《漏洞利用及渗透测试基础》实验报告

姓名：管昀玫 学号：2013750 班级：计科一班

**实验名称：**

堆溢出Dword Shoot模拟实验

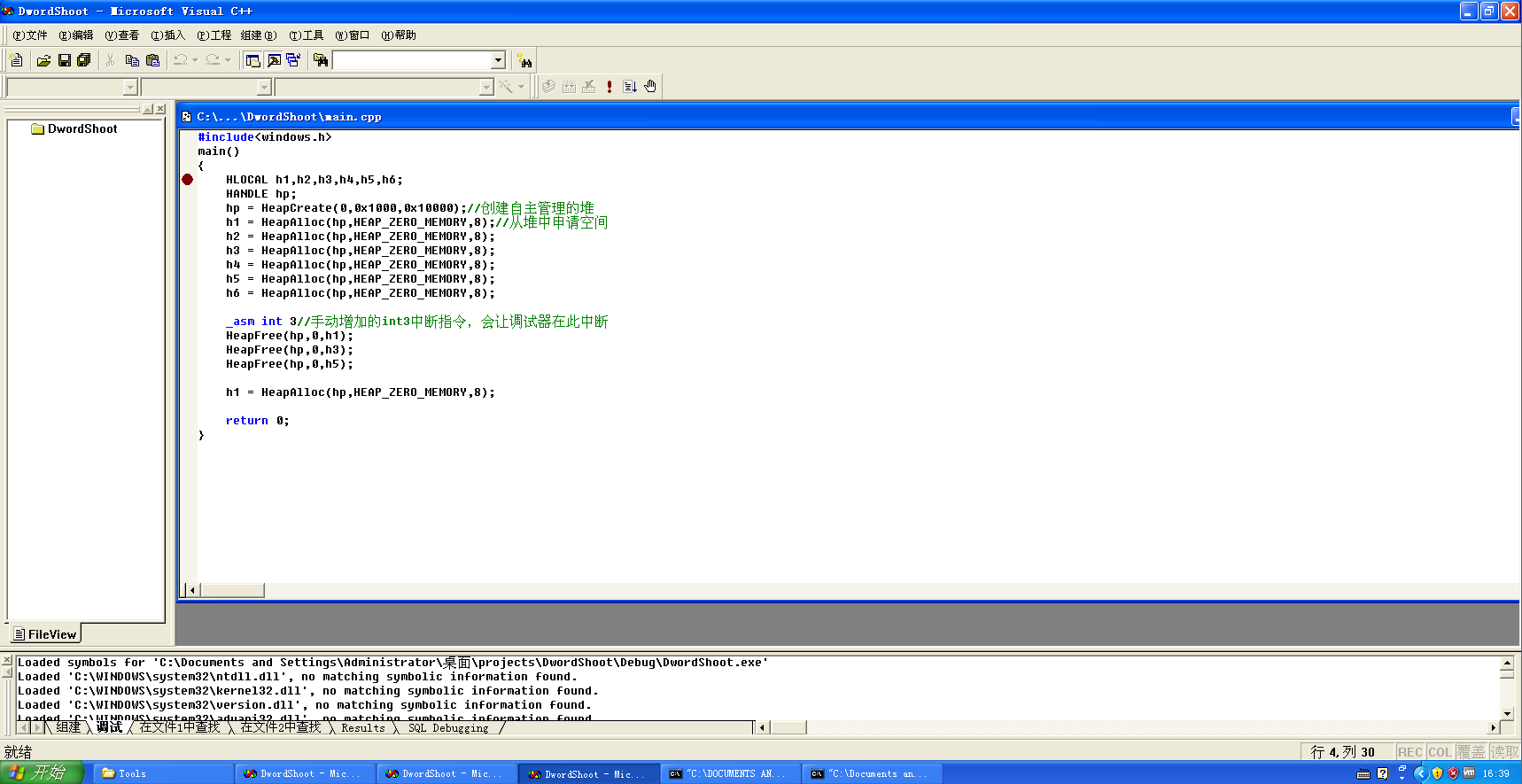
**实验要求：**

以第四章示例4-4代码为准，在VC IDE中进行调试，观察堆管理结构，记录Unlink节点时的双向空闲链表的状态变化，了解堆溢出漏洞下的Dword Shoot攻击。

