《软件安全》实验报告

姓名： 王众 学号：2313211

实验名称：

堆溢出Dword Shoot攻击实验

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.进入VC反汇编程序

实验所用的源代码如下：

```
#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;
}
```

我们先观察程序从而对其有一个基本的了解。首先，程序创建了一个大小为`0x1000`的堆区。并向其中连续申请了6个块身大小为8字节的堆块，加上块首实际上就是6个大小为16字节的堆块。

然后我们释放奇数次申请的堆块，从而防止堆块合并。

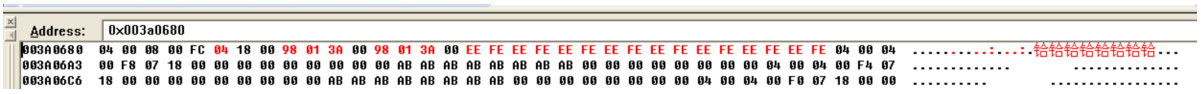
当三次释放结束之后，会形成三个16字节的空闲堆块，并将其放入列表。因为大小为16字节，所以会依次放入`freeliest[2]`这个空闲列表中，他们依次是`h1`、`h3`、`h5`。

我们再次申请8字节的堆区内存，加上块首是16字节，因此会从`freelist[2]`所表示的空表中摘取第一个空闲堆块，即`h1`。之后我们对`h1`的前后指针进行修改，便可观察到`Dword shoot`攻击。下面，我们将设置断点，贯彻整个攻击的两个指针的变化流程。

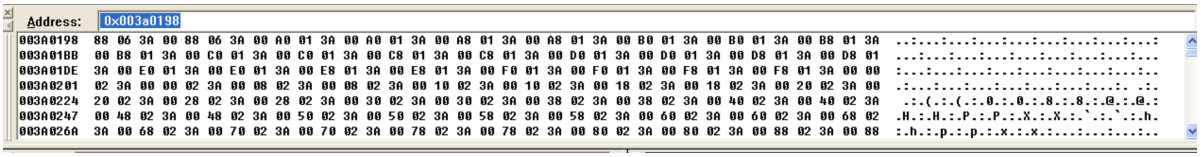
2.堆管理过程中的内存具体变化

在实验过程中，我们主要通过打断点来实现代码过程的观察。

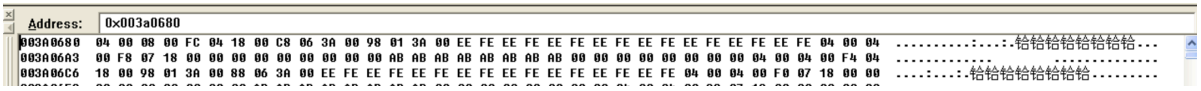
将鼠标移到第一个块身`h1`处，观察到其地址为`0x003a0688`，因为这是块身的起始地址，再减去8 就是块首的地址`0x003a0680`。当执行完`h1`堆块的释放后，我们跳转到这个地址观察。



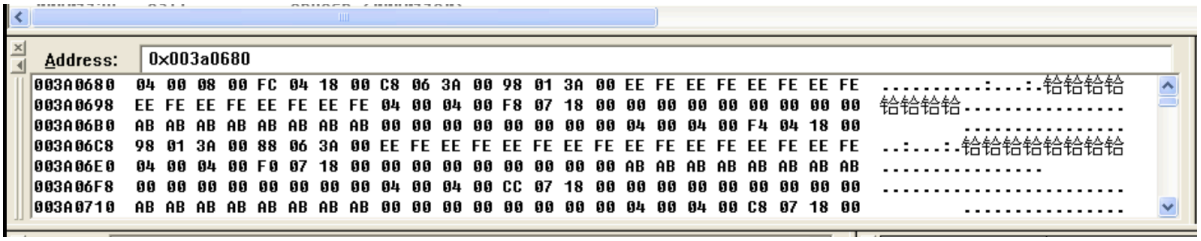
观察`0x003a0680`开始的内存，前八个字节是块首的一些信息。后八个字节分别是`flink`和`blink`对应的内容，可以看到他们都指向了`0x003a0198`。根据堆块空表的管理方式，且h2 没有东西放进来，我们可以推测出这个地址 实际上就是`freelist[2]`的地址。我们可以跳转到这个地址观察即可。



我们发现，`freelist[2]`的`flink`和`blink`均指向了`0x003a0688`，这就是刚才释放的`h1`的地 址，同时也说明此时这条链上只链入了`h1`一个空闲堆块，前向指针和后向指针都指向`h1`。 接着，我们来完成释放`h3`的操作，释放后，继续跳转到`h1`的内存处查看。



