《软件安全》实验报告

姓名: 李雅帆 学号: 2213041 班级: 信安班

一、实验名称:

堆溢出 Dword Shoot 模拟实验

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.流程解析:

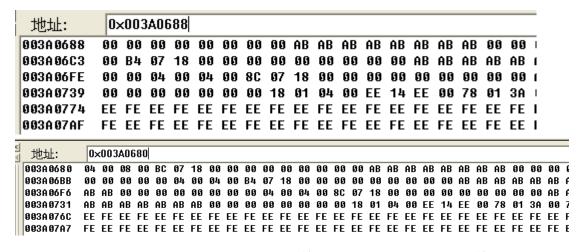
- (1) 程序首先创建了一个大小为 0x1000 的堆区, 并从其中连续申请了 6 个块身大小为 8 字节的堆块, 加上块首实际上是 6 个 16 字节的堆块。
 - (2) 释放奇数次申请的堆块是为了防止堆块合并的发生。
- (3) 三次释放结束后,会形成三个 16 个字节的空闲堆块放入空表。因为是 16 个字节,所以会被依次放入 freelist[2]所标识的空表,它们依次是 h1、h3、h5。
- (4) 再次申请 8 字节的堆区内存,加上块首是 16 个字节,因此会从 freelist[2] 所标识的空表中摘取第一个空闲堆块出来,即 h1。
- (5) 如果我们手动修改 h1 块首中的指针, 应该能够观察到 DWORD SHOOT 的发生。

2.实验过程

在 VC6 中创建新的工程,并进入 DeBug 模式。

(1) 执行 HeapFree(hp,0,h1)语句时。

执行 HeapFree(hp,0,h1)前, h1 块身的首地址为 0x003a0688, 对应的块首的 起始地址为 0x003a0680。



当执行 HeapFree(hp,0,h1)后,块身的前 8 个字节发生了变化,前 8 个字节分表示的是 Flink 和 Blink,二者的值都是 0x003a0198,即 2 是 freelist[2]的地址。 我们转到 freelist[2]的地址处可以发现,freelist[2]的 Flink 和 Blink 都指向了 0x003a0688,这就是我们释放的 h1 的块身的首地址,由于 freelist[2]中目前只有 h1 一个堆块,因此两个值是完全相同的。

