《漏洞利用及渗透测试基础》实验报告

姓名：赵悦蛟 学号：2313650 班级： 1071

**实验名称：**

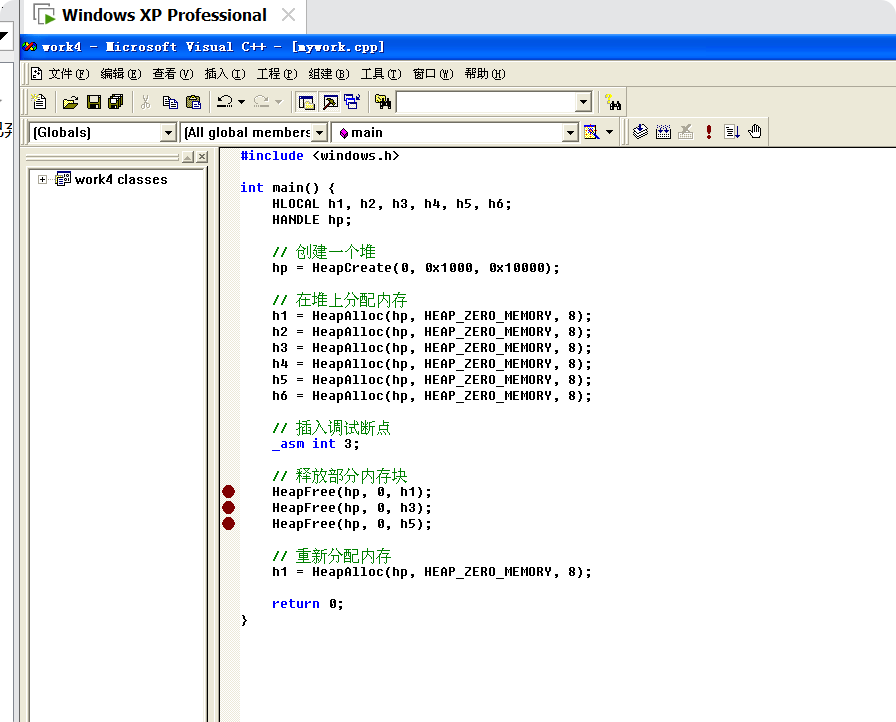
堆溢出

**实验要求：**

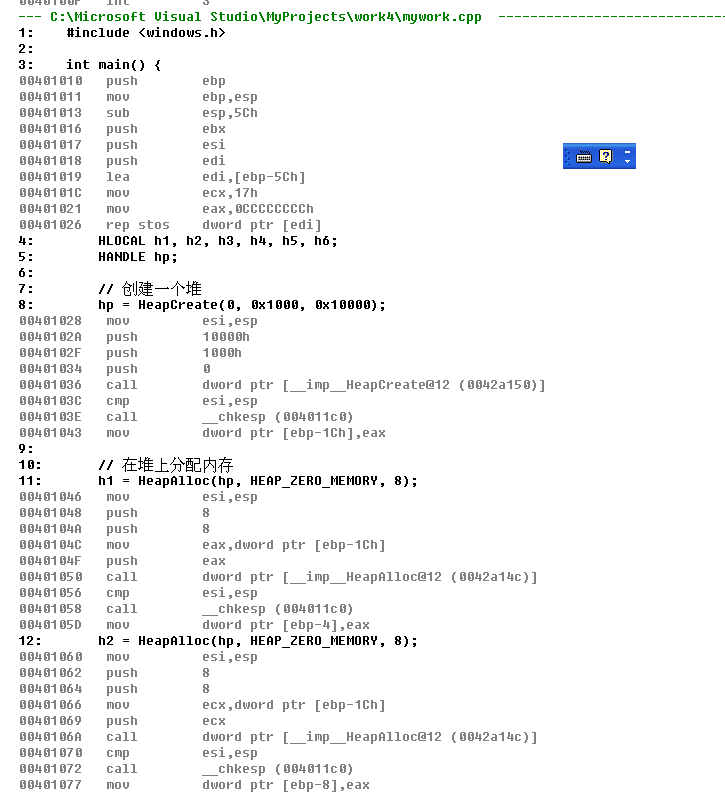
以第四章示例4-4代码为准，在VC IDE中进行调试，观察堆管理结构，记录Unlink节点时的双向空闲链表的状态变化，了解堆溢出漏洞下的Dword Shoot攻击。

