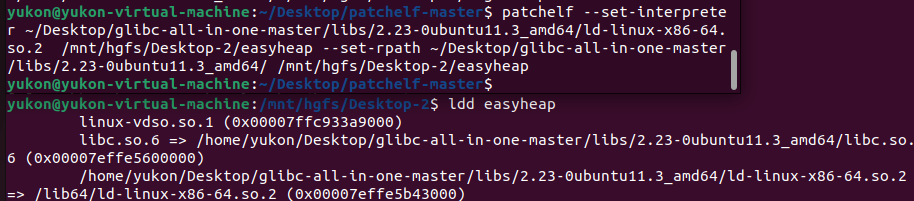
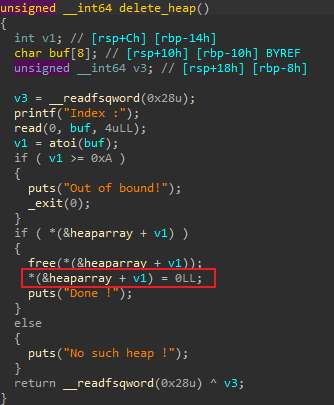
buu



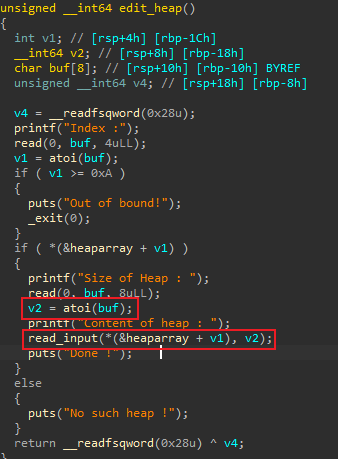
先根据给的ubuntu版本patchelf



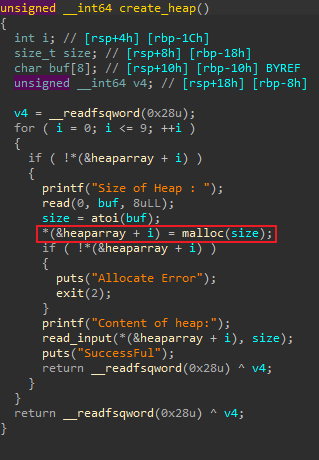
看ida：

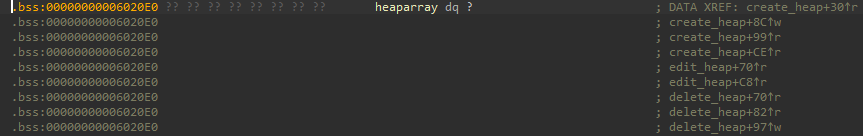


无uaf

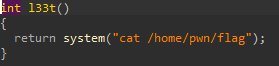


edit的字节数可以自定义，存在堆溢出





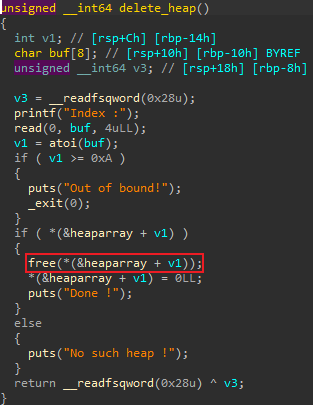
heaparray存放在bss段



有后门，但是实际上没有这个文件，但是这个函数提供了system的got表和plt表，

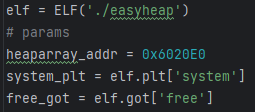
**思路：**

可以利用system\_plt修改free的got表



并通过修改heaparray+v1位置的值为’/bin/sh’从而控制free(heaparray+v1)为system(‘/bin/sh’)从而getshell