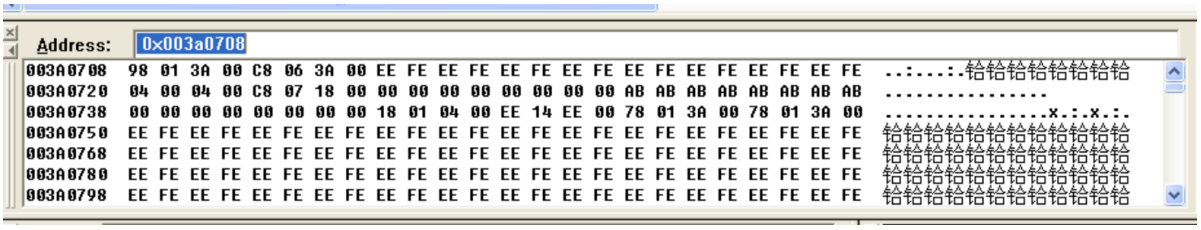
我们发现，与上一步相比，`h1`的`flink`指针位置发生了改变，说明在空表上`freelist[2]`中又链 入了一个空闲堆块，即`h3`。 `h1`的前向指针此时指向的`0x003a06c8`就是`h3`堆块的块身地址。我们跳转 到 h3 所在的地址`0x003a06c8`查看即可。



这里，`0 03A06C8`前四个字节是`flink`的值，指向的就是`freelist[2]`，后四个字节是`blink`的值，指向的是`h1`堆块的块身。这说明此时空表结构为：`freelist[2]`链接`h1`链接`h3`，`h3`又和`freelist[2]`双向 链接。

我们继续进行`h5`的释放，释放完之后，跳转到`0x003a06c8`，即 h3 的块身地址查看

发现前指针发生了变化。指向了`0708`，所以我们进行进一步的跳转。

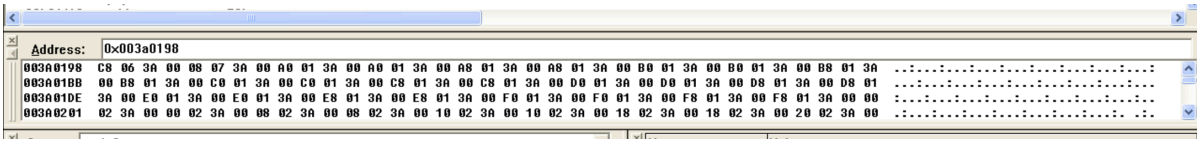


可以发现，`flink`指针所指向的地址依然是`freelist[2]`，后面的`blink`指向的是`h3`的地址，此时，我们已经完成了三个堆块的释放，我们分析出此时块表内存的存储结构应该是：`freelist[2]`链接`h1`链接`h3`链接`h5`，`h5`又和`freelist[2]`双向链接。

综上所述，此时各个堆块的`flink`和`blink`：

	flink的地址	**blink的地址**
Freelist[2]	**0x003a0688(h1)**	**0x003a06708(h5)**
h1	**0x003a06c8(h3)**	**0x003a0198(freelist[2])**
h3	**0x003a0708(h5)**	**0x003a0688(h1)**
h5	**0x003a0198(freelist[2])**	**0x003a06c8(h3)**

最后我们重新分配一个块身为8字节的堆。取下第一个空闲堆块`h1`.这时我们先跳转到`0x003a0198`，即`freelist[2]`所在的位置。



跳转到`0x003a06c8`，即`h3`处，我们发现，我们发现`h3`堆块的`blink`变为了`0x003a0198`（`freelist[2]`），即发生了**将`h1`后向指针的值写入到`h1`前向指针所指的地址内存里。**

以上就是完成了`Dword Shoot`攻击原理的展示。卸下一个堆块的时候，会将其前向指针和后向指针的值写入到其指向的内存当中，因此我们可以利用这个来实现一次`Dword Shoot`攻击。

心得体会：

1.学会了如何在VC6中进行建立堆，释放堆的操作。

2.通过跟踪堆块内存位置，深入地理解了堆表的内存管理形式，理解了如何进行堆表的合并和空表的链接等知识点，明白了头尾指针（`flink`和`blink`）的变化形式。

3.理解`Dword shoot`的攻击原理，即**精心构造一个地址和一个数据，当这个空闲堆块从链表里卸下的时候，就获得一次向内存构造的任意地址写入一个任意数据的机会。**

4.通过学习了解到了栈溢出、堆溢出的危害，包括之前自己设计代码的缺陷性。