**实验过程：**

# 1.按照例4-4在xp虚拟机里用vc6编写好代码

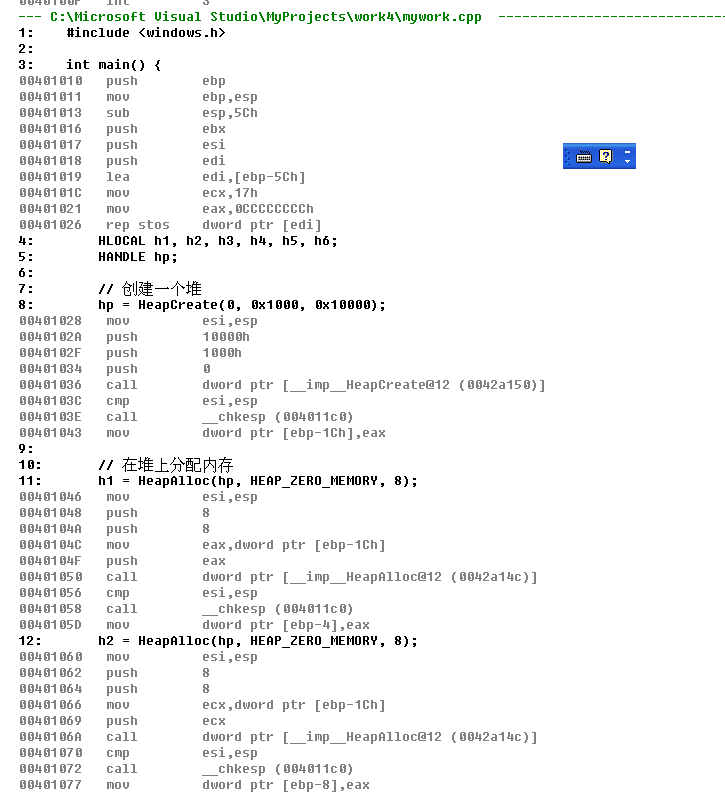


# 2.在 HeapFree(hp, 0, h1);HeapFree(hp, 0, h3);HeapFree(hp, 0, h5);三条语句处设置断点，进入debug模式观察程序运行时的汇编指令和地址变化。

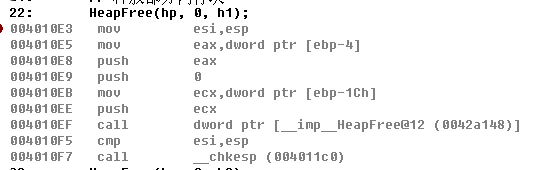
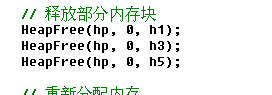


# 3.程序流程解析

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。

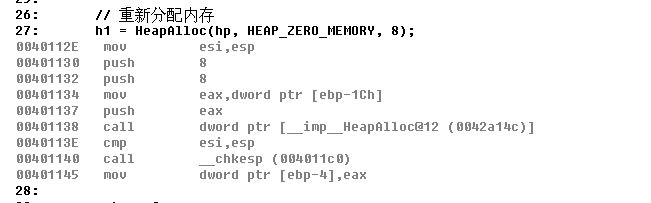


1. 用Heapfree释放奇数次申请的堆块，防止堆块合并的发生，释放h1时的汇编语句如图所示



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

1. 调用HeapAlloc函数重新分配内存



第个 push 8 将分配大小（8字节）压入栈中，作为 HeapAlloc 的第三个参数。第二个 push 8 将选项标志 HEAP\_ZERO\_MEMORY 的值压入栈中，作为 HeapAlloc 的第二个参数。call dword ptr [\_\_imp\_\_HeapAlloc@12 (8042a14c)]调用HeapAlloc函数重新分配内存。