地址:	0>	<003/	10680																														
003A0680	64	00	98 96) A8	64	18	00 9	8 0	1 3	A O	0 98	01	3A	99	EE	FE	EE	FE E	ΕF	E EE	FE	EE	FE	EE	FE	EE	FE	EE	FE	94	99	64	99 A
003A06BC	00	00	90 00	04	99	04	00 f	10 0	7 1	8 9	0 00	00	99	00	00	00	00	00 A	B A	B AB	AB	AB	AB	AB	AB	00	00	00	00	00	00	00	99 (
003A06F8	00	00	90 00	9 9 9	00	00	00 (94 0	0 0	40	0 98	97	18	00	00	00	00	00 6	0 0	0 00	99	AB	AB	AB	AB	AB	AB	AB	AB	00	99	99	99 (
003A0734	AB	AB I	AB AB	99	99	00	00 (90 0	0 0	0 0	0 18	01	64	99	EE	14	EE	00 7	8 0	1 3A	99	78	91	3A	00	EE	FE	EE	FE	EE	FE	EE	FE E
003A0770	EE	FE I	EE FE	EE	FΕ	EE	FE E	E F	ΕE	ΕF	E EE	FE	EE	FΕ	EE	FΕ	EE	FE E	ΕF	E EE	FE	ΕE	FΕ	EE	FΕ	EE	FΕ	EE	FΕ	EE	FΕ	EE	FE E
100010710																																	
1	Will be accessed																																
	地址: 0×003A0198																																
地址:		0:	×003	3A0	198	l																											
j 地址: 003A019	98	0: 88		3A0 3A			3 0	6 3	i A	00	AO	01	31	A (30	AO	01	3A	91	3 A	3 6	1 :	3A	00	A	8 (91	3A	00	9 B	0	01	3A
4			96		00	8				00 00						A 0	01 01		_			-	3A 3A	00			91 91	3A 3A			_		3A 3A
003A019	3C	88	96 91	3A	00	9 8:	9 0	1 3	A		CO	01	3	A (30			3A	0	9 C	8 6	1			D	0 (0	9 D	0	01	
003A019 003A011 003A011	BC E 0	88 B8	96 91 91	3A 3A	00	9 8 9 C	9 0	1 3	A	00 00	C 0	01 01	3	A (30 30	E8	01 01	3A	0	9 C	9 0	1 :	3A 3A	00 00	DI F	0 (91 91	3A 3A	00	9 D 9 F	8	01 01	3A 3A
003A019 003A011 003A011 003A02	BC E 0 04	88 B8 E 0	96 91 91 92	3A 3A 3A 3A	00 00 00	9 8: 9 C 9 E	9 9 9 9 8 9	1 3 1 3 2 3	A A	00 00	C 0 E 8	01 01 02	31	A (30 30 30	C8 E8 10	01 01 02	3A 3A	01 01	9 C(9 F) 9 1	3	1 1 2	3A 3A 3A	00 00	DI FI	0 (0 (8 (91 91 92	3A 3A 3A	0(0(9 D 9 F 9 1	8	01 01 02	3A 3A 3A
003A019 003A011 003A011 003A029	BC E 0 04 28	88 B8 E 0 00 28	06 01 01 02 02	3A 3A 3A 3A	0(0(0(0(9 8: 9 C 9 E 9 0:	9 9 9 9 8 9	1 3 1 3 2 3 2 3	A A A	00 00 00	08 08 08	01 01 02 02	31 31 31	A (30 30 30	C8 E8 10 30	01 01 02 02	3A 3A 3A	() () () ()	3 Ct 3 Ft 3 1t	8 6 9 6 9 6	1 2 2 2	3A 3A 3A 3A	00 00 00	Di Fi 18	0 (0 (8 (8 (91 91 92 92	3A 3A 3A 3A	0 (0 (0 (9 D 9 F 9 1	8	01 01 02 02	3A 3A 3A 3A
003A019 003A011 003A011 003A02	BC E 0 04 28	88 B8 E 0	06 01 01 02 02	3A 3A 3A 3A	0(0(0(0(9 8: 9 C 9 E 9 0:	9 9 9 9 8 9	1 3 1 3 2 3 2 3	A A A	00 00	08 08 08	01 01 02 02	31 31 31	A (30 30 30	C8 E8 10	01 01 02	3A 3A 3A	() () () ()	3 Ct 3 Ft 3 1t	8 6 9 6 9 6	1 2 2 2	3A 3A 3A	00 00	Di Fi 18	0 (0 (8 (8 (91 91 92	3A 3A 3A	0 (0 (0 (9 D 9 F 9 1	8	01 01 02 02	3A 3A 3A
003A019 003A011 003A011 003A029	BC E 0 04 28 4C	88 B8 E 0 00 28	96 91 91 92 92	3A 3A 3A 3A	0 (0 (0 (0 (8; 0 C 0 E 0 0; 0 2;	9 0 9 0 8 0 8 0	1 3 1 3 2 3 2 3 2 3	IA IA IA IA	00 00 00	C 0 E 8 0 8 3 0 5 0	01 01 02 02	3: 3: 3: 3:	A (A	30 30 30 30	C8 E8 10 30	01 01 02 02	3A 3A 3A 3A	01 01 01	9 C(9 F) 9 1) 9 3)	3 6 9 6 9 6 8 6 8 6	1 2 2 2	3A 3A 3A 3A	00 00 00	DI FI 18 30	0 (0 (8 (8 (91 91 92 92	3A 3A 3A 3A	0 (0 (0 (9 D 9 F 9 1 9 4	8 8 8 10	01 01 02 02 02	3A 3A 3A 3A

