

CVE-2012-1889（暴雷漏洞）分析报告

启明星辰安全研究团队

近几天，炒的最火的莫过于 CVE-2012-1889（暴雷）漏洞了，目前该漏洞仍然为 0day 漏洞。下面是对该漏洞的一些分析。

漏洞分析：

IE8 运行利用代码，崩溃时如图所示：

5DD8D75A	FF75 28	push	dword ptr [ebp+28]
5DD8D75D	8B08	mov	ecx, dword ptr [eax]
5DD8D75F	FF75 24	push	dword ptr [ebp+24]
5DD8D762	FF75 20	push	dword ptr [ebp+20]
5DD8D765	57	push	edi
5DD8D766	6A 03	push	3
5DD8D768	FF75 14	push	dword ptr [ebp+14]
5DD8D76B	68 F8A7D85D	push	GUID_NULL
5DD8D770	53	push	ebx
5DD8D771	50	push	eax
5DD8D772	FF51 18	call	dword ptr [ecx+18]
5DD8D775	8945 0C	mov	dword ptr [ebp+C], eax
5DD8D778	8B06	mov	eax, dword ptr [esi]
5DD8D77A	56	push	esi
5DD8D77B	FF50 08	call	dword ptr [eax+8]
5DD8D77E	EB 79	jmp	short 5DD8D7F9
5DD8D780	8B 57000780	mov	eax, 80070057
5DD8D785	E9 8E000000	jmp	5DD8D818
5DD8D78A	8D4D 18	lea	ecx, dword ptr [ebp+18]

可见，ecx 的不可用导致了程序崩溃，我们向上看 ecx 是从哪里来的

.text:5DD8D75D mov ecx, [eax] ； 从 eax 这里来的，这里可以判断出
eax 被当作一个对象指针,ecx 为虚表

.text:5DD8D75F push [ebp+arg_1C]
.text:5DD8D762 push [ebp+arg_18]
.text:5DD8D765 push edi
.text:5DD8D766 push 3
.text:5DD8D768 push [ebp+arg_C]
.text:5DD8D76B push offset _GUID_NULL
.text:5DD8D770 push ebx
.text:5DD8D771 push eax
.text:5DD8D772 call dword ptr [ecx+18h]

再向上看 eax 是从哪里来的

.text:5DD8D751 mov eax, dword ptr [ebp+pvarg.anonymous_0+8]
.text:5DD8D754 cmp eax, ebx ； 这里做了一下比较，如果 eax 为 0 则
直接失败跳走，出错时肯定是 eax 不为 0，且没有被赋予一个正确的对象指针导致。
.text:5DD8D756 mov esi, eax
.text:5DD8D758 jz short loc_5DD8D780

如果 eax 为 0，会跳到这里


```
.text:5DD8D780          mov     eax, E_INVALIDARG
.text:5DD8D785          jmp     loc_5DD8D818 ; 函数返回失败并提示无效的参
数
```

继续调试之前，这里需要先提一下漏洞的利用代码，经过调试，在漏洞的利用代码中，最关键的一句是调用了这个属性 `obj.definition(0)`；

通过 MSDN 得知，definition 是一个属性，只读的，是不能传参的。

definition*

Returns the definition of the node in the document type definition (DTD) or schema. Read-only.