# 通过调试程序观察堆内存变化

使用vc6.0自身的调试器具体了解堆管理过程中的内存变化，使用F10快捷键进行单步调试。

（1）执行HeapFree(hp,0,h1）语句时。

hp为0x003a0000,h1为0x003a0688，根据堆块结构知道h1堆0x003a0688，块首起始位置为0x003a0680。观察该语句执行后，对应所示。

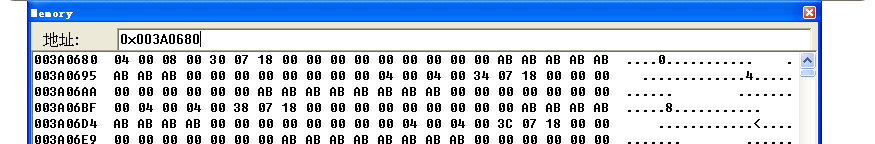


图 1h1堆块首起始位置

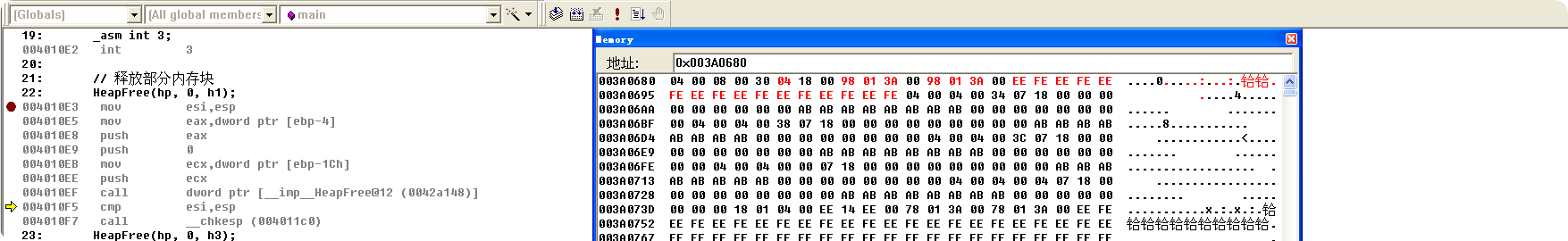


图 2调用HeapFree(hp, 0, h1)函数时的内存变化

可见，除了块首状态变化外，0x003a0688开始的块身位置的前8字节（flink和blink)发生了变化，由0x000000变为具体的有效地址。注意到，这是第一个16字节的堆块释放，将被链入freelist[2］空表中，而此时flink和blink 的值都是0x003a0198，也是freelist[2]的地址。

