

《软件安全》实验报告

姓名：何叶 学号：2313487 班级：范玲玲班

一、实验名称

堆溢出Dword Shoot攻击实验

二、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。

三、实验过程

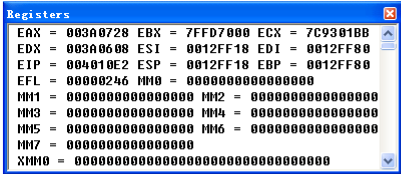
1.从源码进入反汇编模式

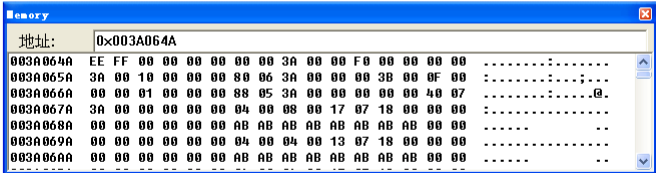
```
#include<windows.h>
main()
{
    HLOCAL h1, h2,h3,h4,h5,h6;
    HANDLE hp;
    hp = HeapCreate(0,0x1000,0x10000); //创建自主管理的堆
    h1 = HeapAlloc(hp,HEAP_ZERO_MEMORY,8); //从堆里申请空间
    h2 = HeapAlloc(hp,HEAP_ZERO_MEMORY,8);
    h3 = HeapAlloc(hp,HEAP_ZERO_MEMORY,8);
    h4 = HeapAlloc(hp,HEAP_ZERO_MEMORY,8);
    h5 = HeapAlloc(hp,HEAP_ZERO_MEMORY,8);
    h6 = HeapAlloc(hp,HEAP_ZERO_MEMORY,8);

    //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;
}
```





整个流程解析：

- (1)程序首先创建了一个大小为0x1000的堆区,并从中连续申请了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)3次释放结束后,会形成3个16字节的空闲堆块放入空表。因为是16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字节,因此会从freelist[2]所标识的空表中摘取第一个空闲堆块,即h1。
- (5)如果手动修改h1块首中的指针,应该能够观察到Dword Shoot 攻击的发生。

2.观察堆管理结构

```
#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;
}
```

Registers	
EAX	= 003A0728 EBX = 7FFD8000
ECX	= 7C9301B8 EDX = 003A0608
ESI	= 0012FF18 EDI = 0012FF80
EIP	= 004010E5 ESP = 0012FF18
EBP	= 0012FF80 EFL = 00000246
MM0	= 0000000000000000
MM1	= 0000000000000000
MM2	= 0000000000000000
MM3	= 0000000000000000

Memory	
地址:	0x003a0680
003A0680	04 00 00 00 73 07 18 00 00 00 00 00 00 00 00 00 AB AB ...S.....
003A0692	AB AB AB AB AB 00 00 00 00 00 00 00 00 00 04 00 04 00
003A06A4	77 07 18 00 00 00 00 00 00 00 00 00 00 AB AB AB AB w.....
003A06B6	AB AB 00 00 00 00 00 00 00 00 00 04 00 04 00 7B 07 18 00{...
003A06C8	00 00 00 00 00 00 00 00 00 AB AB AB AB AB AB AB 00 00
003A06DA	00 00 00 00 00 00 04 00 04 00 7F 07 18 00 00 00 00 00
003A06EC	00 00 00 00 AB AB AB AB AB AB AB AB 00 00 00 00 00 00
003A06FE	00 00 04 00 04 00 43 07 18 00 00 00 00 00 00 00 00 00C.....
003A0710	AB AB AB AB AB AB AB 00 00 00 00 00 00 00 00 00 04 00
003A0722	04 00 47 07 18 00 00 00 00 00 00 00 00 00 AB AB AB AB ..G.....

打断点进入调试反汇编,看到h1的地址为0x003a0688,块首地址为0x003a0680;

```
#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;
}
```

Registers	
EAX	= 00000001 EBX = 7FFD8000
ECX	= 7C93003D EDX = 003A0608
ESI	= 0012FF18 EDI = 0012FF80
EIP	= 004010FC ESP = 0012FF18
EBP	= 0012FF80 EFL = 00000246
MM0	= 0000000000000000
MM1	= 0000000000000000
MM2	= 0000000000000000
MM3	= 0000000000000000

