

# 《漏洞利用及渗透测试基础》实验报告

姓名：申宗尚 学号：2213924 班级：信息安全

## 实验名称：

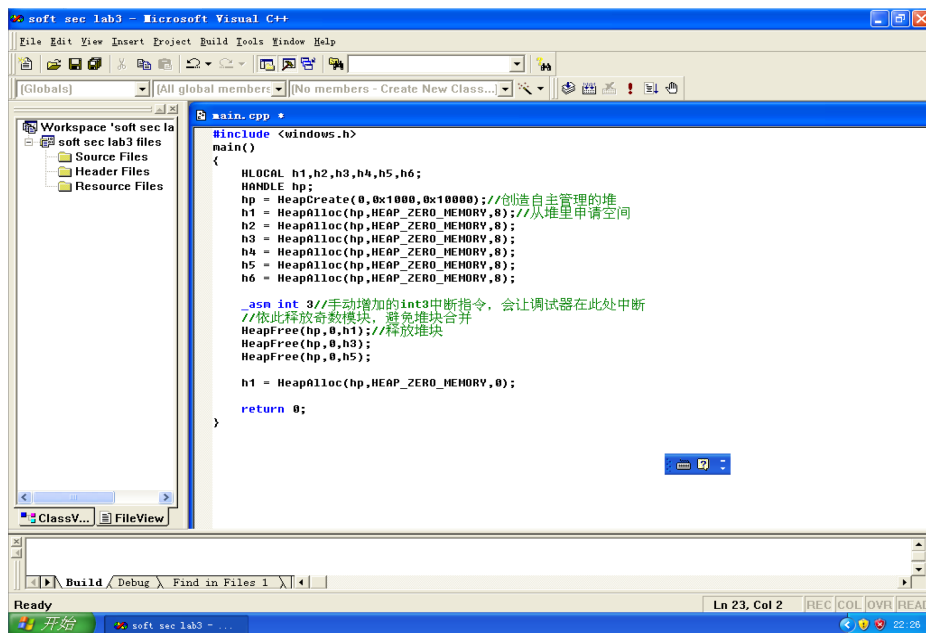
堆溢出 Dword Shoot 模拟实验

## 实验要求：

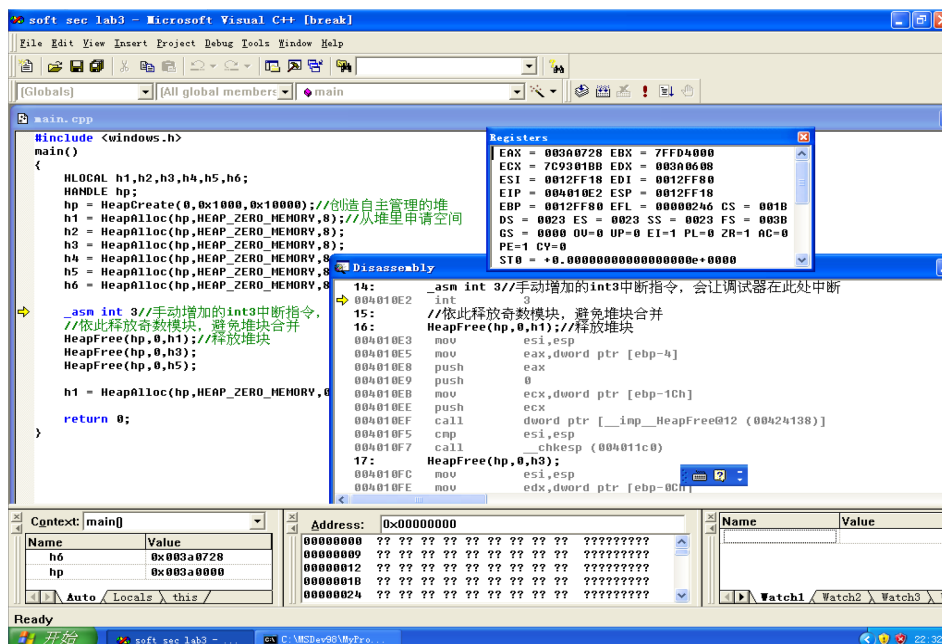
以第四章示例 4-4 代码为准，在 VC IDE 中进行调试，观察堆管理结构，记录 Unlink 节点时的双向空闲链表的状态变化，了解堆溢出漏洞下的 Dword Shoot 攻击。