转到0x003a0198处：

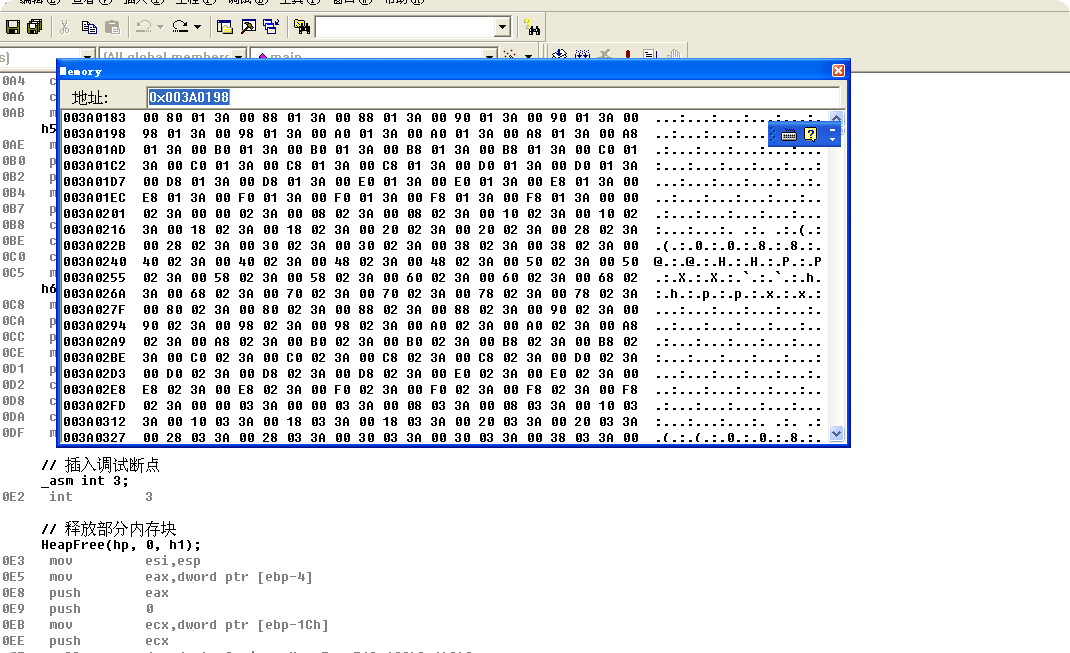


图 3 调用函数前0x003a0198处内存

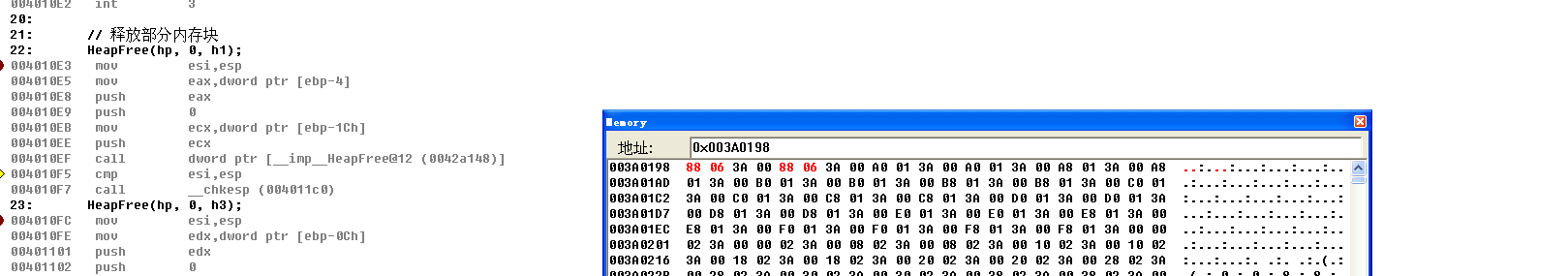


图 4 执行HeapFree(hp,0,h1）时

0x003a0198处内存

可见，freelist[2］的 flink 和 blink 都是 0x003a0688。这意味着，当前freelist[2］唯一后继结点就是刚刚空闲的h1块（地址为0x003a0688),h1块的唯一前继结点是 freelist[2]。其他地址（freelist[3]、freelist[4]、freelist[5]）的 flink 和 blink 均指向自身，说明是空表。

