

《软件安全》实验报告

姓名：邢清画 学号：2211999 班级：1023

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. 代码分析

```
#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 //手动增加的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;
}
```

1. 运行实验代码，停止在 int3 断点处，创建大小为 0x1000 的堆区，此时已经完成了 h1 至 h6 的堆块初始化，对于每个堆块，申请 8 字节+块首 8 字节=16 字节的堆块。
2. 释放三次奇数次申请后，形成三个 16 个字节的空闲堆块放入空表。
3. 16 个字节依次放入 freelist[2]所标识的空表。
4. 再次申请 8 字节的堆区内存，从 freelist[2]所标识的空表中摘取第一个空闲堆块（h1）。
5. 手动修改 h1 块首中的指针，可观察到 DWORD SHOOT。

2. DEBUG 观察堆内存变化

1. 释放 h1:

执行 HeapFree(hp, 0, h1) 语句前，hp 为 0x003A0000, h1 为 0x003A0688，加上块首为 0x003A0680，执行后内存变化如下：

The screenshot shows the Visual Studio IDE with the code from the previous block. The execution has stopped at the line `HeapFree(hp, 0, h1);`. The registers window shows the following values:

Register	Value
EAX	00000001
ECX	7C93003D
EDX	003A0608
ESI	0012FF18
EDI	0012FF80
EIP	004010FC
ESP	0012FF18
EBP	0012FF80
EFL	00000246
CS	001B
DS	0023
ES	0023
SS	0023
FS	003B
GS	0000
OV	0
UP	0
EI	1
PL	0
ZR	1
AC	0
PE	1
CV	0
ST0	+0.0000000000000000e+0000

The memory window shows the address 0x003A0680. The memory contents are as follows:

Address	Value
003A066C	01 00 00 00 88 05 3A 00 00 00 00 00 40 07 3A 00
003A0680	04 00 08 00 E7 04 18 00 08 01 3A 00 08 01 3A 00
003A0694	EE FE EE FE EE FE EE FE EE FE EE FE EE FE EE FE
003A06A8	00 00 00 00 00 00 00 00 AB AB AB AB AB AB AB AB

只有 h1 放进来，freelist[2]的地址(0x003A0198)，freelist[2]的前后向指针都指向 h1(0x003A0688)，即释放的 h1 的块身的首地址，freelist[2]唯一后继节点是空闲的 h1 块(0x003A0688)。

Address:	0x003A0198
003A0198	88 06 3A 00 88 06 3A 00 A0 01 3A 00
003A01A4	A0 01 3A 00 A8 01 3A 00 A8 01 3A 00
003A01B0	B0 01 3A 00 B0 01 3A 00 B8 01 3A 00

2. 释放 h3:

执行 HeapFree (hp, 0, h3) 前 h3 块身的首地址为 0x003A06C8，执行完 HeapFree (hp, 0, h3) 之后，freelist[2]的Blink 指向 h3 块身首地址 0x003A06C8。

Address:	0x003A0198
003A0184	80 01 3A 00 88 01 3A 00 88 01 3A 00 90 01 3A 00
003A0198	88 06 3A 00 C8 06 3A 00 A0 01 3A 00 A0 01 3A 00
003A01AC	A8 01 3A 00 B0 01 3A 00 B0 01 3A 00 B8 01 3A 00

返回 0x003A0680 查看，前向指针 Flink 也改变到 0x003A06C8，即 h3 变到了 h1 后面。

Address:	0x003A0680
003A066C	01 00 00 00 88 05 3A 00 00 00 00 00 40 07 3A 00 00 00 00
003A0680	04 00 08 00 E7 04 18 00 C8 06 3A 00 98 01 3A 00 EE FE EE
003A0694	EE FE EE FE EE FE EE FE EE FE EE FE 04 00 04 00 E3 07 18
003A06A8	00 00 00 00 00 00 00 00 AB AB AB AB AB AB AB AB 00 00 00

3. 释放 h5:

执行 HeapFree (hp, 0, h5) 前，h5 块身首地址为 0x003A0708。

继续按 F10 释放，变成 0x003A0708，即 h5 块身首地址，h5 放到了 freelist[2]的末尾。

Address:	0x003A0198
003A0184	80 01 3A 00 88 01 3A 00 88 01 3A 00 90 01 3A 00
003A0198	88 06 3A 00 08 07 3A 00 A0 01 3A 00 A0 01 3A 00
003A01AC	A8 01 3A 00 B0 01 3A 00 B0 01 3A 00 B8 01 3A 00

继续 F101，0x003A06C8 是 h3，在 freelist 的第一个空闲块。

Address:	0x003A0198
003A0198	C8 06 3A 00 08 07 3A 00 A0 01 3A 00 A0 01 3A 00
003A01AC	A8 01 3A 00 B0 01 3A 00 B0 01 3A 00 B8 01 3A 00
003A01C0	C0 01 3A 00 C0 01 3A 00 C8 01 3A 00 C8 01 3A 00

输入 h3 的地址 0x003A06C8，Blink 变成 0x003A0198，即 freelist 的第二个。

Address:	0x003A06C8
003A06C8	08 07 3A 00 98 01 3A 00 EE FE EE FE EE FE EE FE EE FE
003A06DC	EE FE EE FE 04 00 04 00 1C 07 18 00 00 00 00 00 00 00
003A06F0	AB AB AB AB AB AB AB AB 00 00 00 00 00 00 00 00 04 00

执行完 HeapFree (hp, 0, h3) 和 HeapFree (hp, 0, h5) 后，h1, h2, h5 的前后指针状态依次为：0x003A0688，0x003A06C8，0x003A0708。

4. 执行 HeapAlloc (hp, HEAP_ZERO_MEMORY, 8) 语句时，需要从 freelist[2]中摘下 16 字节的堆块，摘下 h1 观察内存区 freelist[2]和 h3 的指针。freelist[2]的Flink 被修改为了 0x003A06C8 即 h3, h3 的Blink 被修改为了 0x003A0198，即 freelist[2]。

Address:	0x003A0198	Address:	0x003A06C8
003A0198	C8 06 3A 00	003A06C8	98 07 3A 00 98 01 :

5. Dword Shoot 攻击:

通过一定办法将 h1 的 Flink 和 Blink 改写成特定地址和特定数值,即完成一次 Dword Shoot 攻击。

3. 心得体会:

通过实验,掌握了堆的管理方式,以及空表的结构,存储方式,在进行内存分配或释放时,操作系统的堆管理器会更新这些管理结构,以反映当前堆的状态。

双向链表确保了内存的高效利用和回收。

了解了通过堆溢出漏洞来进行 Dword Shoot 攻击的有效手段。