Script Syntax

```
var objXMLDOMNode = oXMLDOMNode.definition;
```

这里怎么给传了个参数呢，传了一个参数之后它又会怎么样呢？其实该漏洞就是因为调用该属性时非法传递了这么一个参数，才导致最终触发了漏洞。

现在从头开始研究这个漏洞。根据栈回溯，我们追踪到了这里。

5DDA6E3F	8BFF	mov	edi, edi	
5DDA6E41	55	push	ebp	
5DDA6E42	8BEC	mov	ebp, esp	
5DDA6E44	FF75 24	push	dword ptr [ebp+24]	
5DDA6E47	8B45 08	mov	eax, dword ptr [ebp+8]	
5DDA6E4A	FF75 20	push	dword ptr [ebp+20]	
5DDA6E4D	83C0 E8	add	eax, -18	
5DDA6E50	FF75 1C	push	dword ptr [ebp+1C]	
5DDA6E53	FF75 18	push	dword ptr [ebp+18]	argstruct
5DDA6E56	FF75 14	push	dword ptr [ebp+14]	
5DDA6E59	FF75 10	push	dword ptr [ebp+10]	
5DDA6E5C	FF75 0C	push	dword ptr [ebp+0C]	index
5DDA6E5F	6A 00	push	0	
5DDA6E61	68 84F5E15D	push	offset _dispatch<IXMLDOMNode,&L	StructA
5DDA6E66	50	push	eax	
5DDA6E67	E8 4466FEFF	call	_dispatchImpl::InvokeEx	
5DDA6E6C	5D	pop	ebp	
5DDA6E6D	C2 2000	ret	20	

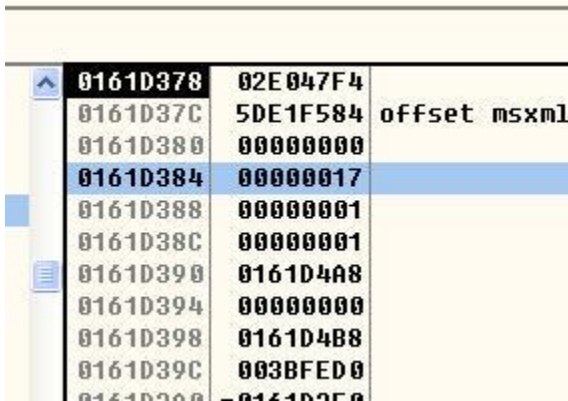
这里传进来几个参数，有些参数目前还不知道做什么的，这里只提几个和漏洞有关的参数。

参数 StructA 这是一个结构

参数 FunIndex 为需要找的属性对应的函数的索引值。在 msxml 中有这样结构的一些分发表（我们暂时称其为 dipatchtable）。

```
.data:5DE1F583          db      0
.data:5DE1F584 ; struct IUnknown * _dispatch_IXMLDOMNode__GUID const LIBID_MSXML2__GUID const IID_IXMLDOMNode__s_dispatchinfo
.data:5DE1F584 ?s_dispatchinfo@?$_dispatch@UIXMLDOMNode@@$?LIBID_MSXML2@3U_GUID@3B$?IID_IXMLDOMNode@3U3@B@2UDISPATCHINFO@@
.data:5DE1F584 ; DATA XREF: _dispatch<IXMLDOMNode,&GUID const LIBID_MSXML2,&GUID const
.data:5DE1F584 ; DOMNode::GetTypeInfoCount(uint *):loc_5DDA4B27to ...
.data:5DE1F588 dd offset _IID_IXMLDOMNode
.data:5DE1F588 ; DATA XREF: _dispatch<IXMLDOMNode,&GUID const LIBID_MSXML2,&GUID const
.data:5DE1F58C dd offset _LIBID_MSXML2
.data:5DE1F590 db      1
.data:5DE1F591 db      0
.data:5DE1F592 db      0
.data:5DE1F593 db      0
.data:5DE1F594 dd offset ?s_rgDOMNodeMethods@3PAUINVOKE_METHOD@3A ; INVOKE_METHOD * s_rgDOMNodeMethods
.data:5DE1F598 db      20h
.data:5DE1F599 db      0
.data:5DE1F59A db      0
.data:5DE1F59B db      0
.data:5DE1F59C dd offset ?s_DOMNode_DispatchMap@3PAUDISPIDTOINDEX@3A ; DISPIDTOINDEX * s_DOMNode_DispatchMap
.data:5DE1F5A0 db      20h
.data:5DE1F5A1 db      0
```