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

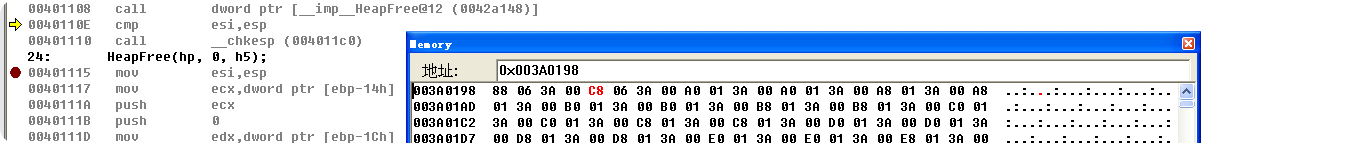


图 5执行HeapFree(hp,0,h3）时0x003a0198处内存

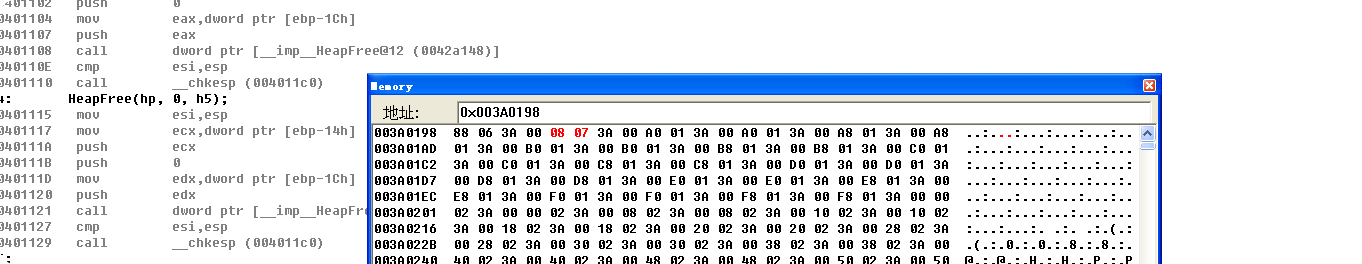


图 6执行HeapFree(hp,0,h5）时0x003a0198处内存

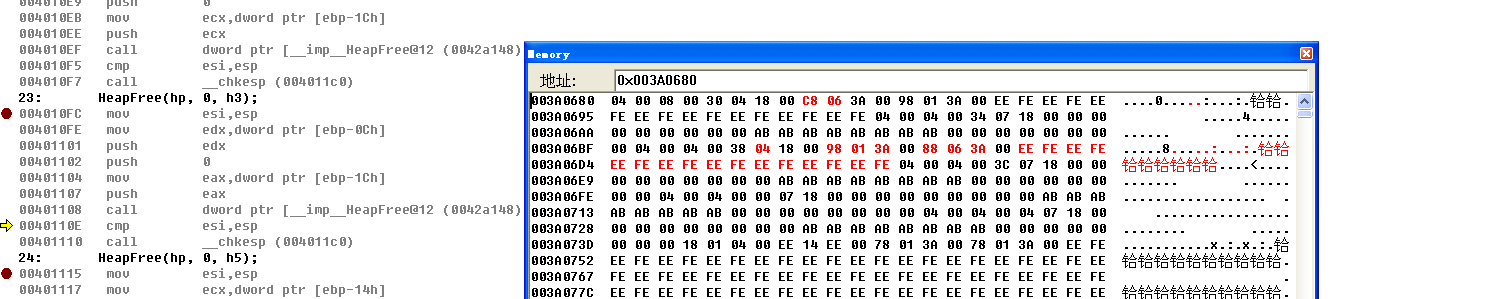


图 7执行HeapFree(hp,0,h3）时0x003a0680处内存

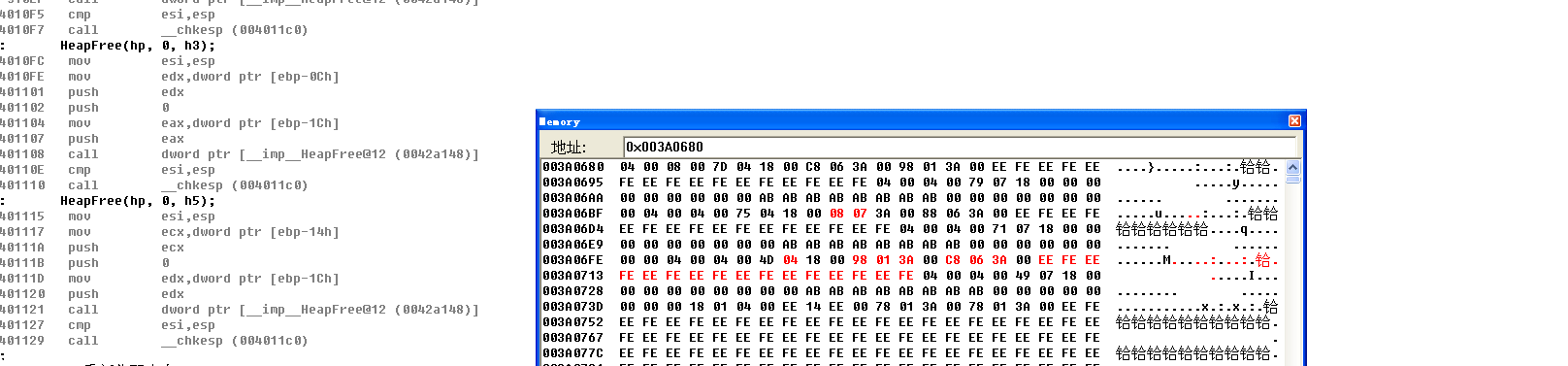


图 8执行HeapFree(hp,0,h5）时0x003a0680处内存

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

（3）执行h1=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169的堆块，首先摘得h1（地址为0x003a0688)。