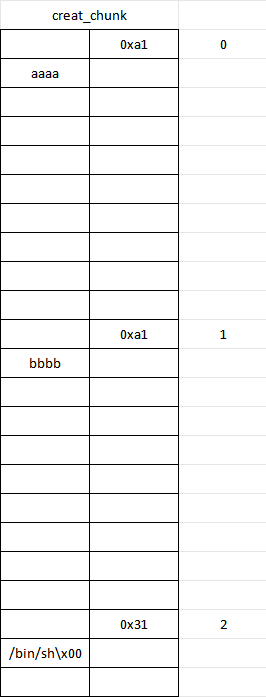
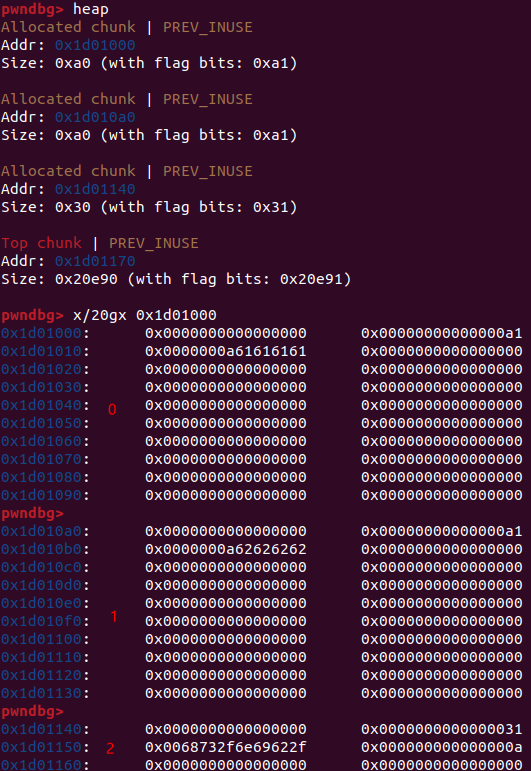
实现：



找好必要的参数

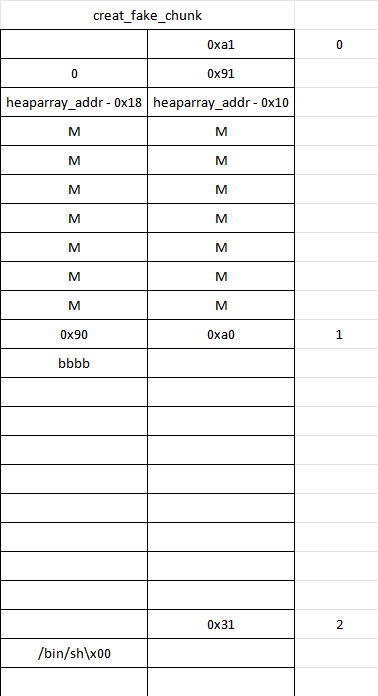
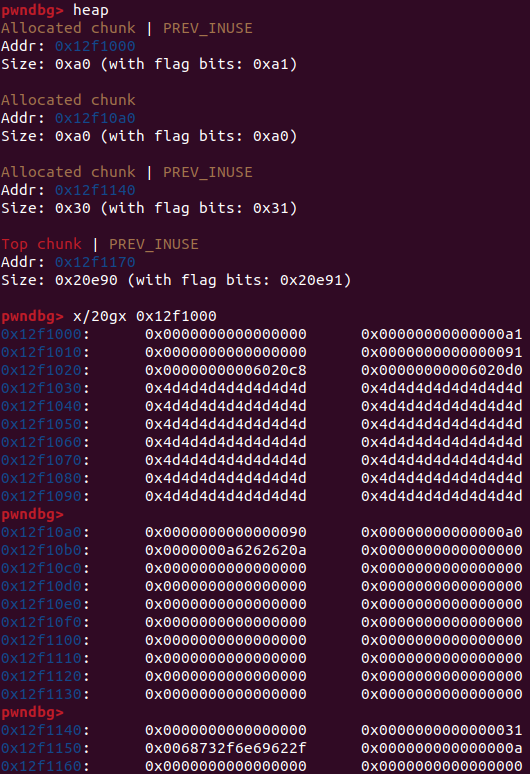


创建三个堆0,1,2



栈视图如上

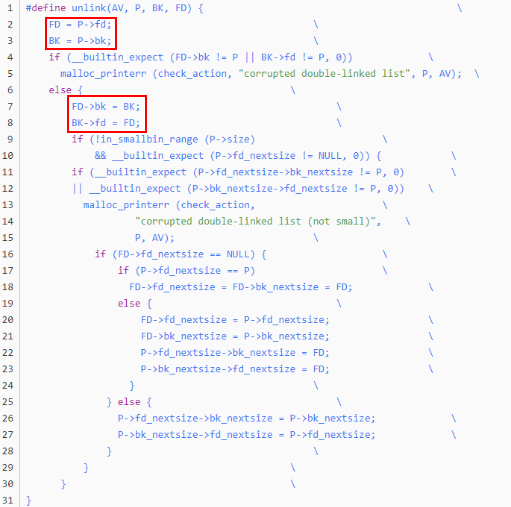
伪造一个fake\_chunk(从0，0x91开始，也就是chunk0的下一行开始)：



