《软件安全》实验报告

姓名： 冯思程 学号：2112213 班级：1120（张健）

**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. 首先，先在xp系统中打开VC然后创建好文件，将源码输入，源码展示如下：

#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);

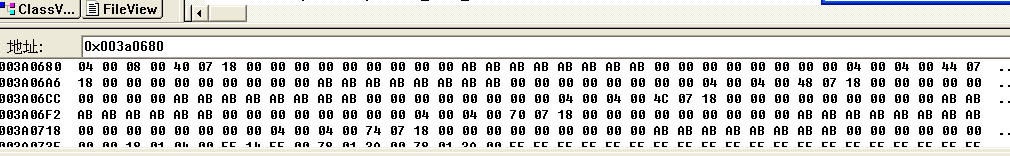
    HeapFree(hp,0,h5); //现在freelist[2]有3个元素

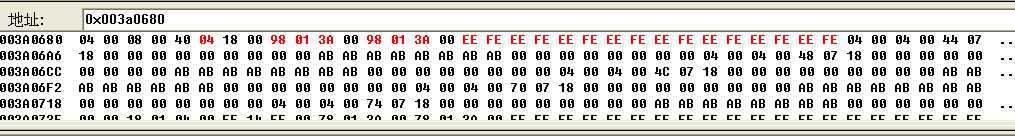
    h1 = HeapAlloc(hp,HEAP\_ZERO\_MEMORY,8);

    return 0;

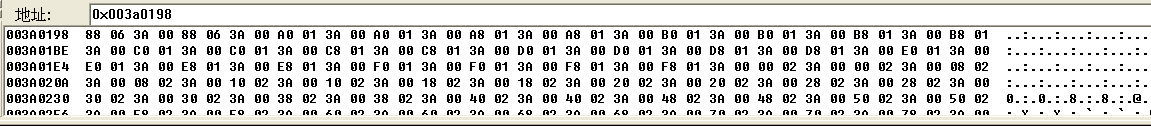
}

1. 按F5进入debug模式，首先将鼠标箭头放到hp上可以发现hp的0x003a0000，h1是0x003a0688，因为堆块分为块首和块身，块首大小为8字节，而且块首在块身的前面。所以我们可以知道h1堆块的块首起始位置是0x003a0680，块身的起始位置是0x003a0688，我们将0x003a0680地址输入，然后点击F10执行将h1释放的语句，同时观察堆块发生的变化，下面中的上图是没执行该语句之前，下图是执行该语句之后：





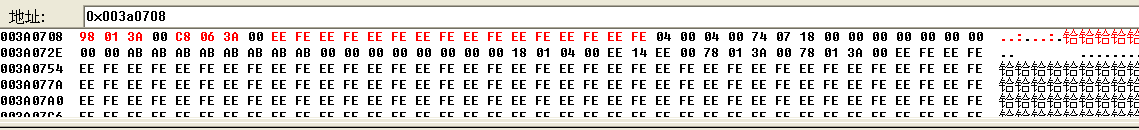
首先，发现块首状态发生了变化，这是由于块的状态发生了改变。然后我发现块身的起始的flink和blink也发生了变化（flink在前，blink在后，各占4字节），两者都是由0x00000000变成了0x003a0198。这是第一个块被释放然后链入到freelist[2]空链表中，而flink和blink的共同地址0x003a0198也是freelist[2]的地址，将该地址输入后观察，如下图：



发现组成前8字节的flink和blink都是0x003a0688，说明freelist[2]链接的节点只有h1一个块，同时发现其他地址的flink和blink都指向自身，说明都是空表。

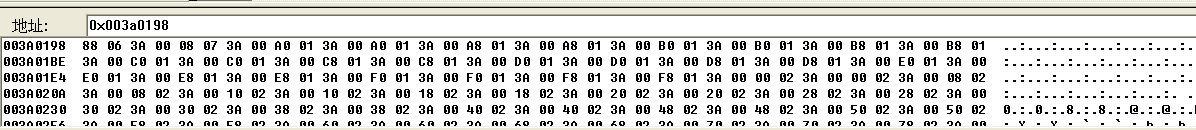
3然后按F10分别执行h3和h5的指令，我知道h3的块身起始地址是0x003a06c8，h5块身的起始地址是0x003a0708，在执行完释放h3和h5语句后去找到对应的地址进行观察，如下图：

