《软件安全》实验报告

姓名：王昱　　学号：2212046　班级：信息安全班

实验名称：

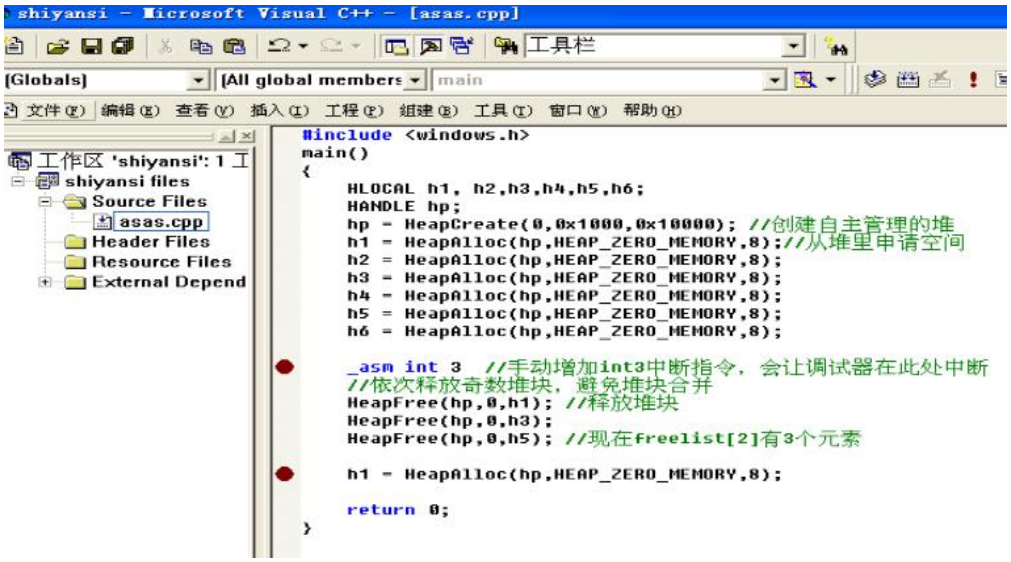
堆溢出 Dword Shoot 模拟实验

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. 进入 VC++



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。

（2）释放奇数次申请的堆块是为了防止堆块合并的发生。

（3）三次释放结束后，会形成三个 16 个字节的空闲堆块放入空表。因为是 16 个字节，所以会被依次放入 freelist[2]所标识的空表，它们依次是 h1、h3、h5。

（4）再次申请 8 字节的堆区内存，加上块首是 16 个字节，因此会从 freelist[2]所标识的空表中摘取第一个空闲堆块出来，即 h1。

此时，在取 h1 的时候，如果我们通过堆溢出，手动修改 h1 块首中的前后向指针，就能够观察到 DWORD SHOOT 的发生。

2.实验内容

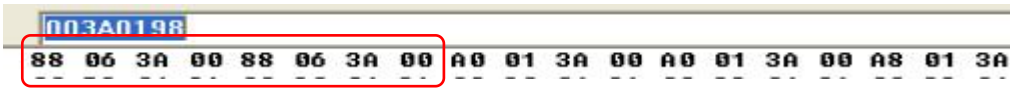
（1）进入 DeBug 模式，执行到刚申请完地址，还没有释放 h1 的位置，即第一个断点的前一句。

此时查看 h1,h2,h3,h4,h5,h6 的地址为：

h1:0x003a0688　　h2:0x003a06a8　　h3:0x003a06c8
h4:0x003a06e8　　h5:0x003a0708　　h6:0x003a0728
此时 h1 在内存中如下（其他堆也类似）：

h1 的前八个字节(0x003A0680-0x003A0688)是块首，负责记录块的信息。
后八个字节是块身，负责记录块的内容，此时都为 0。
(2)释放 h1

h1 的块首：块的相关状态信息改变
h1 的块身：前向指针 flink（0x003a0688-0x003a068C）：003A0198
后向指针 blink（0x003a068D-0x003a068F）：003A0198
由于此时 freelist[2]只有一个元素，所以前向指针，后向指针均为 h1 的地址，h1 也指向 freelist，所以 003A0198 即为 freelist[2]的地址。

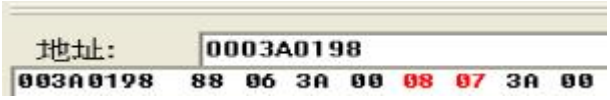


此时查看 freelist[2]的地址 0x003A0198，可以看到储存了 h1 的地址（003A0688），此时前后指针均指向 h1，说明空表中只有一个元素，freelist[2]和相互指向对方。
（2）释放 h3
执行完释放 h3 的代码后，首先查看 freelist[2]的变化

Flink 指向(0x003A0688)h1，Blink 指向（003A06C8）h3,说明空表中有两个元素分别为 h1 和 h3。
接着查看 h3:

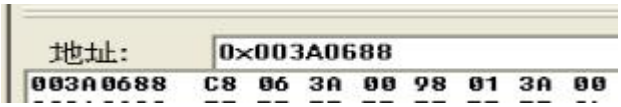
h3 的 Flink 则指向了 h1 块身的首地址（0x003A0688），Blink 指向了 freelist[2]的地址（0x003A0198），这说明，h3 已经串入了 freelist[2]中，前面连着 h1,后面连着 freelist[2]。
最后查看 h1：

h1 的 Flink 指向了 freelist[2]的地址（0x003A0198），Blink 指向了 h3(0x003A06C8).
（3）释放 h5
执行完释放 h5 的代码后，查看 freelist[2]的变化



