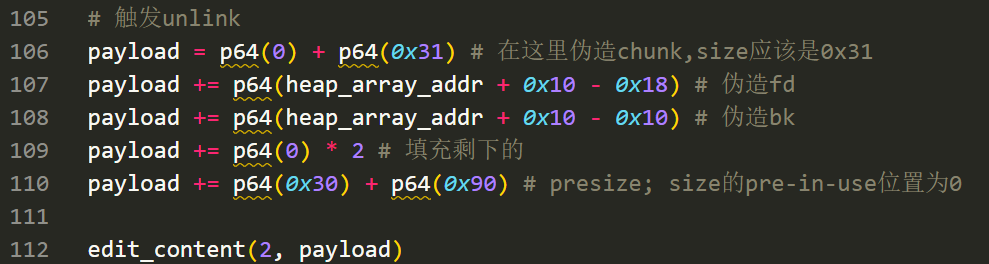


此时的堆内容：

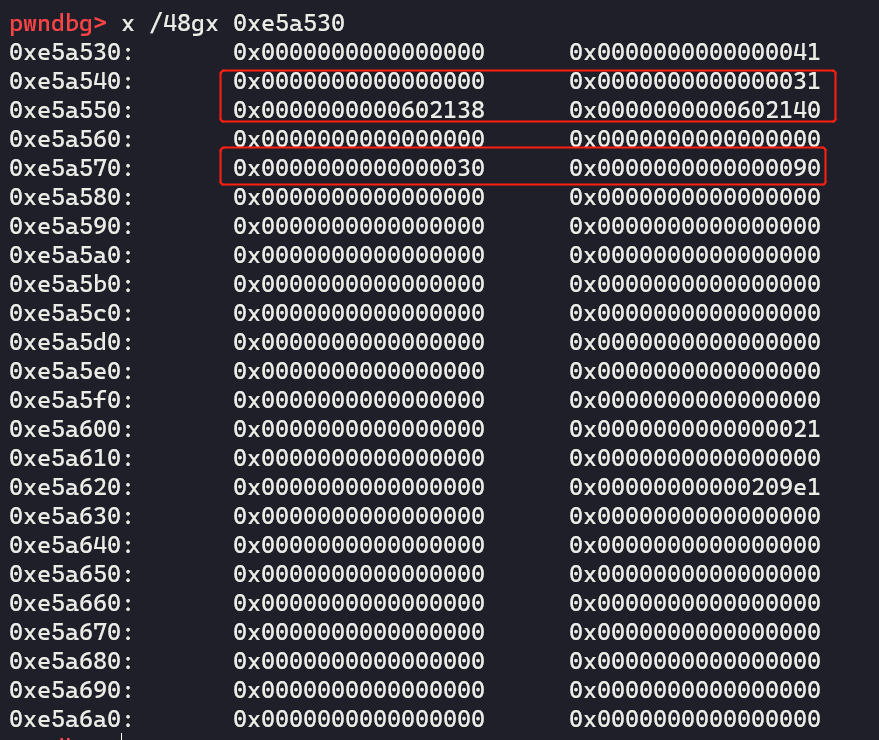


是可以对应起来的。

然后edit功能修改一下堆内容：

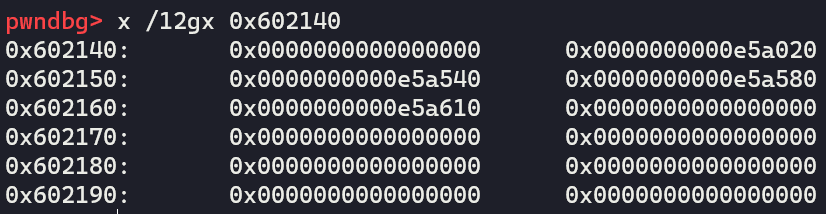


修改后：

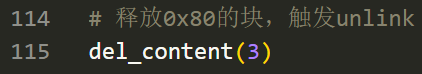


这个时候，释放0xe5a680处的chunk就会让0xe5a550的chunk触发unlink。