Memory	
地址:	0x003a0680
003A0680	04 00 00 00 73 04 18 00 98 01 3A 00 98 01 3A 00 EE FE ...S.....拾
003A0692	EE FE EE FE EE FE EE FE EE FE EE FE 04 00 04 00 拾拾拾拾拾拾拾拾...
003A06A4	77 07 18 00 00 00 00 00 00 00 00 00 00 AB AB AB AB w.....
003A06B6	AB AB 00 00 00 00 00 00 00 00 00 04 00 04 00 7B 07 18 00{...
003A06C8	00 00 00 00 00 00 00 00 00 AB AB AB AB AB AB AB 00 00
003A06DA	00 00 00 00 00 00 04 00 04 00 7F 07 18 00 00 00 00 00
003A06EC	00 00 00 00 AB AB AB AB AB AB AB AB 00 00 00 00 00 00
003A06FE	00 00 04 00 04 00 43 07 18 00 00 00 00 00 00 00 00 00C.....
003A0710	AB AB AB AB AB AB AB 00 00 00 00 00 00 00 00 00 04 00
003A0722	04 00 47 07 18 00 00 00 00 00 00 00 00 00 AB AB AB AB ..G.....

按f10跳转,看0x3a0680地址,其中,flink和blink都为003a0198,指向同一个地址,该地址即为freelist[2];

Memory	
地址:	0x003a0198
003A0198	88 06 3A 00 88 06 3A 00 A0 01 3A 00 A0 01 3A 00 A8 01 ..
003A01AA	3A 00 A8 01 3A 00 B0 01 3A 00 B0 01 3A 00 B8 01 3A 00 :.
003A01BC	B8 01 3A 00 C0 01 3A 00 C0 01 3A 00 C8 01 3A 00 C8 01 ..
003A01CE	3A 00 D0 01 3A 00 D0 01 3A 00 D8 01 3A 00 D8 01 3A 00 :.
003A01E0	E0 01 3A 00 E0 01 3A 00 E8 01 3A 00 E8 01 3A 00 F0 01 ..
003A01F2	3A 00 F0 01 3A 00 F8 01 3A 00 F8 01 3A 00 00 02 3A 00 :.
003A0204	00 02 3A 00 08 02 3A 00 08 02 3A 00 10 02 3A 00 10 02 ..
003A0216	3A 00 18 02 3A 00 18 02 3A 00 20 02 3A 00 20 02 3A 00 :.
003A0228	28 02 3A 00 28 02 3A 00 30 02 3A 00 30 02 3A 00 38 02 {.
003A023A	3A 00 38 02 3A 00 40 02 3A 00 40 02 3A 00 48 02 3A 00 :.

跳转003a0198地址,看到freelist[2]的flink和blink都指向003a0688,为h1释放的地址, freelist中链入了h1的空闲堆块;

```
#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);

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

    h1 = HeapAlloc(hp,HEAP_ZERO_MEMORY,8);

    return 0;
}
```

Registers	
EAX = 00000001	EBX = 7FFD8000
ECX = 7C93003D	EDX = 003A0608
ESI = 0012FF18	EDI = 0012FF80
EIP = 00401115	ESP = 0012FF18
EBP = 0012FF80	EFL = 00000246
MM0 = 0000000000000000	
MM1 = 0000000000000000	
MM2 = 0000000000000000	
MM3 = 0000000000000000	

Memory	
地址:	0x003a0198
003A0198	88 06 3A 00 C8 06 3A 00 A0 01 3A 00 A0 01 3A 00 A8 01
003A019A	3A 00 A8 01 3A 00 B0 01 3A 00 B0 01 3A 00 B8 01 3A 00
003A019C	B8 01 3A 00 C0 01 3A 00 C0 01 3A 00 C8 01 3A 00 C8 01
003A019E	3A 00 D0 01 3A 00 D0 01 3A 00 D8 01 3A 00 D8 01 3A 00
003A01A0	E0 01 3A 00 E0 01 3A 00 E8 01 3A 00 E8 01 3A 00 F0 01
003A01A2	3A 00 F0 01 3A 00 F8 01 3A 00 F8 01 3A 00 00 02 3A 00
003A01A4	00 02 3A 00 08 02 3A 00 08 02 3A 00 10 02 3A 00 10 02
003A01A6	3A 00 18 02 3A 00 18 02 3A 00 20 02 3A 00 20 02 3A 00

按f10释放h3后, 观察003a0680, h1的flink改变为h3的堆块地址003a06c8;

跳转入003a06c8后观察到flink指向freelist[2], blink指向h1的堆块块身;

```
#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);

    //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;
}
```