(2) 依次执行 HeapFree(hp,0,h3)和 HeapFree(hp,0,h5)后。

执行完 HeapFree(hp,0,h3)后,我们注意到此时 freelist[2]的 Blink 的值已经指向了 h3 块身的首地址。

-																							
1	地址:	0>	(003	3A01	98																		
Ī	003A0198	88	96	3A	99	C8	96	3A	00	ΑØ	01	3A	00	ΑØ	01	3A	00	A8	01	3A	00	A8	01
ı	003A01BC	B8	01	3A	00	CO	91	3A	00	CO	01	3A	00	C8	01	3A	00	C8	01	3A	00	DØ	01
ı	003A01E0	ΕØ	01	3A	00	ΕØ	91	3A	00	E8	01	3A	00	E8	01	3A	00	FØ	01	3A	00	FØ	01
ı	003A0204	99	02	3A	00	98	02	3A	00	98	02	3A	00	10	02	3A	00	10	02	3A	00	18	02
ı	003A0228	28	02	3A	00	28	02	3A	00	30	02	3A	00	30	02	3A	00	38	02	3A	00	38	02
ı	003A024C	48	02	3A	99	50	02	3A	00	50	02	3A	00	58	02	3A	00	58	02	3A	00	60	02
ı	003A0270	70	02	3A	99	70	02	3A	00	78	02	3A	00	78	02	3A	00	80	02	3A	00	80	02
	I · ·																						

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, h5 放到了 freelist 的末尾。

地址: 0×003A0198																						
003A0198	88	96	3A	00	88	96	3A	00	AØ	01	3A	00	ΑØ	01	3A	00	A8	91	3A	00	A8	01
003A01BC	B8	01	3A	00	CO	01	3A	00	CØ	01	3A	00	C8	01	3A	00	C8	91	3A	00	DØ	01
003A01E0	ΕØ	01	3A	99	ΕØ	01	3A	00	E8	01	3A	00	E8	01	3A	00	FØ	91	3A	99	FØ	01
003A0204	99	02	3A	00	98	02	3A	00	98	02	3A	00	10	02	3A	00	10	02	3A	00	18	02
003A0228	28	02	3A	00	28	02	3A	00	30	02	3A	00	30	02	3A	00	38	92	3A	00	38	02
003A024C	48	02	3A	00	50	02	3A	00	50	02	3A	00	58	02	3A	00	58	02	3A	00	60	-
003A0270	70	02	3A	00	70	02	3A	00	78	02	3A	99	78	02	3A	00	80	02	3A	00	80	02
地址: 0×003A0198																						
003A016	18	90	9 ()	10	00	99	0	g	00	00	9	9 (90	99	90) [0	00	00	ı	0	00
003A012	C	00	9 0	0	00	00	0	0	00	00	91	9 (30	99	90	9 6	0	00	00	0	0	00
003A015	0	90	9 0	0	00	99	0	0	99	00	9	9 1	10	99	90	9 6	0	00	00	0	0	00
003A017	4	90	9 0	0	00	00	1 4	8	97	3A	91	g 1	18	07	36	1 8	0	80	01	3	A	00
003A019	8	88	3 0	6	3A	00	0	8	97	3A	0	9 6	10	01	36	1 8	0	ΑØ	01	3	A	00
003A01B	C	B8	8 0	11	3A	00	C	0	91	3A	0	9 (0 ;	01	36	0	0	С8	01	3	A	00

(3) 执行 HeapAlloc(hp,HEAP_ZERO_MEMORY,8)语句时。

执行完语句,可以看到, freelist[2]的 Blink 变成了 h3 块身的首地址, h3 的 Flink 变成了 freelist[2]的地址, h1 块身中目前没有数据,全部为 0。

