《软件安全》实验报告

姓名：蒋枘言　学号：2313546　班级：信息安全班

一、实验名称

堆溢出 Dword Shoot 模拟实验

二、实验要求

以第四章示例 4-4 代码为准，在 Microsoft Visual C++ 6.0 中进行调试，观察堆管理结构，记录卸载（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。

三、实验过程

首先打开 VC6，编写以下程序，按 F5 开始调试。

mainpp.cpp

```
#include <windows.h>
int main() {
    HLOCAL h1, h2, h3, h4, h5, h6;
    HANDLE hp;
    hp = HeapCreate(0, 0x1000, 0x1000); // 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int 3中断指令，让调试器在此处中断

    // 依次释放奇数次申请的堆块，避免堆块合并
    HeapFree(hp, 0, h1);
    HeapFree(hp, 0, h3);
    HeapFree(hp, 0, h5);

    h1 = HeapAlloc(hp, HEAP_ZERO_MEMORY, 8);
    return 0;
}
```

（1）堆初始化与内存布局（断点之前）

```
HANDLE hp = HeapCreate(0, 0x1000, 0x1000);                  // 创建堆
HLOCAL h1~h6 = HeapAlloc(hp, HEAP_ZERO_MEMORY, 8);          // 分配堆块
```

HANDLE 是 Windows 中的通用句柄类型，用于标识系统资源（如文件、进程、堆等），本质是一个指针大小的整数（32 位系统为 4 字节，64 位为 8 字节），表示对内核对象的引用。HLOCAL 特定于本地堆（Local Heap）的内存块句柄。

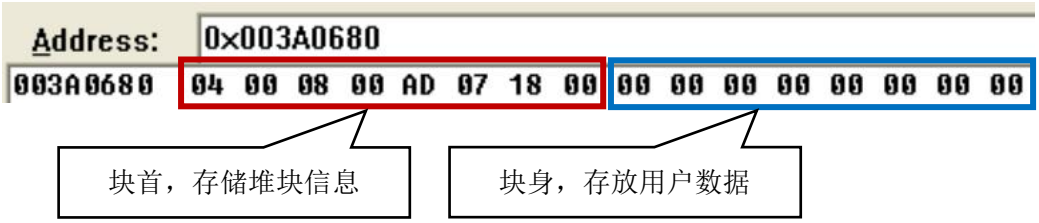
HeapCreate 函数的作用是创建一个新的私有堆，原型为：

```
HANDLE HeapCreate(
    DWORD flOptions,          // 堆创建选项
    SIZE_T dwInitialSize,     // 初始堆大小（字节）
    SIZE_T dwMaximumSize      // 最大堆大小（字节）
);
```

HeapAlloc 函数的作用是从指定堆中分配一块内存，原型为：

```
LPVOID HeapAlloc(
    HANDLE hHeap,  // 堆句柄（由 HeapCreate 返回）
    DWORD dwFlags, // 分配选项（HEAP_ZERO_MEMORY 表示分配的内存初始化为 0）
    SIZE_T dwBytes   // 请求分配的大小（字节）
);
```

这几行程序创建了一个大小为 0x1000 字节的堆区，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）。此时 hp 为 003A0000，h1 为 003A0688（h1 块身的起始位置），故 h1 块首的起始位置为 003A0680。



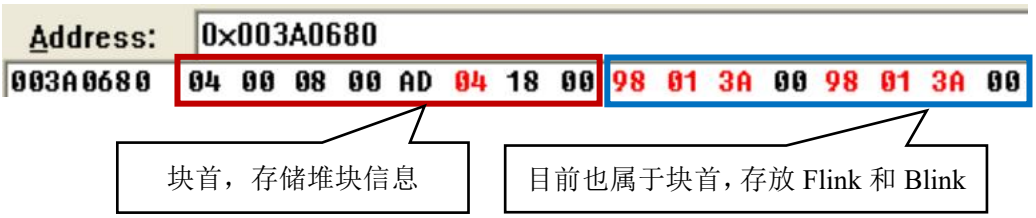
（2）释放奇数次堆块（断点之后）

```
HeapFree(hp, 0, h1);
```

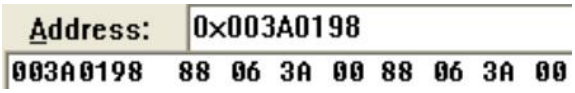
HeapFree 函数的作用是释放通过 HeapAlloc 分配的内存块，原型为：

```
BOOL HeapFree(
    HANDLE hHeap,      // 堆句柄
    DWORD dwFlags,     // 释放选项（通常为 0）
    LPVOID lpMem       // 要释放的内存指针
);
```