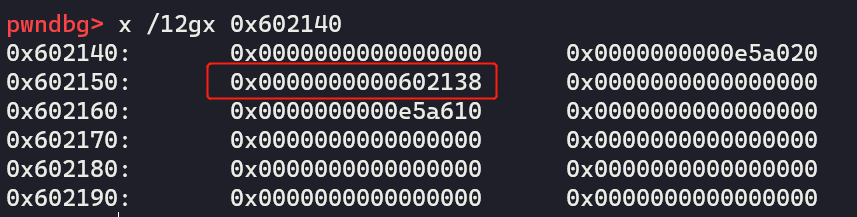
再看一下堆数组：



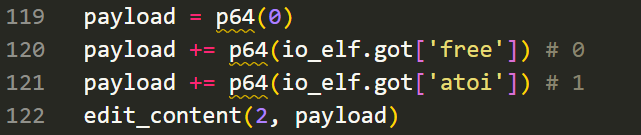
然后，释放：



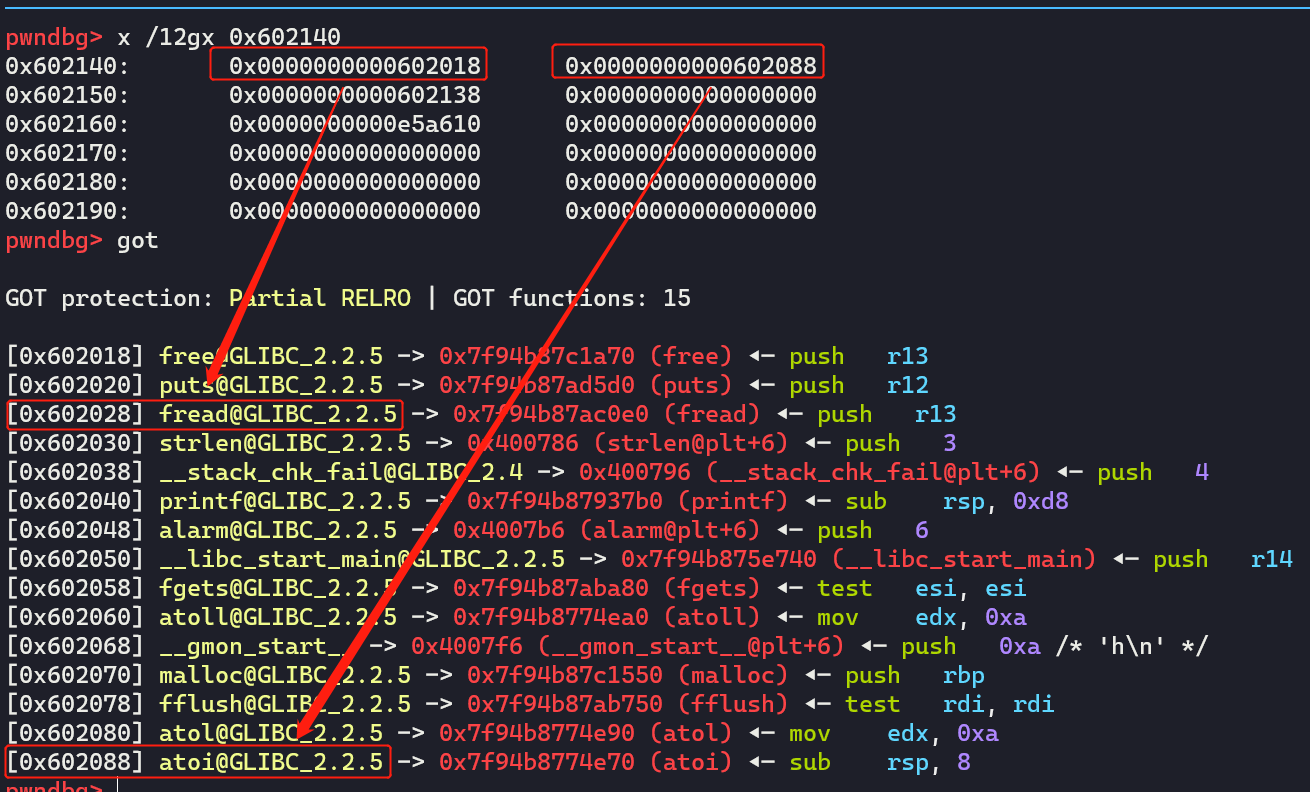
此时：



这个时候，我们可以去修改0x602140 和0x602148处的地址。

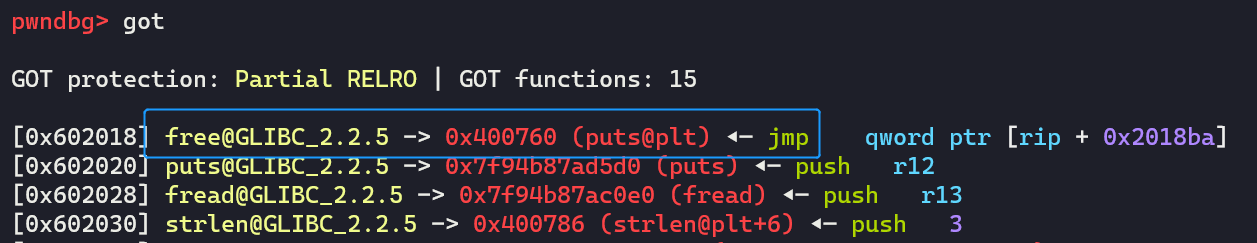


这个时候：



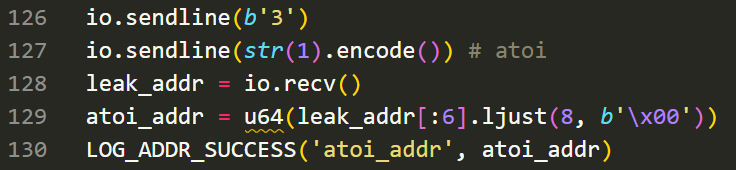