函数索引为 0x17，函数传进来的 FunIndex 也正为 0x17。

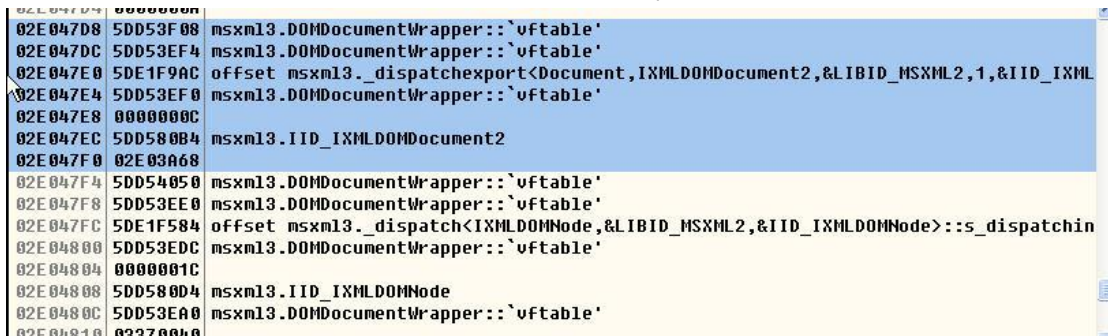


Address	Value	Comment
0161D378	02E047F4	
0161D37C	5DE1F584	offset msxml
0161D380	00000000	
0161D384	00000017	
0161D388	00000001	
0161D38C	00000001	
0161D390	0161D4A8	
0161D394	00000000	
0161D398	0161D4B8	
0161D39C	003BFED0	

继续说 dispatchtable 结构，偏 0x14 位置的数据为 Name 表的总个数。

后面 0x18 位置为 NameIndex 表。该索引使用刚才说的 FunIndex 计算出来 NameIndex，再利用 NameIndex 在 NameMap 中查找到对应的结构。比如刚才传入的 FunIndex 为 0x17，通过计算在 NameMap 中的索引为 0x6。对应的 NameMap 表中则为 definition 这一项。最后 0x1C 位置的为 NameIndex 表包含的 NameIndex 的个数，当然和 Name 表包含的 Name 个数是一样的。

接着说 StructDisp 这个结构，由于不知道这个结构的具体含义，因此暂时叫做 StructDisp。该结构大小为 0x1C。且一般存在两个类似的 StructDisp 结构，位置紧邻。



Address	Value	Comment
02E047D8	5DD53F08	msxml3.DOMDocumentWrapper::`vftable'
02E047DC	5DD53EF4	msxml3.DOMDocumentWrapper::`vftable'
02E047E0	5DE1F9AC	offset msxml3._dispatchexport<Document,IXMLDOMDocument2,&LIBID_MSXML2,1,&IID_IXML
02E047E4	5DD53EF0	msxml3.DOMDocumentWrapper::`vftable'
02E047E8	0000000C	
02E047EC	5DD580B4	msxml3.IID_IXMLDOMDocument2
02E047F0	02E03A68	
02E047F4	5DD54050	msxml3.DOMDocumentWrapper::`vftable'
02E047F8	5DD53EE0	msxml3.DOMDocumentWrapper::`vftable'
02E047FC	5DE1F584	offset msxml3._dispatch<IXMLDOMNode,&LIBID_MSXML2,&IID_IXMLDOMNode>::s_dispatchin
02E04800	5DD53EDC	msxml3.DOMDocumentWrapper::`vftable'
02E04804	0000001C	
02E04808	5DD580D4	msxml3.IID_IXMLDOMNode
02E0480C	5DD53EA0	msxml3.DOMDocumentWrapper::`vftable'

图中阴影部分为 StructDisp 结构的截图，另一个紧邻着它，大小也为 0x1C。上面提到了两个分发表，这两个分发表负责分发的属性/成员函数不同。对应两个分发表存在了两个 StructDisp 结构。为了区分两个 StructDisp，我们把前面的叫做 StructDispA，后面的叫做 StructDispB。

argstruct 为传进来的参数所组织成的一个结构。

```
Struct argstruct
{
    偏移 0 pStructB (指向另一个结构，在该结构中有传进来的参数具体的值)
    偏移 8 参数个数
}
```

在 StructB 中包含参数具体的值。

下面跟进上面那个函数：

```
.text:5DD8D4B0      mov     edi, edi
.text:5DD8D4B2      push   ebp
```

```

.text:5DD8D4B3      mov     ebp, esp
.text:5DD8D4B5      sub     esp, 24h
.text:5DD8D4B8      push    ebx
.text:5DD8D4B9      call    ?g_pfnEntry@@@3P6GPAUTLSDATA@@@XZA ; TLSDATA
* (*g_pfnEntry)(void)
.text:5DD8D4BF      xor     ebx, ebx
.text:5DD8D4C1      cmp     eax, ebx
.text:5DD8D4C3      mov     [ebp+var_4], eax
.text:5DD8D4C6      jnz     short loc_5DD8D4D9 ; 这里具体在做什么不大清楚, 但可以肯定的是, 如果 eax 为 0 的话则表示失败, 不为 0 则跳走继续下面的流程, 在接下来的调试中, 总是成功的
.text:5DD8D4C8      push    ebx
.text:5DD8D4C9      call    ?g_pfnExit@@@3P6GXPAUTLSDATA@@@ZA ; void
(*g_pfnExit)(TLSDATA *)
.text:5DD8D4CF      mov     eax, E_FAIL
.text:5DD8D4D4      jmp     loc_5DD8D5B9