注意，这里不光伪造了chunk0，还对chunk1的p标志位进行了修改，将0xa1改为了0xa0，即将p从1置为0，并将chunk1的prev\_size置为0x91，即，从而便于待会儿free掉chunk1的时候将chunk1与伪造后的chunk0中的fakechunk合并

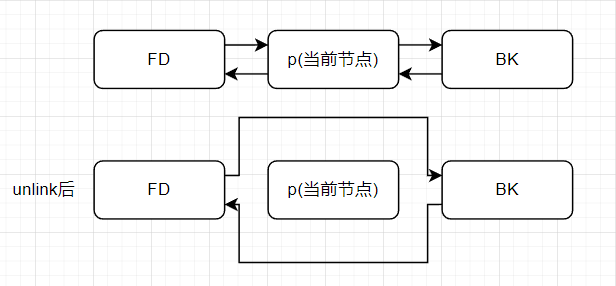
参照源码：

先看unlink

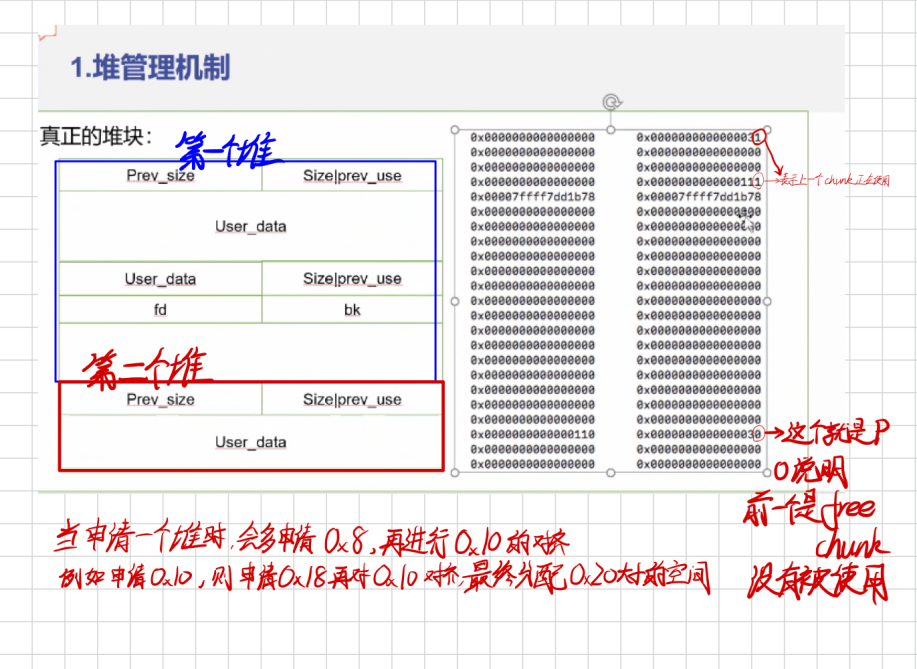


经典的脱链操作，其他暂时不看（懒）

相当于：



顺便简单回顾一下堆的基本结构



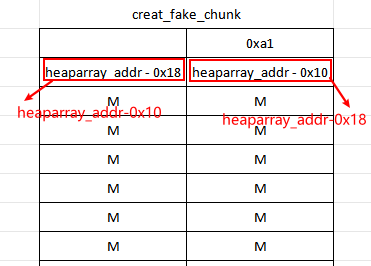
所以改了prev\_inuse为0后就会判断前一个没有被使用，至于为什么会合并，再看源码：



