**示例4-4：**

|  |
| --- |
| #include <windows.h>  main()  {      HLOCAL h1, h2,h3,h4,h5,h6;      HANDLE hp;      hp = HeapCreate(0,0x1000,0x10000); //创建自主管理的堆      h1 = HeapAlloc(hp,HEAP\_ZERO\_MEMORY,8);//从堆里申请空间      h2 = HeapAlloc(hp,HEAP\_ZERO\_MEMORY,8);      h3 = HeapAlloc(hp,HEAP\_ZERO\_MEMORY,8);      h4 = HeapAlloc(hp,HEAP\_ZERO\_MEMORY,8);      h5 = HeapAlloc(hp,HEAP\_ZERO\_MEMORY,8);      h6 = HeapAlloc(hp,HEAP\_ZERO\_MEMORY,8);      \_asm int 3  //手动增加的int3中断指令，会让调试器在此处中断      //依次释放奇数堆块，避免堆块合并      HeapFree(hp,0,h1); //释放堆块      HeapFree(hp,0,h3);      HeapFree(hp,0,h5); //现在freelist[2]有3个元素      h1 = HeapAlloc(hp,HEAP\_ZERO\_MEMORY,8);      return 0;  } |

整个流程解析：

（1） 程序首先创建了一个大小为 0x1000 的堆区，并从其中连续申请了6个块身大小为 8 字节的堆块，加上块首实际上是6个16字节的堆块。

（2）释放奇数次申请的堆块是为了防止堆块合并的发生。

（3）三次释放结束后，会形成三个16个字节的空闲堆块放入空表。因为是16个字节，所以会被依次放入freelist[2]所标识的空表，它们依次是h1、h3、h5。

（4）再次申请8字节的堆区内存，加上块首是16个字节，因此会从freelist[2]所标识的空表中摘取第一个空闲堆块出来，即h1。

（5）如果我们手动修改h1块首中的指针，应该能够观察到 DWORD SHOOT 的发生。

**接下来，我们通过调试程序来观察堆内存变化，调试手段为采用VC6自身的调试器，具体了解堆管理过程中的内存变化：**

**1）执行HeapFree(hp,0,h1)语句时**

hp为0x003a0000, h1为0x003a0688，根据堆块结构，我们知道h1堆块的块身起始位置为0x003a0688，块首起始位置为0x003a0680。观察该语句执行后，对应的内存变化，如下图4-10：

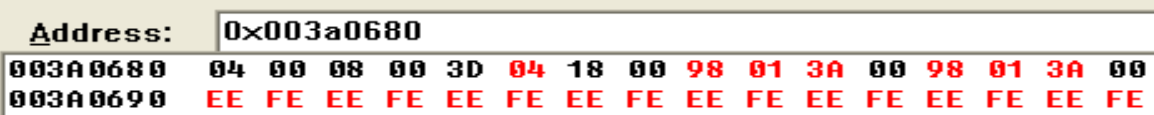


图4-10 0x003a0680处内存

可见，除了块首状态变化外，0x003a0688开始的块身位置的前8个字节（Flink和Blink）发生了变化，由0x000000变为具体的有效地址。注意到，这个是第一个16字节的堆块释放，将被链入到freelist[2]空表中，而此时Flink和Blink的值都是0x003a0198，也是freelist[2]的地址。我们转到0x003a0198处，观察内存为图4-11所示：

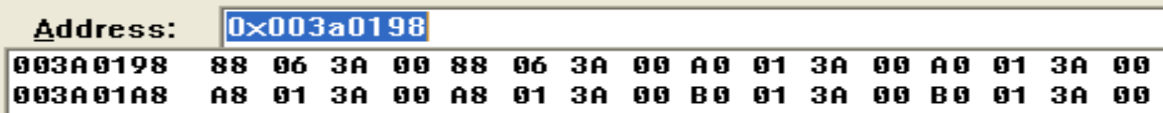


图4-11 0x003a0198处内存

可见，freelist[2]的Flink和Blink都是0x003a0688。意味着，当前freelist[2]唯一后继节点就是刚刚空闲的h1块（地址为0x003a0688），而h1块是唯一前继节点是freelist[2]。其它地址（freelist[3]、freelist[4]、freelist[5]）的Flink和Blink均指向自身，说明都是空表。

**2）依次执行HeapFree(hp,0,h3)和HeapFree(hp,0,h5)后**

可知，此时freelist[2]链表状态为：freelist[2]<=>h1<=>h3<=>h5。

**3）执行HeapAlloc(hp,HEAP\_ZERO\_MEMORY,8)语句时**

此时，当再次分配空间的时候，从freelist[2]的双向链表里摘下一块大小为16字节的堆块，首先摘得h1（地址为0x003a0688）。

观察此时：

（1）freelist[2]（地址为0x003a0198）所存储的信息为：Flink（前4个字节）为0x003a0688，Blink（后4个字节）为0x003a0708。

（2）h1（地址为0x003a0688）所存储的信息为：Flink为0x003a06c8，Blink为0x003a0198。

（3）h3（地址为0x003a06c8）所存储的信息为：Flink为0x003a0708，Blink为0x003a0688。

摘走h1之后，内存变为：

（4）freelist[2]（地址为0x003a0198）的**前4个字节变为**0x003a06c8，实际发生了将h1后向指针（值为0x003a0198）地址处的值写为h1前向指针的值。

（5）h3（地址为0x003a06c8）的Blink变为h1->Blink，即0x003a0198，实际发生了将h1前向指针（值为0x003a06c8）地址处的值写为h1后向指针的值。

**4）Dword Shoot攻击**

假设在执行该语句之前，h1的Flink和Blink被改写为特定地址和特定数值，那么就完成一次Dword Shoot攻击。

|  |
| --- |
| **注意：**在Windows XP以后的操作系统中，因为引入地址随机化等防护措施，使得此类的堆溢出Dowrd Shoot攻击变的越来越难。 |