```

做一些参数检查

```

.text:5DD8D4EB      test    byte ptr [ebp+arg_14+1], 40h
.text:5DD8D4EF      jz      short loc_5DD8D4FB ; 调试中传进来的固定为 0
.text:5DD8D4F1      mov     eax, E_FAIL
.text:5DD8D4F6      jmp     failed

.text:5DD8D4FB      cmp     [ebp+arg_8], bl ; 调试中传进来的固定为 0
.text:5DD8D4FE      mov     eax, [ebp+argstruct] ; 这里 eax 被赋值为 argstruct
结构
.text:5DD8D501      mov     edi, [ebp+FunIndex] ; edi 为 FunIndex
.text:5DD8D504      jz      short loc_5DD8D54F ; 这里跳

.text:5DD8D54F      mov     esi, [eax+0Ch] ; argstruct 结构偏移 0xC
.text:5DD8D552      cmp     esi, ebx
.text:5DD8D554      jz      short loc_5DD8D58A ; 调试中必跳
跳到了这个地方
.text:5DD8D58A      mov     ecx, [ebp+ StructDispB]; StructDisp
.text:5DD8D58D      mov     edx, [ecx] ; 取出 vtable
.text:5DD8D58F      lea     esi, [ebp+arg_8]
.text:5DD8D592      push    esi
.text:5DD8D593      push    [ebp+arg_20]
.text:5DD8D596      push    [ebp+pvarg]
.text:5DD8D599      push    eax ; argstruct
.text:5DD8D59A      push    [ebp+arg_14] ; 传进来的是 1
.text:5DD8D59D      push    [ebp+arg_10] ; 传进来的是 1
.text:5DD8D5A0      push    offset _GUID_NULL
.text:5DD8D5A5      push    edi ; FunIndex

```

```
.text:5DD8D5A6          push    ecx          ; StructDispB
.text:5DD8D5A7          call    dword ptr [edx+18h] ; DOMDocumentWrapper::Invoke
```

跳到了这里，取 StructDispB 向上相邻的另一个 StructDisp 结构 StructDispA

```
.text:5DDAE81          sub     [esp+arg_0], 1Ch
.text:5DDAE86          jmp     ?Invoke@DOMDocumentWrapp
```

进入函数 DOMDocumentWrapper::Invoke

```
.text:5DDACA94          mov     edi, edi
.text:5DDACA96          push    ebp
.text:5DDACA97          mov     ebp, esp
.text:5DDACA99          push    edi
.text:5DDACA9A          call    ?g_pfnEntry@@@3P6GPAUTLSDATA@@@XZA ; TLSDATA
* (*g_pfnEntry)(void)
.text:5DDACAA0          mov     edi, eax
.text:5DDACAA2          test    edi, edi
.text:5DDACAA4          jnz     short loc_5DDACAB4; 和上面一样，成功，跳走
.text:5DDACAA6          push    eax
.text:5DDACAA7          call    ?g_pfnExit@@@3P6GXAUTLSDATA@@@ZA ; void
(*g_pfnExit)(TLSDATA *)
.text:5DDACAAD          mov     eax, E_FAIL
.text:5DDACAB2          jmp     short loc_5DDACB15
```

检查 FunIndex 的值

```
.text:5DDACAB4          mov     eax, [ebp+FunIndex]
.text:5DDACAB7          cmp     eax, 1          ; 比较 FunIndex，这里
```

FunIndex 为 0x17，两处都不跳

```
.text:5DDACABA          push    esi
.text:5DDACABB          jle     short loc_5DDACAE6
.text:5DDACABD          cmp     eax, 24h
.text:5DDACAC0          jge     short loc_5DDACAE6
```