然后利用edit功能：



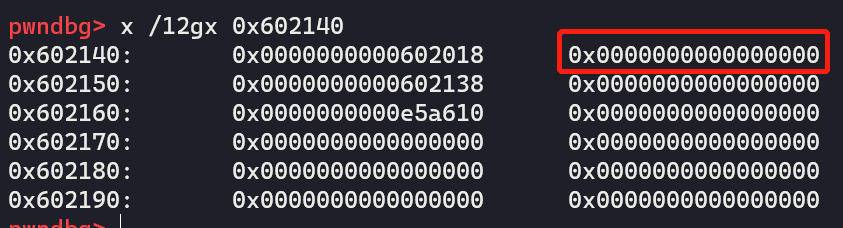


这个时候，free就变成了puts。

泄露地址：



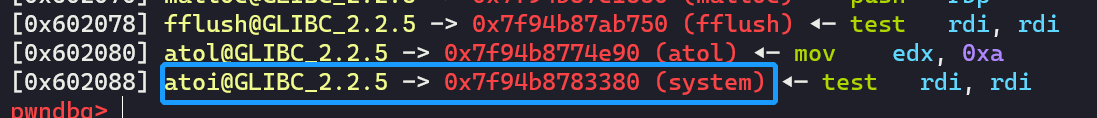
然后去看一下堆数组：



重新填上并修改atoi为system

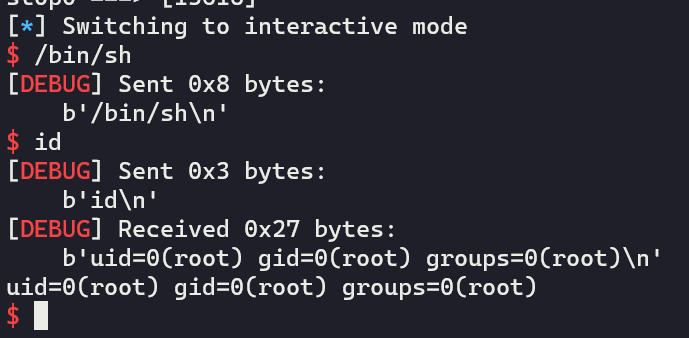


看一下got：



OK，修改成功！

接下来只需要输入/bin/sh就可以了。



成功拿到shell！