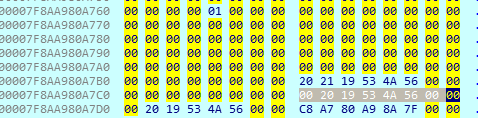
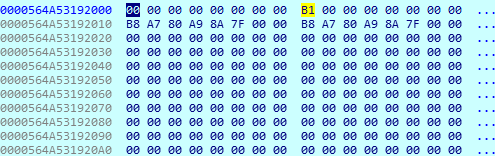
-3》unsort bin attack：在malloc的过程中，unsortbin会从链表上卸下来（只要分配的大小不是fastchunk大小），会把bk+0x10的地方写入本unsort\_bin的地址。（实际调试中发现，写入的是topchunk指针的地址，main\_arena+0x58处）

-2》把0xb0大小的chunk free掉后，其确实挂在了unsort bin：

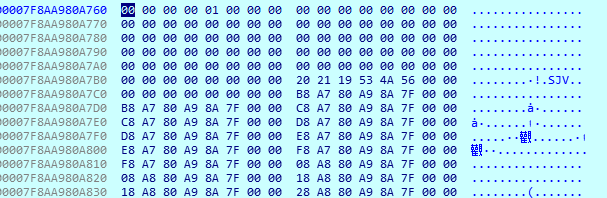


但是，刚free后好像不是双向循环链表，因为空闲的大小0xb0chunk的fd和bk指向的是top chun指针的地址：

非多线程时，malloc\_state里好像没有mutex，且只有一个空闲的unsort\_chunk时，该chunk的fd/bk指向topchunk。即main\_arena+0x58.

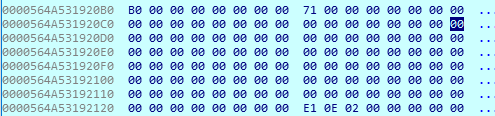


当空闲的0xb0chunk被重新malloc后，unsort bin指向topchunk指针的地址：

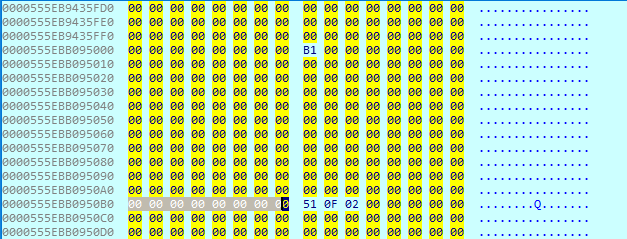


Fast chunk被free后：

空闲chunk挂到对应的fastbin，最后一个fastchunk指向0（unsortbin指向的是topchunk）：



-1》topchunk头部好像是0x10，那么前8字节一直是0咯



0》puts()函数的作用与语句printf(“%s\n“,s);的作用形同。

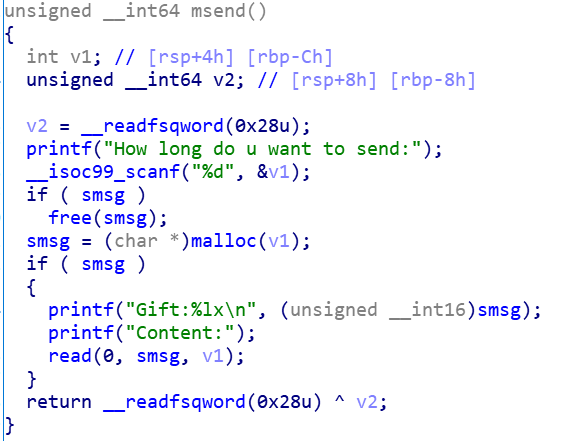
1》extern int strcmp(const char \*s1,const char \*s2);