## 实验过程：

1. 进入 VC6，打开提供的 4-4 代码进行调试。



2. 按 F10 进入调试界面，项目停止在手动设置的断点(\_asm int 3)处。



3. 对于代码分析得知,此时已经创建了自主管理的堆 hp,并在其中申请了 6 个堆块 h1, h2, h3, h4, h5, h6, 通过对 locals 表查询得知 h1 的值(块值)存储在 0x003a0688 处。

Name	Value
h2	0x003a06a8
h1	0x003a0688
hp	0x003a0000

由于 windows 的堆管理系统,知申请过的堆块前含有 8 字节的块首信息,包括块是否被占用、块长度等,定位至 0x003a0688-8=0x003a0680 处,看到储存信息如下:

Address	Hex	ASCII
003A0680	04 00 08 00 77 07 18 00	....w....
003A0689	00 00 00 00 00 00 00 AB	.....
003A0692	AB AB AB AB AB AB 00 00	... ..
003A069B	00 00 00 00 00 04 00 04	.....
003A06A4	73 07 18 00 00 00 00 00	s.....

其中,前八位为块首信息,随后的的 8 字节信息均初始化为 0。

4. 运行下一行代码 HeapFree(hp, 0, h1), 释放 h1 的堆,看到 0x003a0680 处信息如图变化,由于堆管理系统,该块长度为 8 字节,因此会进入 FreeLink[2]的链表中,块首信息变为 12 字节,后面的值被初始化为 EFEE,前面的两次 003a0198 代表其 flink、blink,因为此时 FreeLink[2]只有一个块,因此均指向 FreeLink[2]的地址处。

Address	Hex	ASCII
003A0680	04 00 08 00 DE 04 18 00	.....
003A0689	01 3A 00 98 01 3A 00 EE	.....铅
003A0692	EE FE EE FE EE FE EE FE	铅铅铅铅.
003A069B	FE EE FE EE FE 04 00 04	.....
003A06A4	DA 07 18 00 00 00 00 00	.....

5. 随后运行下一行代码 HeapFree(hp, 0, h3), 此时只释放奇数堆,由于为了避免堆的合并,释放 h3 的堆, 0x003a0680 处的信息如图变化:

Address	Hex	ASCII
003A0680	04 00 08 00 DE 04 18 00	.....
003A0689	06 3A 00 98 01 3A 00 EE	.....铅
003A0692	EE FE EE FE EE FE EE FE	铅铅铅铅.
003A069B	FE EE FE EE FE 04 00 04	.....
003A06A4	DA 07 18 00 00 00 00 00	.....

可以发现,代表 blink 的 003a0198 并未改变,因为 h1 仍是 FreeLink[2]的首元素,而其 flink 变为 003a06c8,指向 h3 的地址。

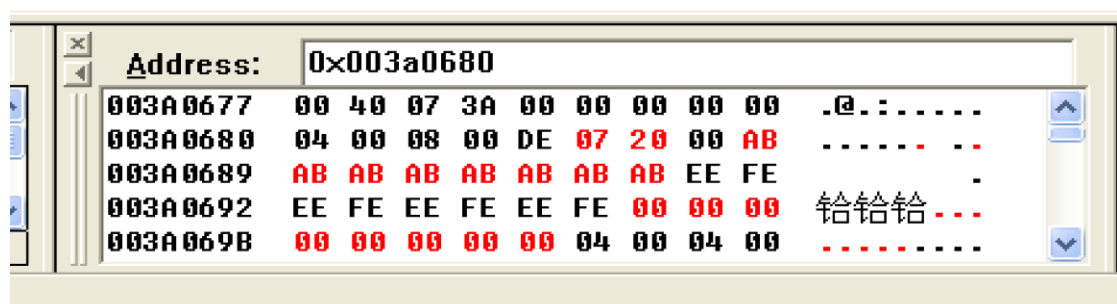
再查看 003a06c8-12=003a06bc 的位置，发现其 blink 为 003a0688，为 h1 的地址，而 flink 为 003a0198，为 FreeLink[2]的地址。

```

CPU - main thread, module Test
00401041 . B8 CCCCCCCC mov eax,CCCCCCCC
00401046 . F3:AB rep stos dword ptr [edi]
00401048 . 8B45 08 mov eax,dword ptr [ebp+8]
0040104B . 50 push eax
0040104C . 68 1C104300 push offset 0043101C
00401051 . E8 CA710000 call strcmp
00401056 . 83C4 08 add esp,8
00401059 . 8945 FC mov dword ptr [ebp-4],eax
0040105C . 33C0 xor eax,eax
0040105E . 837D FC 00 cmp dword ptr [ebp-4],0
00401062 . 0F94C0 sete al
00401065 . 5F pop edi
00401066 . 5E pop esi
00401067 . 5B pop ebx
00401068 . 83C4 44 add esp,44
0040106B . 3BEC cmp ebp,esp
0040106D . E8 3E720000 call _chkesp
00401072 . 8BE5 mov esp,ebp
00401074 . 5D pop ebp
00401075 . C3 ret
  