观察此时的内存。

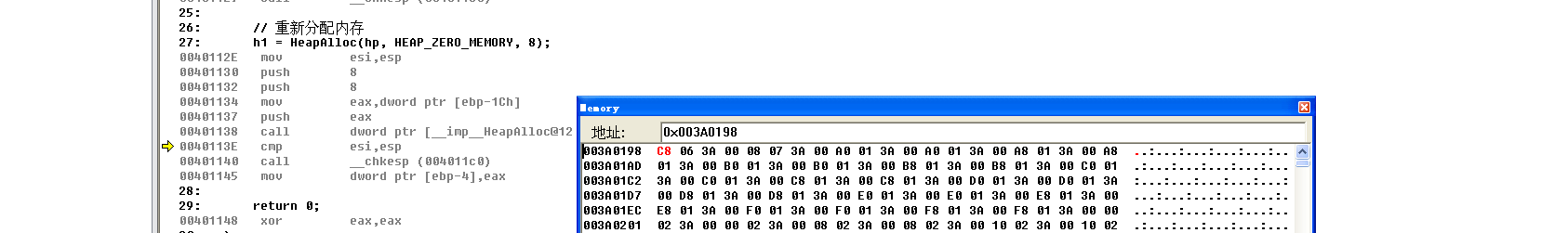


图 9执行HeapAlloc时0x003a0198处内存

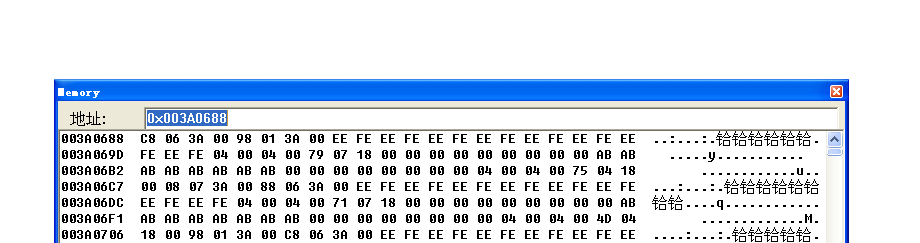


图 10执行HeapAlloc前0x003a0688处内存

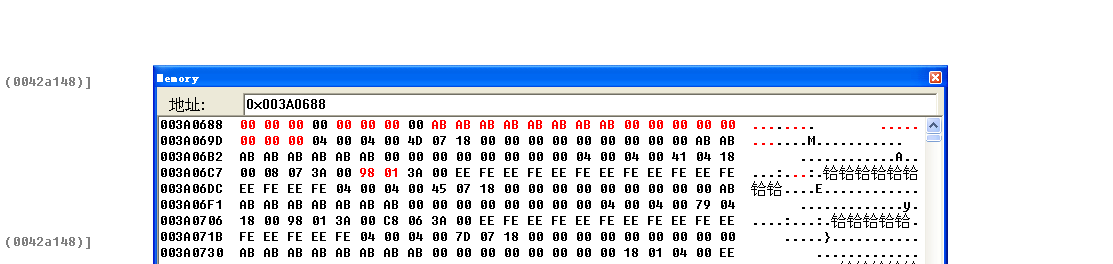


图 11执行HeapAlloc时0x003a0688处内存变化

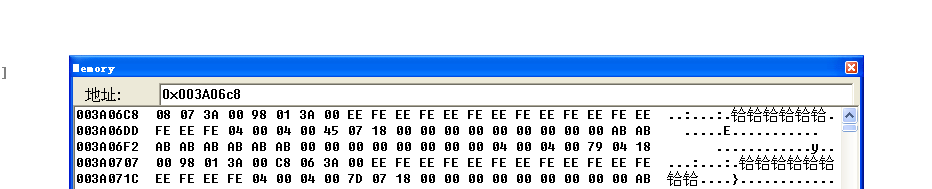


图 12执行HeapAlloc后0x003a0688处内存

①freelist[2]（地址为0x003a0198）所存储的信息：flink（前4字节）为0x003al blink（后4字节）为0x003a0708。

② h1（地址为0x003a0688）所存储的信息：flink为0x003a06c8,blink为0x003a019

③h3（地址为0x003a06c8）所存储的信息：flink 为 0x003a0708,blink为0x003a068

摘走h1之后，内存的变化：

①freelist[2]（地址为0x003a0198）的前4字节变为0x003a06c8，实际发生了将向指针（值为0x003a0198）地址处的值写为h1前向指针的值。

②h3（地址为0x003a06c8）的blink变为h1→blink，即0x003a0198，实际发生了前向指针（值为0x003a06c8）地址处的值写为h1后向指针的值。