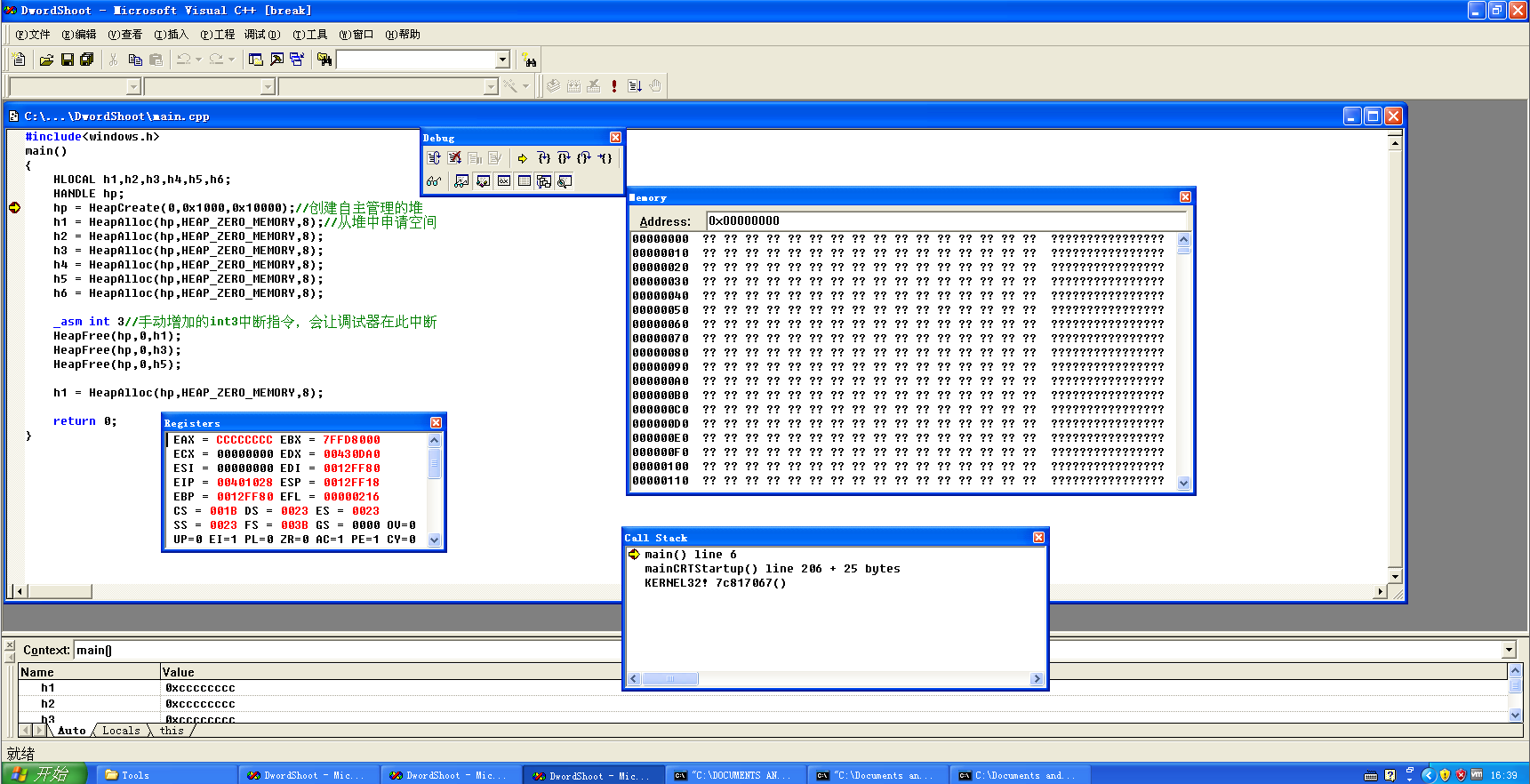
**实验过程：**

1. 软件准备

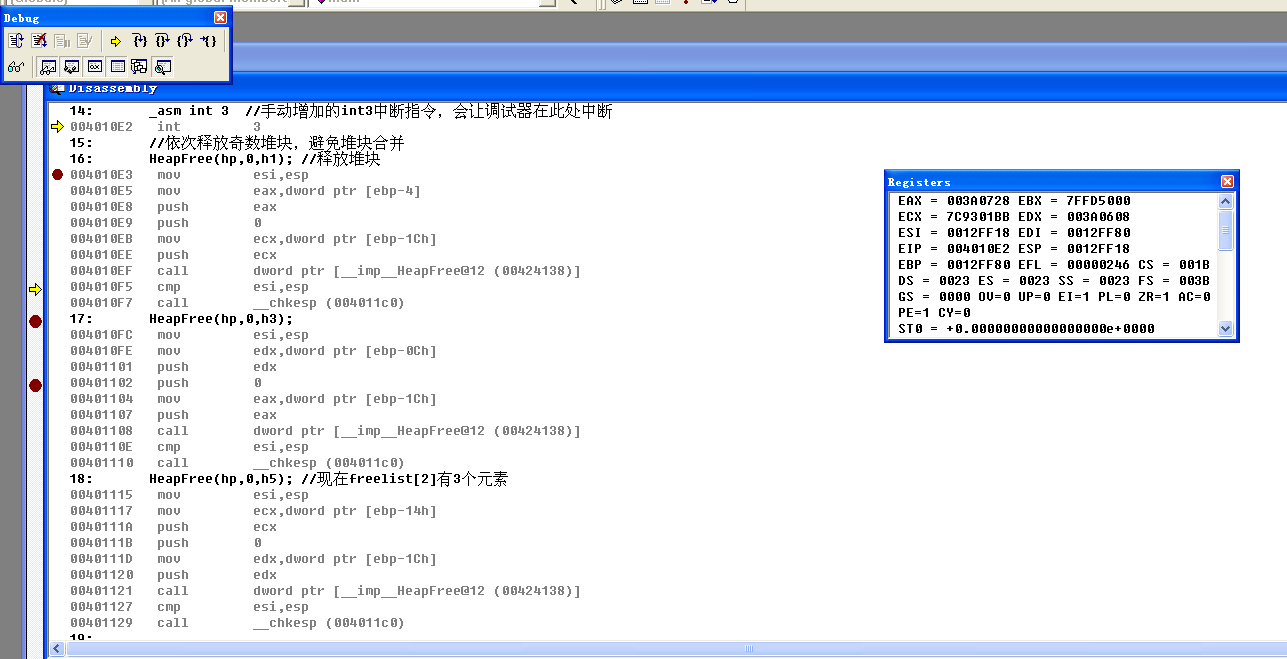
在VC6.0中编写代码，并设置断点



进入debug模式

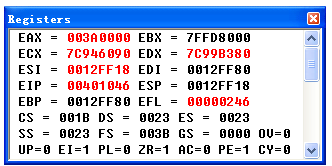


进入反汇编模式

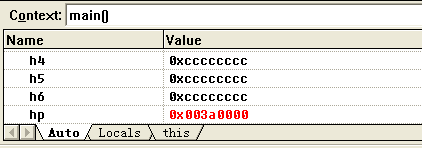


1. 流程解析

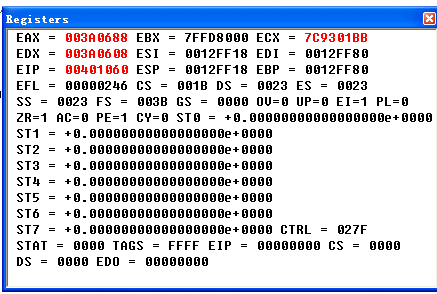
执行完HeapCreate指令后，寄存器状态如下

