《漏洞利用及渗透测试基础》实验报告

姓名: 平世龙 学号: 2012656 班级: 1074

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 编写代码如下:

```
#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;
```

- 2. 插入断点,调试程序,进入反汇编模式。
- 3. 创建块首及6个堆块并赋初值。

Name	Value
h1	Охоссоссо
h2	Oxcccccc
h3	Oxcccccc
h4	Oxcccccc
h5	Oxcccccc
h6	Oxcccccc
hp	Oxcccccc

4. 创建大小为 0x1000 的堆区,并为每个堆块申请大小为 8 字节的空间:

=	1 H3 = 1
Name	Value(块身起始位置)
h1	0x003a0688
h2	0x003a06a8
h3	0x003a06c8

h4	0x003a06e8
h5	0x003a0708
h6	0x003a0728
hp	0x003a0000

可以计算出块首起始位置为 0x003a0680。

5. 执行 HeapFree(hp, 0, h1)语句时,查看相关内存变化。

可见,除了块首的状态发生变化外,0x003a0688 开始的块身位置的前 8 个字节(Flink 和 Blink 指针)也发生了变化,由 0x000000 变为具体的地址。可以发现这是第一个堆块释放,第一个堆块被释放后将被链入到 freelist[2]空表中,而此时其Flink 和 Blink 的值都是 0x003a0198,也是 freelist[2] 的地址。我们到 0x003a0198 处观察内存为:

```
Address: Dx003-013-02

BI33-019-08

BI33-08

BI33-08
```

freelist[2]的 Flink 和 Blink 都是 0x003a0688。意味着,当前 freelist[2]唯一后继节点就是刚刚空闲的 h1 块(地址为 0x003a0688),而 h1 块是唯一前继节点是 freelist[2]。其它地址(freelist[3]、freelist[4]、freelist[5])的 Flink 和 Blink 均指向自身,说明都是空表。

- 6. 同理依次执行 HeapFree(hp, 0, h3)和 HeapFree(hp, 0, h5)后:
- (1) freelist[2] (地址为 0x003a0198): Flink (前 4 个字节) 为 0x003a0688, Blink (后 4 个字节) 为 0x003a0708。

Address:	0×003a0198									
003A0136	00	00	00	00	00	00	00	00		
003A0167	00	00	00	00	00	01	00	00		
003A0198	88	96	3A	00	08	07	3A	00		
003A01C9	01	3A	00	C8	01	3A	00	DØ		
003A01FA	3A	00	F8	01	3A	00	00	02		
003A022B	00	28	02	3A	00	30	02	3A		
003A025C	58	02	3A	00	60	02	3A	00		
0000000	00	20	aa	00	an	20	00	00		

(0) 11 (likk) 4 0 000 0000) F1: 1 4 0

(2) h1 (地址为 0x003a0688): Flink 为 0x003a06c8, Blink 为 0x003a0198。

Address:	0>	0×003a0688								
003A0626	00	00	00	00	00	00	00	00		
003A0657	00	00	00	3A	00	10	00	00		
003A0688	C8	06	3A	00	98	01	3A	00		
003A06B9	00	00	00	00	00	00	00	04		
003A06EA	00	00	00	00	00	00	AB	AB		
003A071B	FE	EE	FE	EE	FE	04	00	04		
003A074C	78	01	3A	00	EE	FE	EE	FE		
00000770	гг	гг	гг	гг	гг	гг	гг	гг		

(3) h3 (地址为 0x003a06c8): Flink 为 0x003a0708, Blink 为 0x003a0688。

Address:	0>	<003	3a06	ic8				
003A0666	38	00	OF	00	00	00	01	00
003A0697	FE	EE	FE	EE	FE	EE	FE	EE
003A06C8	08	07	3A	00	88	96	3A	00
003A06F9	00	00	00	00	00	00	00	04
003A072A	00	00	00	00	00	00	AB	AB
003A075B	FE	EE	FE	EE	FE	EE	FE	EE
003A078C	EE	FE	EE	FE	EE	FE	EE	FE
00000700	гг	гг	гг	гг	гг	гг	гг	гг

可知,此时 freelist[2]链表状态为: freelist[2]<=>h1<=>h3<=>h5

7. 执行 HeapAlloc(hp, HEAP ZERO MEMORY, 8) 语句

此时,当再次分配空间的时候,从 freelist[2]的双向链表里摘下一块大小为 16 字节的堆块, 首先摘得 h1(地址为 0x003a0688)。

(1)freelist[2](地址为 0x003a0198)的前 4 个字节变为 0x003a06c8,实际发生了将 h1 后向指针(值为 0x003a0198)地址处的值写为 h1 前向指针的值。

Address:	0>	0×003a0198						
003A 0136	00	00	00	00	00	00	00	00
003A0167	00	00	00	00	00	01	00	00
003A0198	C8	96	3A	00	08	97	3A	00
003A01C9	01	3A	00	C8	01	3A	00	DO
003A01FA	3A	00	F8	01	3A	00	00	02
003A022B	00	28	02	3A	00	30	02	3A
003A025C	58	02	3A	00	60	02	3A	00
4000000	00	20	00	00	90	20	aa	00

(2) h3(地址为 0x003a06c8)的 Blink 变为 h1->Blink,即 0x003a0198,实际发生了将 h1 前向指针(值为 0x003a06c8)地址处的值写为 h1 后向指针的值。

Address:	0>	0×003a06c8								
003A0666	3B	00	OF	00	00	00	01	00		
003A0697	AB	00	00	00	00	00	00	00		
003A06C8	08	07	3A	00	98	01	3A	00		
003A06F9	00	00	00	00	00	00	00	04		
003A072A	00	00	00	00	00	00	AB	AB		
003A075B	FE	EE	FE	EE	FE	EE	FE	EE		
003A078C	EE	FE	EE	FE	EE	FE	EE	FE		
00000000	гг	гг	гг	гг	гг	гг	гг	гг		

心得体会:

通过实验,掌握了堆溢出漏洞的相关概念;

通过实验,认识了堆管理结构;

通过实验,了解了堆管理过程中内存管理过程,掌握了堆块操作的详细过程;

通过实验,进一步熟悉了汇编语言。