跳到这里

```
.text:5DDACAC2          push    [ebp+structAll] ; 包含几乎所有信息的一个对象
.text:5DDACAC5          push    [ebp+arg_1C]
.text:5DDACAC8          push    [ebp+arg_18]
.text:5DDACACB          push    [ebp+argstruct] ; argstruct
.text:5DDACACE          push    [ebp+arg_10]
.text:5DDACAD1          push    [ebp+arg_C]
.text:5DDACAD4          push    [ebp+guid]      ; GUID
.text:5DDACAD7          push    eax             ; FunIndex
.text:5DDACAD8          mov     eax, [ebp+StructDispA] ; 传进来的是上方相邻
的 StructDispA
.text:5DDACADB          add     eax, 1Ch        ; 又加回去了，变回 StructDipB
.text:5DDACADE          push    eax
.text:5DDACADF
```



```
call    ?Invoke@DOMNode@@@UAGJJABU_GUID@@@KGPAUtagDISPPARAMS@@@PAUtagVARIANT@@@PAUtagEXCEPINFO@@@PAI@Z
```

这里看看 structAll，这个结构包含了我们看的第一个函数传进来的很多信息，比如

0161D380	00000000
0161D384	00000017
0161D388	00000001
0161D38C	00000001
0161D390	0161D4A8
0161D394	00000000
0161D398	0161D4B8
0161D39C	003BFED0

index,argstruct 等等。

跟进 DOMNode::Invoke 函数，前面依然类似

```
.text:5DDA4CDA      mov     edi, edi
.text:5DDA4CDC      push   ebp
.text:5DDA4CDD      mov     ebp, esp
.text:5DDA4CDF      push   ecx
.text:5DDA4CE0      call   ?g_pfnEntry@@@3P6GPAUTLSDATA@@@XZA ; TLSDATA *
(*g_pfnEntry)(void)
.text:5DDA4CE6      test   eax, eax
.text:5DDA4CE8      mov     [ebp+var_4], eax
.text:5DDA4CEB      jnz     short loc_5DDA4CFE
.text:5DDA4CED      push   eax
.text:5DDA4CEE      call   ?g_pfnExit@@@3P6GXAUTLSDATA@@@@ZA ; void
(*g_pfnExit)(TLSDATA *)
.text:5DDA4CF4      mov     eax, E_FAIL
.text:5DDA4CF9      jmp     locret_5DDA4D94
检查 FunIndex
.text:5DDA4CFE      push   ebx
.text:5DDA4CFF      mov     ebx, [ebp+FunIndex]
.text:5DDA4D02      cmp     ebx, 5Dh
.text:5DDA4D05      push   esi
.text:5DDA4D06      push   edi
.text:5DDA4D07      mov     edi, DISP_E_MEMBERNOTFOUND
.text:5DDA4D0C      jle     short loc_5DDA4D61 ; FunIndex 为 0x17，这里跳
跳到这里
.text:5DDA4D61      push   [ebp+structAll] ; allobject
.text:5DDA4D64      push   [ebp+arg_1C] ; int
.text:5DDA4D67      push   [ebp+arg_18] ; int
.text:5DDA4D6A      push   [ebp+argstruct] ; object
.text:5DDA4D6D      push   [ebp+arg_10] ; int
.text:5DDA4D70      push   [ebp+arg_C] ; int
.text:5DDA4D73      push   [ebp+guid] ; int
.text:5DDA4D76      push   ebx ; FunIndex
```



```

.text:5DD8DAE4      push     edi
.text:5DD8DAE5      mov      edi, [ebp+argstruct] ; argstruct
.text:5DD8DAE8      jz       short loc_5DD8DB15; 不跳
.text:5DD8DAEA      cmp      [edi+0Ch], eax ;
.text:5DD8DAED      jz       short loc_5DD8DAF7 ; 为 0 跳
跳到这里
.text:5DD8DAF7      push     [ebp+structAll]
.text:5DD8DAFA      push     [ebp+arg_20]
.text:5DD8DAFD      push     [ebp+arg_1C] ; 0
.text:5DD8DB00      push     edi ; argstruct
.text:5DD8DB01      push     [ebp+arg_14] ; 1
.text:5DD8DB04      push     [ebp+arg_10] ; 1
.text:5DD8DB07      push     [ebp+FunIndex]
.text:5DD8DB0A      push     esi ; dispatchtable
.text:5DD8DB0B      push     [ebp+StructDispB]
.text:5DD8DB0E