\*\*这里向后合并也可以说是向低位屏蔽，意会即可\*\*

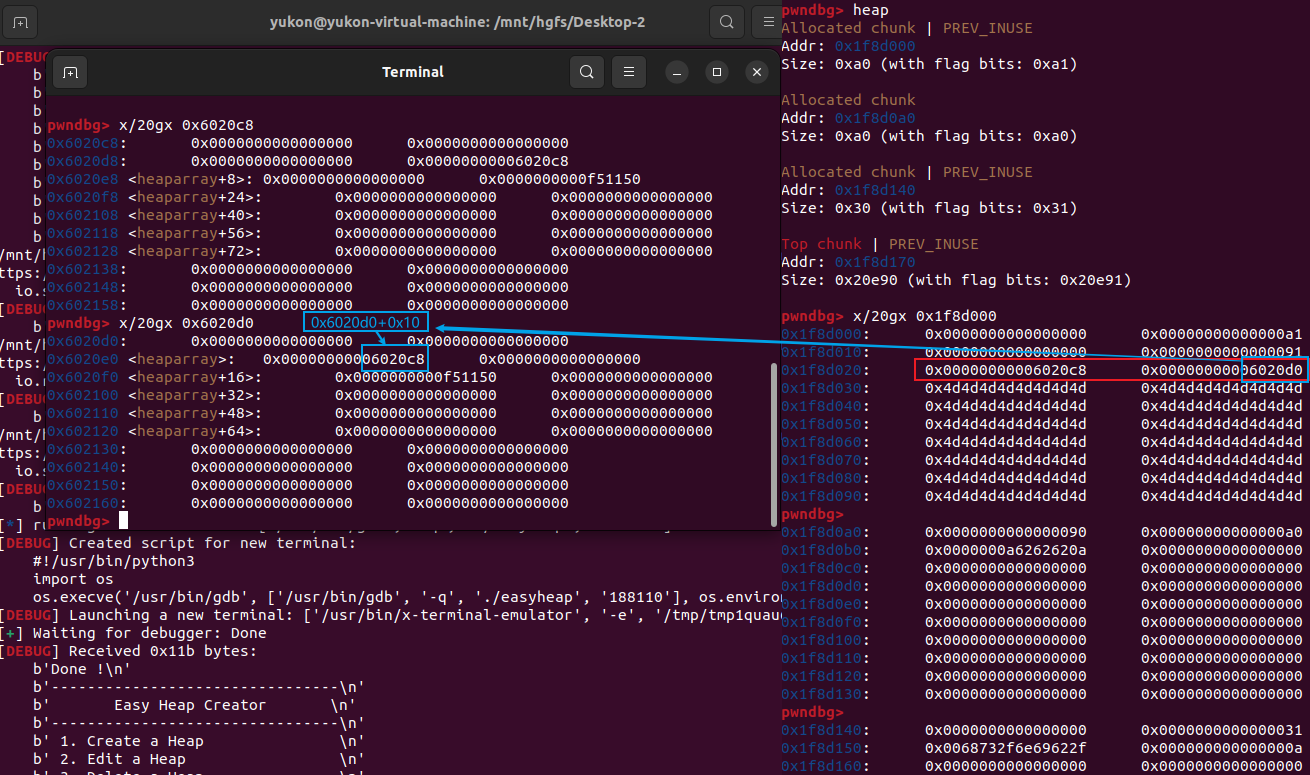
重要的是合并的时候的unlink这里

指针p现在指向chunk0，所以unlink后chunk0的FD的bk指向了chunk0的bk和fd，也就是将heaparray\_addr-0x18地址的值修改为heaparray\_addr-0x10，并将heaparray\_addr-0x18地址的值修改为heaparray\_addr-0x18：



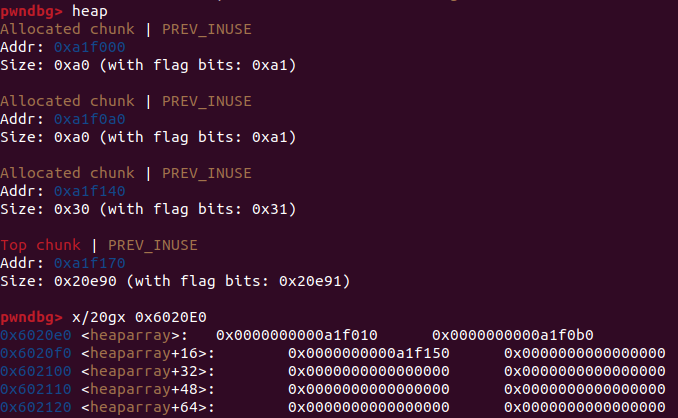
Gdb看：

我们先看其中BK的修改情况，也就是bk+0x10=6020d0+0x10地址的值改为了0x6020c8(下图左边为delete(1)后，右图为delete(1)前)（为啥是+0x10啊？）