Registers	
EAX = 00000001	EBX = 7FFD8000
ECX = 7C93003D	EDX = 003A0608
ESI = 0012FF18	EDI = 0012FF80
EIP = 0040112E	ESP = 0012FF18
EBP = 0012FF80	EFL = 00000246
MM0 = 0000000000000000	
MM1 = 0000000000000000	
MM2 = 0000000000000000	
MM3 = 0000000000000000	

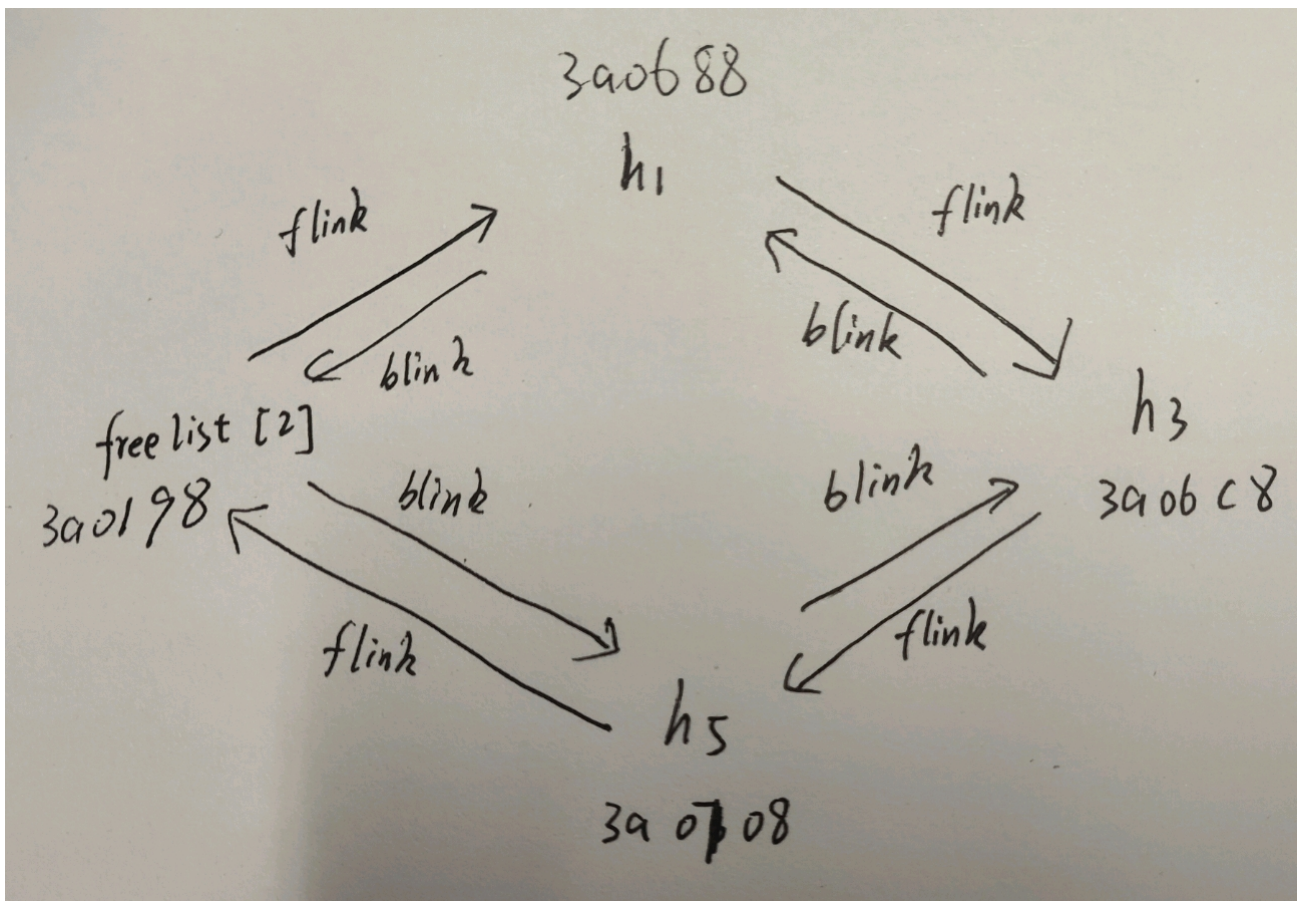
Memory	
地址:	0x003a0198
003A0198	88 06 3A 00 08 07 3A 00 A0 01 3A 00 A0 01 3A 00 A8 01
003A019A	3A 00 A8 01 3A 00 B0 01 3A 00 B0 01 3A 00 B8 01 3A 00
003A019C	B8 01 3A 00 C0 01 3A 00 C0 01 3A 00 C8 01 3A 00 C8 01
003A019E	3A 00 D0 01 3A 00 D0 01 3A 00 D8 01 3A 00 D8 01 3A 00
003A01A0	E0 01 3A 00 E0 01 3A 00 E8 01 3A 00 E8 01 3A 00 F0 01
003A01A2	3A 00 F0 01 3A 00 F8 01 3A 00 F8 01 3A 00 00 02 3A 00
003A01A4	00 02 3A 00 08 02 3A 00 08 02 3A 00 10 02 3A 00 10 02