```

call ?InvokeHelper@_dispatchImpl@@SGJPAXPAUDISPATCHINFO@@JKGPAUtagDISP
跟进_dispatchImpl::InvokeHelper

```

.text:5DD8D6BE      mov      edi, edi
.text:5DD8D6C0      push     ebp
.text:5DD8D6C1      mov      ebp, esp
.text:5DD8D6C3      sub      esp, 10Ch
.text:5DD8D6C9      push     ebx
.text:5DD8D6CA      push     esi
.text:5DD8D6CB      xor      ebx, ebx
.text:5DD8D6CD      cmp      [ebp+arg_1C], ebx
.text:5DD8D6D0      push     edi
.text:5DD8D6D1      push     ebx ; perrinfo
.text:5DD8D6D2      push     ebx ; dwReserved
.text:5DD8D6D3      setnz    [ebp+var_1]
.text:5DD8D6D7      mov      [ebp+var_8], ebx

```

前面做了一些检查，暂时不知道意图

```

.text:5DD8D6DA      call     __imp__SetErrorInfo@8 ; SetErrorInfo(x,x)
.text:5DD8D6E0      mov      esi, [ebp+dispatchtable] ; 获得 dispatchtable
.text:5DD8D6E3      lea      eax, [ebp+NameIndex] ;使用栈上的缓冲区用于接

```

收返回的 NameIndex

```

.text:5DD8D6E6      push     eax
.text:5DD8D6E7      movzx    eax, byte ptr [esi+1Ch] ; count of NameIndex
.text:5DD8D6EB      push     eax
.text:5DD8D6EC      push     dword ptr [esi+18h] ; NameIndexMap
.text:5DD8D6EF      push     [ebp+FunIndex]
.text:5DD8D6F2

```

call ?FindIndex@_dispatchImpl@@KGJPAUDISPIDTOINDEX@@HAAH@Z ; 通过传进来的
FunIndex 在 NameIndexMap 中查找其对应的 NameMap 中的索引，这里获得的索引为 6

5DD8D6E0	8B75 0C	mov	esi, dword ptr [ebp+C]
5DD8D6E3	8D45 F4	lea	eax, dword ptr [ebp-C]
5DD8D6E6	50	push	eax
5DD8D6E7	0FB646 1C	movzx	eax, byte ptr [esi+1C]
5DD8D6EB	50	push	eax
5DD8D6EC	FF76 18	push	dword ptr [esi+18]
5DD8D6EF	FF75 10	push	dword ptr [ebp+10]
5DD8D6F2	E8 69F9FFFF	call	_dispatchImpl::FindIndex
5DD8D6F7	3BC3	cmp	eax, ebx
5DD8D6F9	8945 0C	mov	dword ptr [ebp+C], eax
5DD8D6FC	0F8C 00010000	jl	5DD8D802
5DD8D702	8B45 F4	mov	eax, dword ptr [ebp-C]
5DD8D705	8B7D 1C	mov	edi, dword ptr [ebp+1C]
5DD8D708	395F 08	cmp	dword ptr [edi+8], ebx
5DD8D70B	8B4E 10	mov	ecx, dword ptr [esi+10]
ebx=00000000			
eax=00000000			
0161D258	00000006		

```
.text:5DD8D6F7          cmp     eax, ebx          ; 比较是否成功
.text:5DD8D6F9          mov     [ebp+retvalue], eax
.text:5DD8D6FC          jl      failed

.text:5DD8D702          mov     eax, [ebp+ NameIndex] ;取得返回的 NameIndex
.text:5DD8D705          mov     edi, [ebp+argstruct]
.text:5DD8D708          cmp     [edi+8], ebx      ; argstruct 偏移 8 为参数个数,
比较参数个数
.text:5DD8D70B          mov     ecx, [esi+10h]    ; NameMap
.text:5DD8D70E          lea     eax, [eax+eax*2]
.text:5DD8D711          lea     eax, [ecx+eax*8] ; 通过刚才获得的 NameIndex
再计算进而在 Name 表中查找出 definition 结构对应的偏移
.text:5DD8D714          jbe     short loc_5DD8D78A; 漏洞触发脚本中调用
definition 时传进了一个参数, 因此参数个数不为 0, 所以这里不跳
.text:5DD8D716          cmp     [ebp+FunIndex], ebx
.text:5DD8D719          jz      short loc_5DD8D78A ; 当然也不跳
.text:5DD8D71B          test    [ebp+arg_10], 1
.text:5DD8D71F          jz      short loc_5DD8D78A ; 不跳转
.text:5DD8D721          test    byte ptr [eax+16h], 2 ; 这里和下面的与 9 比较都
是查看找到的 definition 的一些属性
```