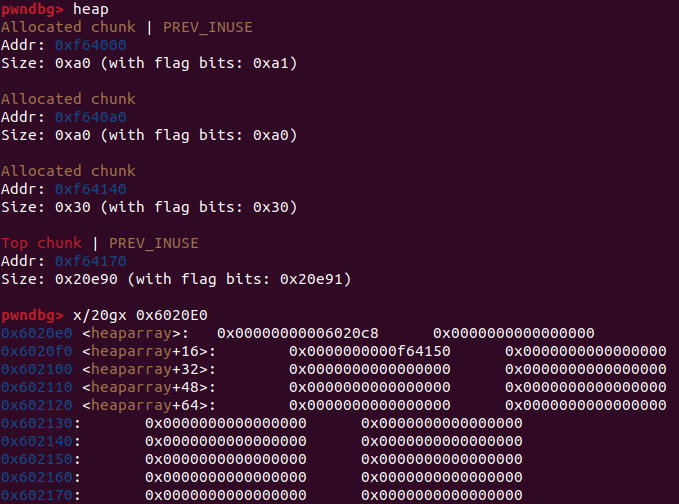


而bk+0x10，即0x6020d0+0x10，也即heaparray-0x10+0x10，就是heaparray的值，所以我们这样修改完之后就可以通过修改heaparray+0来修改bk+0x10的值，我们将bk+0x10的值修改为free的got表值

对比一下heaparray，一开始：



delete(1)之后：

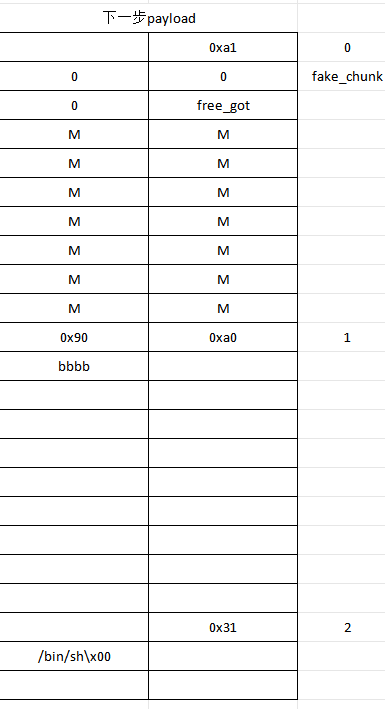


再看“下一步payload”：

现在因为heaparray+0=0x6020c8，所以填充0x18个无效字符就能到达heaparray(0x6020E0-0x6020c8=0x18)



edit后：



即fake\_chunk的bk指针(heaparray)现在指向了free\_got的位置

所以我们现在对chunk0进行edit就能编辑heaparray+0=free\_got(0x602018)了

got表和plt表不了解可以硬记为：要将一个函数改为另一个函数，就将这个函数的got表改为另一个函数的plt表即可

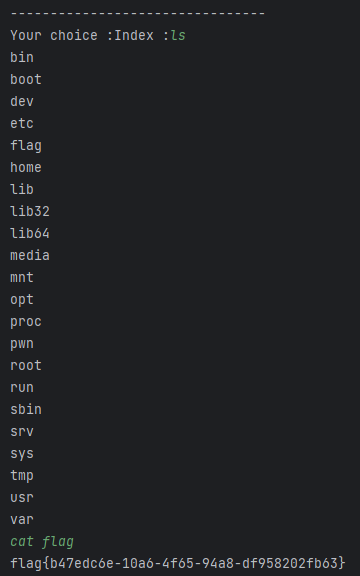
所以有最后一步脚本：



将free\_got(0x602018)改为了system\_plt

然后再在Delete(2)里调用了free(heaparray+chunk2\_offset)也就是system(‘/bin/sh’)

成功getshell



完整exp：