等于返回零，小于返回负数，大于返回正数。

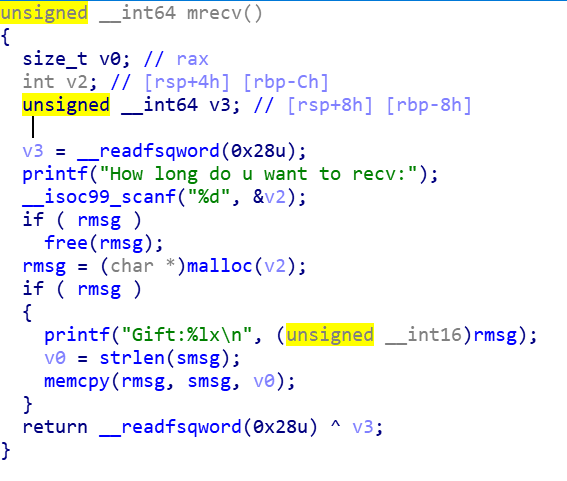
2》从文件读取密码的爆破思路：

当连续open文件1024次上后，使文件描述符数组满了后，无法再打开文件，即读取到的密码为空。这样就可以绕过密码。

3》程序功能：



Send：psend指针处有值，则free掉他；然后申请自定义大小的chunk，并可填入内容。



Recv功能：若指针处有值，则free掉。之后malloc一个任意大小的堆块填充至指针；并将send功能指针指向的chunk内容通过strlen和memcpy拷贝到recv功能指针指向堆块。

（漏洞点：这里依据psend指针内容的长度而不是recv自己分配chunk的大小来recv内容，存在溢出。

信息泄漏：（过大）当psend内容为0时，泄漏fd(unsort\_bin\_addr)、bk(unsort\_bin\_addr)地址；

漏洞利用：（过小）当psend内容大于recv\_chunk时，则会溢出到下个chunk。

）

**信息泄漏**

Send(0xa0,’a’)

[0x0,0xa0,’a’] [top chunk]

Recv(0x60)

[0x0,0xa0,’a’][0x0,0x60,’a’][top chunk]

send(0xa0,'1234567')

需满足连接起来后续的8个字节（避免strlen()取长度时被00截断），注意一般地址为6个字节。

[0x0,0xa0, fd(unsort\_bin) bk(unsort\_bin)][0x0,0x60,’a’][top chunk]

Encode(8)

对fd(unsort bin)加密

按实际大小加密

recv(0x60)//转移到recv指针处

[0x0,0xa0, fd(unsort\_bin) bk(unsort\_bin)][0x0,0x60, fd(unsort\_bin) bk(unsort\_bin)][top chunk]

p.writeline('decode')

p.readuntil(':')

p.writeline('16')

p.readuntil('\n')//选择解密

按实际大小解密

libc=u64(p.readuntil('\n')[:-1]+chr(0)\*2)-0x3c2760-0x58

提取解密后的数据

**漏洞利用**

考虑向哪里写入one\_gadget实现利用

初始状态：

Chunk a//send

Chunk b//recv

我们可以通过写入a覆盖b的信息。//可以写入precv指针过长的psend指针内容来实现。

步骤一：申请一个大的send

chunk a//空（free掉）

chunk b//recv

chunk c//send

步骤二：recv功能占位a并recv溢出到b的fd指针

chunk a// recv

chunk b//空（free掉) size 0x71（溢出fd指针指向malloc\_hook-0x23）

chunk c//send

步骤三：之后通过send和recv功能各malloc一个0x60的堆即可完成利用。

注意：

此时不知为何free c因和topchunk合并调用malloc\_consolidate产生异常（最后0x60字节利用漏洞时）。

于是，通过覆盖a的同时覆盖到c。伪造一个chunk和top\_chunk相连。

步骤四：

通过send和recv功能各申请一个0x60的堆块，即可也如malloc\_hook。并跳转到one\_gadget。

Recv前：

[0x10+0xa0]

[0x10+0x60]

[0x10+0x150]

Recv把[0x10+0x150]内容覆盖到[0x10+0xa0]，并溢出到[0x10+0x60]

目的是修改[0x10+0x60]的fd指针为p64(libc+elf.symbols['\_\_malloc\_hook']-0x23)；

然后，申请两次fastchunk，第二次申请到malloc\_hook的位置，此时即可修改其内容为one\_gadget了。

注意：

Malloc\_hook的使用是二阶的。