执行完h3和h5的释放：

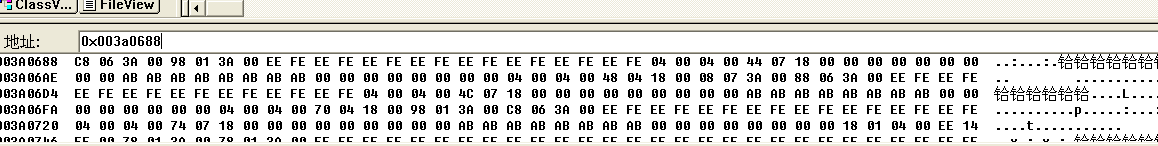


这时候，h5块的flink是0x003a0198，blink是0x003a06c8

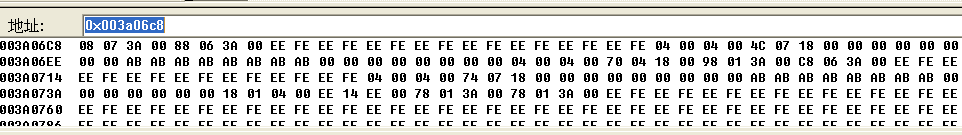
然后，我去观察0x003a0688和0x003a0198和0x003a06c8地址的flink和 blink，如下图：



该地址的flink是0x003a0688，blink是0x003a0708



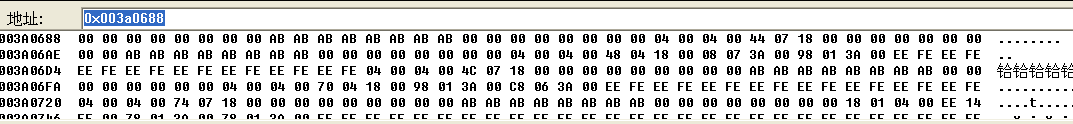
h1块的flink是0x003a06c8，blink是0x003a0198



h3块的flink是0x003a708，blink是0x003a0688

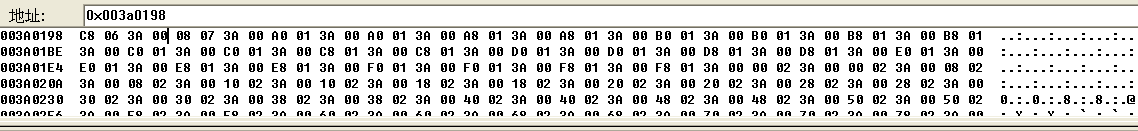
综上，可以发现freelist[2]先链接h1，然后链接h3，然后链接h5，这是一共双向循环链表，可以发现该freelist[2]链表的状态是freelist[2]<=>h1<=>h3<=>h5，且freelist[2]和h5也是链接上的。

4.然后执行h1 = HeapAlloc(hp,HEAP\_ZERO\_MEMORY,8);语句，重新分配空间，查看0x003a0688地址，发现h1块的flink和blink都变成了0x00000000，如下图：



说明h1块被重新分配，不在freelist[2]空闲链表中了。

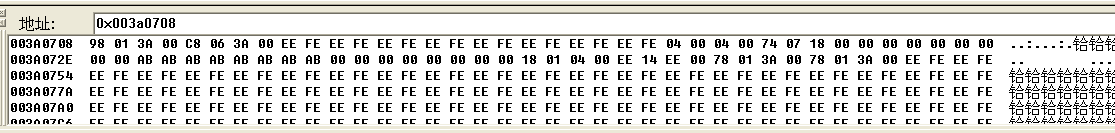
然后去观察地址0x003a0198地址，如下图：



发现变化是flink变化成了0x003a06c8，说明freelist的后继节点从h1块变成了h3块

然后观察地址0x003a06c8，发现blink变成了0x003a0198，说明h3块的前继节点由h1块变成了freelist[2].

最后在观察地址0x003a0708地址，如下图:



发现flink和blink都没有变化。此时我们可以看出freelist[2]的状态变成了freelist[2]<=>h3<=>h5。

5.Dword shoot攻击，我知道了堆溢出的情况如上述所示，那么如果我可以特定h1块的flink和blink就可以实现对内存的任意地址进行操作，从而完成一次Dword Shoot攻击

**心得体会：**

1. 对堆空间分配释放的过程有了更清晰的认知，同时通过自己亲身用VC进行调试分析，使我分析代码的能力更加熟练。
2. 同时对堆的声明，分配，释放，重新分配的原理有了更加深刻的理解。
3. 最后我思考了如何进行一次dword shoot攻击，我简单阐述一下原理，我知道ecx存flink，edi存blink，通过堆溢出将ecx改为我们所要修改的数据，把edi改为我们希望改动的地址，此时我们可以实现把edi所对应的内存值改为ecx。这就是DWORD shoot攻击。之所以要这么做，在从空闲链表卸下一个空闲堆块时候，会执行该语句：mov dword ptr [edi] ecx，该语句可以解释成空闲堆块的前向指针（flink）中的数据写入后向指针（blink）指向的地址，因此前面的操作让我在系统卸下该空闲堆块的时候，我可以在目标地址写入目标数据，实现一次dword shoot。