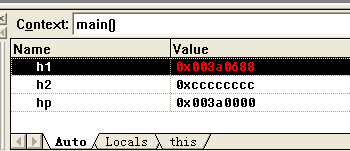


可以看出EAX变为003A0000，也就是HeapCreate的返回值，使得hp变成了003A0000



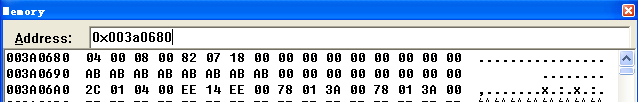
执行完h1 = HeapAlloc指令后，寄存器和变量状态如下：



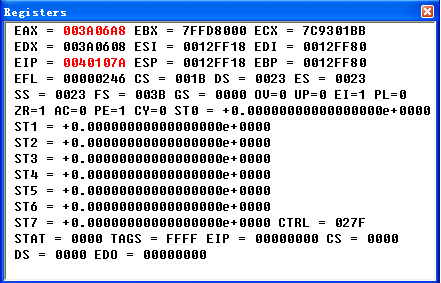


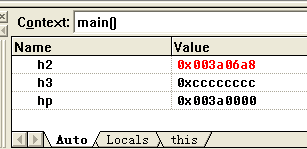
可以看出，此时h1里的值是003A0688，也就是h1这个块的块身起始地址是003A0688，块首的大小是8字节，如此计算出块首的起始地址应该是003A0680。