4	地址:	0:	<00	3A0	198																
ī	003A0108	00	00	00	00	00	00	00	00	00	00	9 00	9 0	0 0	0 0	0 0	0				
	003A012C	99	00	00	00	00	00	00	00	00	00	9 01	9 0	0 0	0 0	0 0	0				
	003A0150	00	00	99	00	00	00	00	00	16	00	9 00	9 0	0 0	0 0	0 0	0				
	003A0174	99	00	00	00	48	97	3A	00	48	97	36	A 0	08	0 0	1 3	A				
	003A0198	C8	96	3A	00	98	97	3A	00	A	01	36	A 0	0 A	0 0	1 3	A				
	003A01BC	B8	01	3A	00	CO	91	3A	99	CE	01	36	A 0	0 C	8 0	1 3	A				
_	,																				
-	地址:	0×	003	A06	88																
ľ	003A05F8	00	00	00	00	00	00	00	00	00	00	00	00	00	0	9 0	9 0	0 T	1 B		
	003A061C	00	00	00	00	00	99	00	00	00	00	99	00	90	9	9 0	9 9	0 0	10		
	003A0640	08	00	C8	00	00	01	00	00	EE	FF	EE	FF	00	0	9 0	9 9	0 0	90		
	003A0664	00	00	3B	99	ØF	00	00	00	01	00	00	00	88	09	3 3 1	A 0	0 0	90		
	003A0688	00	00	00	00	00	00	00	00	ΑB	ΑB	AB	AB	AB	AE	3 AI	3 A	В	10		
<	地址:	0>	(003	A06	C8																
П	003A0638	99	00	00	00	00	00	99	00	98	00	C8	00	00	01	00	00	EE	FF	EE	FF
	003A065C	10	00	00	00	80	96	3A	00	99	99	3B	99	0F	99	99	00	01	99	00	00
	003A0680	94	00	98	99	A8	07	18	00	00	00	00	00	00	00	00	00	AB	AB	AB	AB
	003A06A4	AC	07	18	99	00	00	00	00	00	00	00	00	ΑB	ΑB	AB	AB	AB	AB	AB	AB
	003A06C8	98	07	3A	99	98	01	3A	00	EE	FΕ	EE	FE	EE	FE	EE	FE	EE	FE	EE	FE
	003A06EC	99	99	99	99	AB	AB	AB	AB	AB	AB	AB	AB	00	00	99	99	99	99	99	00

(4) Dword Shoot 攻击。

假设在执行该语句之前, h1 的 Flink 和 Blink 被改写为特定地址和特定数值,那么就完成一次 Dword Shoot 攻击。

四、心得体会:

实验涉及到了堆内存管理的一些基本概念和细节,通过手动操作堆块的申请、释放和指针修改,可以观察到堆内存管理的一些特性和可能出现的问题。

首先,实验中提到了创建大小为 0x1000 的堆区,并连续申请了 6 个 8 字节的堆块,这是为了形成一定的堆结构,并且观察后续的释放操作。

接着,通过释放奇数堆块,形成了三个16字节的空闲块,这些空闲块会被

加入到相应的空闲块链表中。这个步骤展示了堆内存管理中的空闲块合并机制,即相邻的空闲块可以合并成一个更大的空闲块,提高了内存利用率。

然后,在重新申请8字节的堆内存时,程序会从相应大小的空闲块链表中选择一个空闲块返回给程序使用,这展示了堆内存管理中的分配算法,通常是选择合适大小的空闲块以尽量减少内存碎片。

最后,通过手动修改堆块的指针,触发了DWORD SHOOT,这说明了堆内存管理的一个重要问题,即堆块的越界访问或者指针篡改可能导致程序的崩溃或者安全漏洞。

实验通过简单的堆内存管理操作,展示了堆内存的结构和管理方式,以及可能出现的问题,有助于理解堆内存管理机制和进行相关安全分析。