Memory	
地址:	0x003a06c8
003A06C8	08 07 3A 00 98 01 3A 00 EE FE EE FE EE FE EE FE EE FE
003A06DA	EE FE EE FE EE FE 04 00 04 00 7F 07 18 00 00 00 00 00
003A06EC	00 00 00 00 AB AB AB AB AB AB AB AB 00 00 00 00 00
003A06FE	00 00 04 00 04 00 43 04 18 00 98 01 3A 00 C8 06 3A 00
003A0710	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 04 00

释放h5, 查看h3块身地址, flink指向h5地址;

观察h5的地址, flink指向freelist[2],blink指向h3地址;

此时, 各堆块:



相当于freelist[2],h1,h3,h5的双向链表;

```
#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;
}
```

Registers			
EAX	= 003A0688	EBX	= 7FFD8000
ECX	= 7C9301BB	EDX	= 003A0608
ESI	= 0012FF18	EDI	= 0012FF80
EIP	= 00401148	ESP	= 0012FF18
EBP	= 0012FF80	EFL	= 00000246
MM0	= 0000000000000000		
MM1	= 0000000000000000		
MM2	= 0000000000000000		
MM3	= 0000000000000000		

Memory	
地址:	0x003A0198
003A0198	C8 06 3A 00 08 07 3A 00 A0 01 3A 00 A0 01 3A 00 A8 01
003A01AA	3A 00 A8 01 3A 00 B0 01 3A 00 B0 01 3A 00 B8 01 3A 00
003A01BC	B8 01 3A 00 C0 01 3A 00 C0 01 3A 00 C8 01 3A 00 C8 01
003A01CE	3A 00 D0 01 3A 00 D0 01 3A 00 D8 01 3A 00 D8 01 3A 00
003A01E0	E0 01 3A 00 E0 01 3A 00 E8 01 3A 00 E8 01 3A 00 F0 01
003A01F2	3A 00 F0 01 3A 00 F8 01 3A 00 F8 01 3A 00 00 02 3A 00
003A0204	00 02 3A 00 08 02 3A 00 08 02 3A 00 10 02 3A 00 10 02
003A0216	20 0A 10 02 20 0A 10 02 20 0A 20 02 20 0A 20 02 20 0A

Memory	
地址:	0x003A06C8
003A06C8	08 07 3A 00 98 01 3A 00 EE FE EE FE EE FE EE FE EE FE
003A06DA	EE FE EE FE EE FE 04 00 04 00 7F 07 18 00 00 00 00 00
003A06EC	00 00 00 00 AB AB AB AB AB AB AB AB 00 00 00 00 00
003A06FE	00 00 04 00 04 00 43 04 18 00 98 01 3A 00 C8 06 3A 00
003A0710	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 04 00

按f10, 重新分配一个块身为8字节的堆来取下h1地址, 观察freelist[2]地址, 此时flink变为h3地址, 说明h1的flink写入h1的blink;

观察003A06C8, h3的blink变为freelist[2],说明h1的blink写入h1的flink;

卸下一个堆块的时候, 会将其flink和blink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。

四、心得体会

在本次实验中，我掌握了在VC6环境下堆的创建与释放方法，并且意识到到栈溢出和堆溢出的危害性。通过追踪堆块内存位置，我对堆表的内存管理机制有了更深入的理解，包括堆表合并、空闲表链接等关键知识点，以及flink和blink两个指针的变化规律。同时，我掌握了Dword Shoot攻击的原理，即通过构造特定地址和数据，在空闲堆块从链表中移除的瞬间，获得向任意内存地址写入任意数据的机会。