执行这一行语句时，可以看到从 003A0688 位置开始的块身被征用（仅征用 8 字节），用于存放 Flink 和 Blink，此时 Flink 和 Blink 都为 003A0198。由于块身的大小为 16 字节，该块身将被链入 freelist[2]空表中，故 freelist[2]空表的地址也为 003A0198。

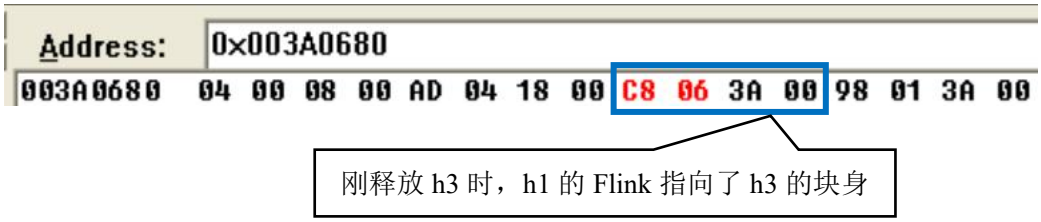


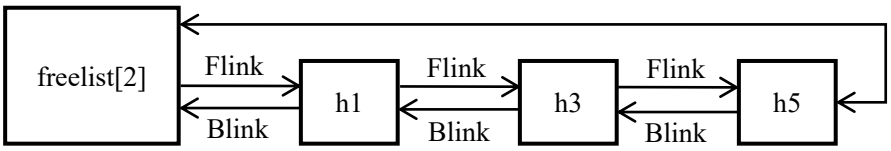
转到 003A0198 处，发现 freelist[2]的 Flink 和 Blink 都为 003A0688，都指向 h1 的块身。此时 h1 是唯一链入 freelist[2]的堆块。



```
HeapFree(hp, 0, h3);
HeapFree(hp, 0, h5);
```

接着依次释放 h3 和 h5，此时它们均链入了 freelist[2]中。



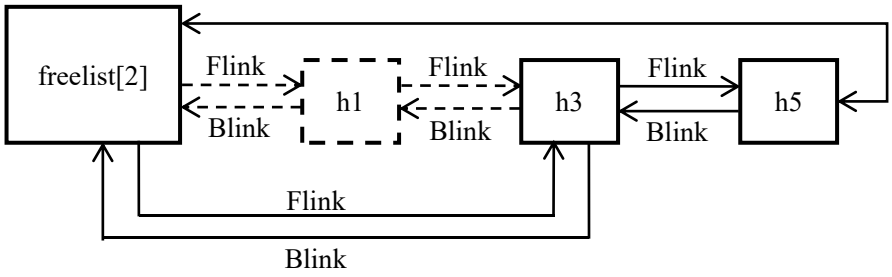


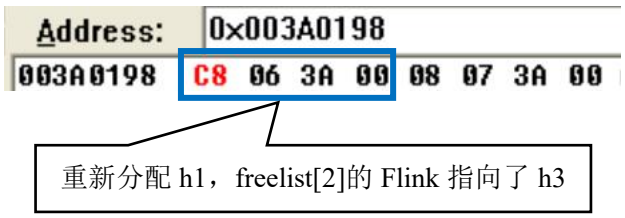
（3）重新分配 h1 触发 Dword Shoot

```
h1 = HeapAlloc(hp, HEAP_ZERO_MEMORY, 8);
```

当从空闲链表中分配 h1 时，系统需将其从 freelist[2]中摘除，执行以下操作：

```
h1->blink->flink = h1->flink;      // 将 h1 的前驱节点的 Flink 指向 h1 的后继
h1->flink->blink = h1->blink;      // 将 h1 的后继节点的 Blink 指向 h1 的前驱
```





若攻击者通过溢出或其他方式篡改 h1 的 Flink 和 Blink，上述操作可向任意地址写入恶意数据，进行 Dword Shoot 攻击。

四、心得体会

通过本次堆溢出 Dword Shoot 模拟实验，我对软件安全中的堆溢出漏洞有了更深入的理解。在实验过程中，我仔细观察了堆管理结构的变化，特别是卸载（Unlink）节点时双向空

闲链表的状态变化。这让我清晰地看到了堆溢出漏洞是如何被利用来实施 Dword Shoot 攻击的。

这次实验让我深刻体会到了软件安全的重要性。堆溢出漏洞作为一种常见的安全威胁，如果被恶意利用，可能会导致系统被攻击、数据泄露等严重后果。因此，在软件开发过程中，必须严格遵循安全编码规范，仔细检查和验证输入数据，避免缓冲区溢出等问题的发生。