```

6. 类似地、执行释放 h5，可以观察到 h3 的 flink 变为 h5 的地址，h5 的 blink 为 h3 的地址，flink 为 FreeLink[2]的地址。

7. 最后进行 h1 的重新申请，运行 h1=HeapAlloc(hp,HEAP\_ZERO\_MEMORY,8);观察到 0x003a0680 如图变化，故又回到了开始的状态，而在 FreeLink 中，h3 的 blink 为 FreeLink[2]的地址，flink 不变，h5 的 blink 不变，flink 不变



## 8. Dword Shoot 漏洞分析:

由于在过程中，每次对于堆从 Freelink 的删除，都要进行前后指针的转换

```

node—>blink—>flink = node—>flink ;
node—>flink—>blink = node—>blink ;
  
```

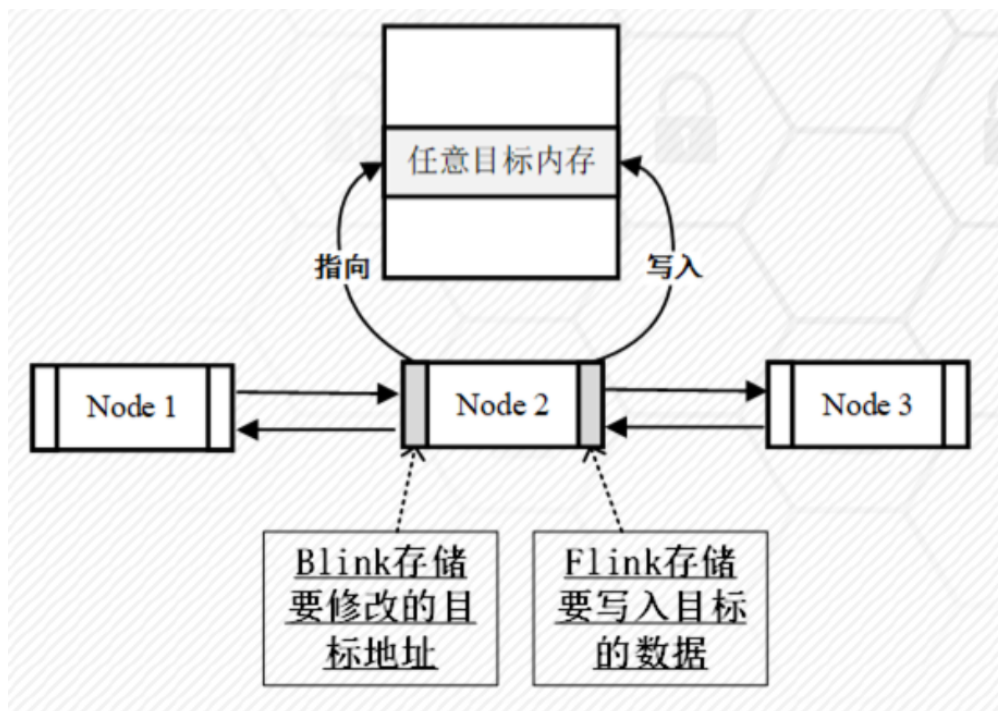
(将前指针的后指针转为当前的后指针，后指针的前指针转为当前的前指针)  
在汇编语言中，即为如下指令：

```

- mov dword ptr [edi], ecx ;
  mov dword ptr [ecx+4], edi ;
  
```

其中，ecx 为空闲可分配的堆区块的前向指针，edi 为该堆块的后向指针

因此，如果对 ecx 和 edi 的值进行修改，以第一条汇编指令为例，就可以在[edi]处植入为 ecx 的值，完成一次 Dword Shoot（对任意地址进行任意值的修改），从而产生安全问题。



#### 心得体会：

通过实验，掌握了使用 VC6 对于 windows.h 中包含的对堆操作的代码、原理的学习，深刻了解了堆控制系统，和空闲双向链表 FreeList 原理的掌握，以及申请堆、释放堆的具体电脑实现和代码流程。

同时，对 Dword Shoot 漏洞的原理进行了学习，通过实践了解了其发生的原因和利用的方式，对堆溢出漏洞有了更深刻的认识。