5DE1F638	5DD58464	UNICODE "definition"
5DE1F63C	00000017	
5DE1F640	00000000	
5DE1F644	00000000	
5DE1F648	00000000	
5DE1F64C	00020009	
5DE1F650	5DD5844C	UNICODE "firstChild"
5DE1F654	00000008	
5DE1F658	00000000	
5DE1F65C	00000000	
5DE1F660	00000000	
5DE1F664	00020009	

也就是图中 0x5DE1F64C 地址处的内容, 前面说了 Name 表中的每个 Name 自身都有一个结构, 在该结构的偏移 0x14 的一个 Word 和之后的偏移 0x16 的一个字节标明了调用该方法/

查询成员的一些属性。具体什么字段代表什么就不清楚了。我想可能和 MSDN 中描述的对应方法 / 成员的一些属性有关。

attributes	Contains the list of attributes for this node. Read-only.
baseName*	Returns the base name for the name qualified with the namespace. Read-only.
childNodes	Contains a node list containing the children nodes. Read-only.
dataType*	Specifies the data type for this node. Read/write.
definition*	Returns the definition of the node in the document type definition (DTD) or schema. Read-only.
firstChild	Contains the first child of the node. Read-only.
lastChild	Returns the last child node. Read-only.
name	Contains the attribute name. Read-only.
namespaceURI*	Returns the Uniform Resource Identifier (URI) for the namespace. Read-only.
nextSibling	Contains the next sibling of this node in the parent's child list. Read-only.
nodeName	Contains the qualified name of the element, attribute, or entity reference, or a fixed string for other node types. Read-only.
nodeType	Specifies the XML Document Object Model (DOM) node type, which determines valid values and whether the node can be modified. Read-only.
nodeValue*	Contains the node value expressed in its defined data type. Read/write.
nodeTypeString*	Returns the node type in string form. Read-only.
nodeValue	Contains the text associated with the node. Read/write.
ownerDocument	Returns the root of the document that contains the node. Read-only.

```
.text:5DD8D725          jz      short loc_5DD8D78A
.text:5DD8D727          cmp     word ptr [eax+14h], 9
.text:5DD8D72C          jnz     short loc_5DD8D78A
```

这里很明显 definition 的属性符合上面的条件，不跳转。

```
.text:5DD8D738          push    ebx
.text:5DD8D739          lea     eax, [ebp+pvarg] ; 这里用一个栈的缓冲区接收返回值
```

```
.text:5DD8D73C          push    eax
.text:5DD8D73D          push    2
.text:5DD8D73F          push    ebx
.text:5DD8D740          push    [ebp+FunIndex]
.text:5DD8D743          push    [ebp+StructDispB]
.text:5DD8D746          call    dword ptr [esi+20h] ; DOMNode::_invokeDOMNode
```

进入 DOMNode::_invokeDOMNode 函数

```
.text:5DDA3B71          mov     edi, edi
.text:5DDA3B73          push    ebp
.text:5DDA3B74          mov     ebp, esp
.text:5DDA3B76          mov     eax, [ebp+FunIndex] ; 通过 FunIndex (0x17) 查找需要跳转的处理函数分支
```

.text:5DDA3B79	add	eax, 0FFFFFFEh
.text:5DDA3B7C	cmp	eax, 21h ; switch 34 cases
.text:5DDA3B7F	push	esi
.text:5DDA3B80	push	edi
.text:5DDA3B81	ja	loc_5DDA3F37 ; jumtable 5DDA3B87 default
case		
.text:5DDA3B87	jmp	ds:off_5DDA3F42[ecx*4] ; switch jump