跳转到块首起始地址查看：

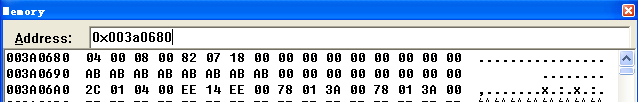


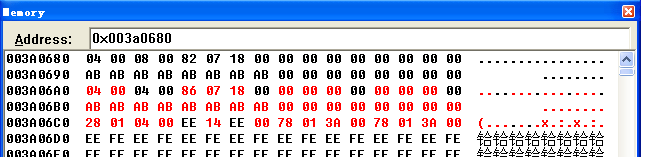
继续执行下一条指令，开辟第二个8字节大小（加上块首是16字节）的堆块：





为了明显起见，将申请堆块之前的图和申请之后的图放在一起作为一个对比

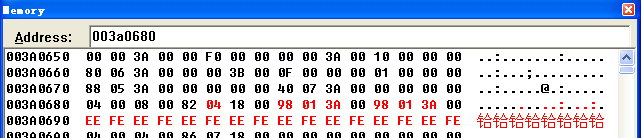




h2的块身地址是003a06a8，则块首地址是003a06a0；此时块首地址上存储的值变成了00 18 07 86 00 04 00 04，块首的状态发生了变化。之后从003a06a8开始，前8个字节全变成了0，被清空，进入到待存储的状态；

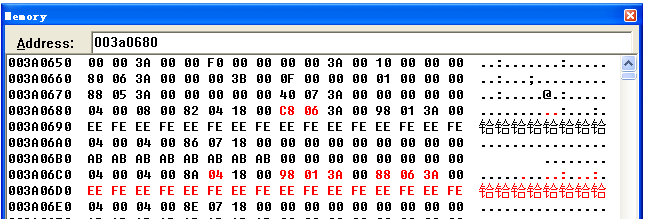
之后到h6的申请都是同理，下略。直接进入到free；

执行完HeapFree(hp,0,h1)之后，内存状态如下：



可以看出，块首的状态发生变化，其中的07变成04，应该是将堆块的状态从占有态变成空闲态；并且从003a68a地址开始，内部的存放的数据发生了变化：之前是没有存放任何数据的00 00 00 00 00 00 00 00，变成了如今的98 01 3a 00 98 01 3a 00，而这里现在存的就是Flink和Blink，在这里都指向了003a0198这个地址，也就是freelist[2]的地址；

执行完HeapFree(hp,0,h3)之后，内存状态如下：

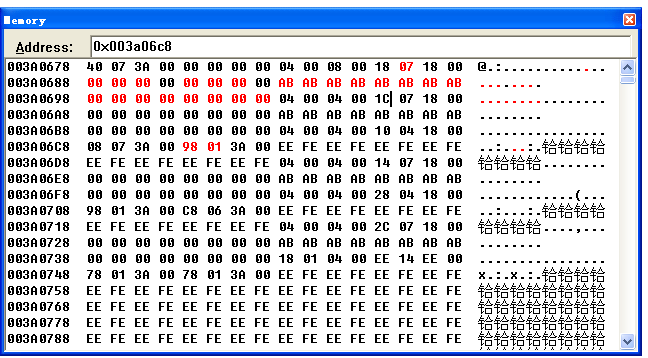


（h3的块首地址是003a06c0）

同样可以看见块首同上面的变化；之前的块身中变成了98 01 3a 00 88 06 3a 00，即在此时，这个块身里保存的信息为：Flink是00 3a 01 98（为freelist[2]的地址），Blink是00 3a 06 88，也就是指向了之前h1的块身；从这里就可以看出来，此时h1和h3就在freelist[2]连接上了；

之后的HeapFree(hp,0,h5)也是同理。

当再次使用HeapAlloc为h1申请堆块时，可以看见此时h1依旧是之前的状态，从freelist[2]上摘了下来，Flink和Blink被释放，都变成了0的待存储状态



**心得体会：**

对堆块的结构、分配、回收等等更加的熟悉了，同时也明白了一些dword shoot的机理；如果在执行HeapAlloc(hp,HEAP\_ZERO\_MEMORY,8)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。但是在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，对于用户来说会越来越安全。

我在实验过程中产生了一些问题：  
 1. 堆块后面的EE FE EE FE 是怎么进行工作的，如果堆块被修改了，是按照什么样的规律去改变内部的值的？

1. 如何具体地完成h1的Flink和Blink的更改？
2. 堆块的块首存储着什么样的信息？

这些都有待我进一步进行探索。