# 手动修改h1块首中的指针，观察Dword Shoot攻击的发生

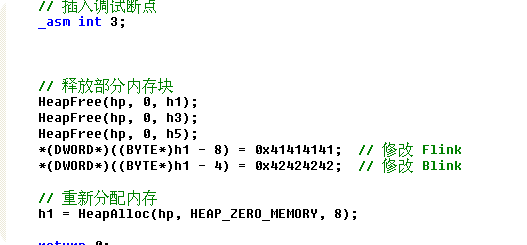


图 13修改h1指针

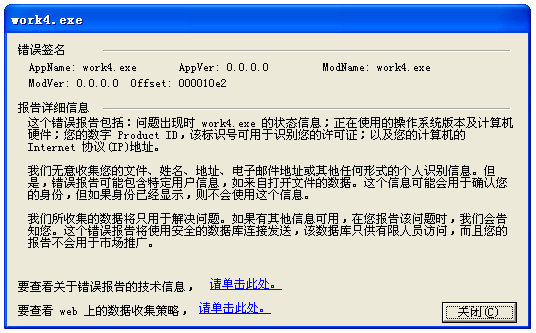


图 14程序运行时提示异常

假设在执行 h1 = 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

在程序中增加\*(DWORD\*)((BYTE\*)h1 - 8) = 0x41414141; 和\*(DWORD\*)((BYTE\*)h1 - 4) = 0x42424242; 两条语句，手动修改h1的flink和blink指向的地址，当 h1 被释放时，堆管理器会尝试将 h1 块插入到空闲链表中。由于 h1 块首中的 Flink 和 Blink 指针被修改为 0x41414141 和 0x42424242，堆管理器会尝试将 h1 块插入到链表中，从而导致内存地址 0x41414141 和 0x42424242 被覆盖，完成了一次Dword Shoot攻击。

**心得体会：**

实验中通过单步调试和内存观察，我能够清晰地看到堆块的状态变化以及指针修改后的内存覆盖效果。我也对调试程序掌握的更加熟练。

通过本次实验，我对堆溢出漏洞及其利用方式有了更深入的理解，在实验中，我通过手动修改 h1 块首中的指针，成功触发了 Dword Shoot 攻击，导致程序崩溃。这使我对 Dword Shoot 攻击的原理和实现过程有了直观的认识。在实验过程中，我通过调试器观察了堆块的结构以及堆管理器的行为，更好地理解了 Windows 堆管理机制中空闲链表的工作原理。