Flink 不变，仍然指向 h1(0x003A0688)，Blink 指向 h5(0x003A0708)。

查看 h1 的变化

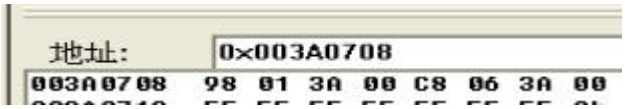


h1 不发生改变。

查看 h3 的变化

Flink 指向 h1 不变，Blink 指向 h5（0x003A0708）。

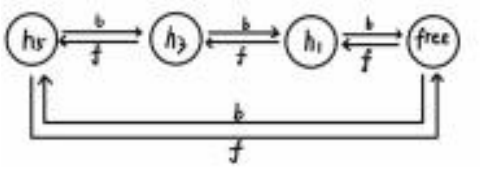
查看 h5 的变化



Flink 指向 h3（0x003A06C8），Blink 指向 freelist[2]。

至此，奇数堆的释放完成。

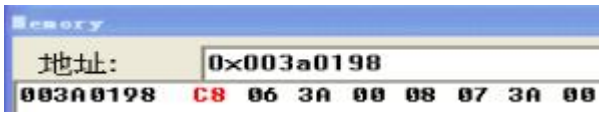
此时空表的逻辑图如下：



(4)创建恢复 h1

当再次申请大小为 8 个字节的堆块时，将从 freelist[2]中摘下第一个堆块返回给程序，也就是摘下了 h1。

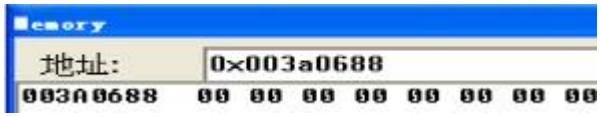
此时 freelist 的变化：



可以看到 Flink 从 h1 变成了 h3，Blink 不变，还是 h5。

相当于执行 node—>blink—>flink = node—>flink。

查看 h1：



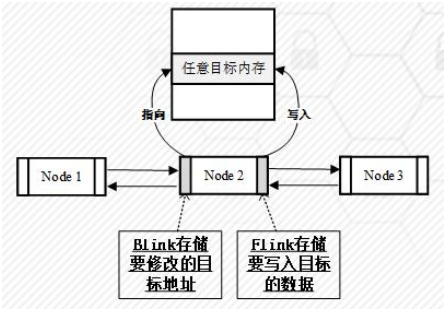
此时 h1 的块身全为 0。

查看 h3：

h3 的 blink 变成了 freelist[2]（003A0198）,flink 变成了 h5(003A0708)。

相当于执行 node—>flink—>blink = node—>blink。

我们可以注意到，在这个过程中其实是发生了



node1->Blink = node2->Blink

node3->Flink = node1->Flink

（5）堆溢出漏洞的 Dword Shoot 攻击

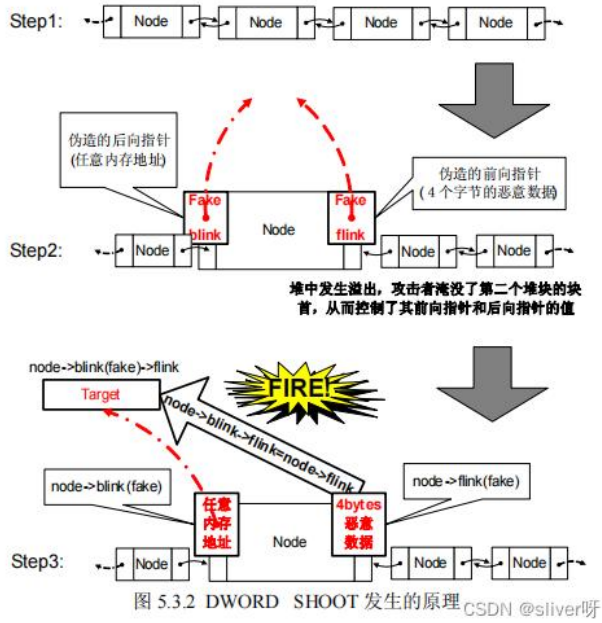


图 5.3.2 DWORD SHOOT 发生的原理 CSDN @sliver呀

如图所示，当堆溢出发生时，非法数据可以淹没下一个堆块的块首。这时，块首是可以被攻击者控制的，即块首中存放的前向指针（ flink ）和后向指针（ blink ）是可以被攻击者伪造的。当这个堆 块被从双向链表中“卸下”时， node -> blink -> flink = node -> flink 将把伪造的 flink 指针值写入伪造的 blink 所指的地址中去，从而发生 DWORD SHOOT。

基于 Dword Shoot 攻击，攻击者甚至可以劫持进程，运行植入的恶意代码。

心得体会：

1. 了解了堆的内存结构，以及申请与释放堆空间的操作。以及这些操作时内存空间发生的变化，也掌握了双向空闲链表插入新节点的过程。

2. 对汇编语言的语法更加熟悉。

3. 通过实验了解了堆溢出漏洞下的 Dword Shoot 攻击；