跳到这里

.text:5DDA3DAD	mov	edx, [ebp+retbuf] ; 获得刚传进来的返回值
buf		

.text:5DDA3DB0	mov	eax, [ebp+StructDispB]
.text:5DDA3DB3	mov	ecx, [eax] ; DOMDocumentWrapper::vtable
.text:5DDA3DB5	add	edx, 8 ;最后将传进来的 retbuf+8 作程序返回数据的地方

.text:5DDA3DB8	push	edx
.text:5DDA3DB9	push	eax
.text:5DDA3DBA	call	dword ptr [ecx+74h] ; get_definition
跟进 get_definition		

.text:5DDA5D31	push	30h
.text:5DDA5D33	push	offset stru_5DDA5E00
.text:5DDA5D38	call	__SEH_prolog
.text:5DDA5D3D	call	?g_pfnEntry@@@3P6GPAUTLSDATA@@@XZA ; TLSDATA
* (*g_pfnEntry)(void)		

.text:5DDA5D43	mov	[ebp+var_1C], eax
.text:5DDA5D46	xor	ebx, ebx
.text:5DDA5D48	cmp	eax, ebx
.text:5DDA5D4A	jnz	short loc_5DDA5D5D
.text:5DDA5D4C	push	ebx
.text:5DDA5D4D	call	?g_pfnExit@@@3P6GXAUTLSDATA@@@ZA ; void
(*g_pfnExit)(TLSDATA *)		

.text:5DDA5D53	mov	eax, E_FAIL
.text:5DDA5D58	jmp	loc_5DDA5DF7

前面类似，返回成功

.text:5DDA5D5D	mov	esi, [ebp+StructDispB]
.text:5DDA5D60	push	esi
.text:5DDA5D61	push	eax
.text:5DDA5D62	lea	ecx, [ebp+var_40]
.text:5DDA5D65	call	OMReadLock::OMReadLock(TLSDATA *, DOMNode *)
.text:5DDA5D6A	mov	edi, [ebp+retbuf]
.text:5DDA5D6D	cmp	edi, ebx
.text:5DDA5D6F	jnz	short loc_5DDA5D89 ; 查看 retbuf 是否存在，存
在，所以跳		
.text:5DDA5D71	lea	ecx, [ebp+var_40]

```
.text:5DDA5D74      call     OMReadLock::~~OMReadLock(void)
.text:5DDA5D79      push     [ebp+var_1C]
.text:5DDA5D7C      call     void (*g_pfnExit)(TlsData *)
.text:5DDA5D82      mov      eax, E_INVALIDARG
.text:5DDA5D87      jmp      short loc_5DDA5DF7
```

跳到这里

```
.text:5DDA5D89      mov      [ebp+ms_exc.disabled], ebx
.text:5DDA5D8C      mov      ecx, [esi+1Ch]
.text:5DDA5D8F      mov      [ebp+var_20], ecx ; NODE 对象
.text:5DDA5D92      mov      [ebp+var_24], ecx
.text:5DDA5D95      call     Node::getDefinition(void) ; 通过节点对象调用真正的 getDefinition
.text:5DDA5D9A      mov      ecx, eax ; 如果成功, eax 则不为 0
.text:5DDA5D9C      mov      [ebp+var_28], ecx
.text:5DDA5D9F      cmp      ecx, ebx
.text:5DDA5DA1      jz       short loc_5DDA5DAC ; 因为这里传进来一个非法的参数, 所以 getDefinition 必定不成功, 跳转实现
.text:5DDA5DAC      xor      ebx, ebx
.text:5DDA5DAE      inc      ebx
.text:5DDA5DAF      mov      [ebp+var_2C], ebx
.text:5DDA5DB2      jmp      short loc_5DDA5DE0
.text:5DDA5DE0      or       [ebp+ms_exc.disabled], 0FFFFFFFFh
.text:5DDA5DE4      lea      ecx, [ebp+var_40]
.text:5DDA5DE7      call     OMReadLock::~~OMReadLock(void)
.text:5DDA5DEC      push     [ebp+var_1C]
.text:5DDA5DEF      call     void (*g_pfnExit)(TlsData *)
.text:5DDA5DF5      mov      eax, ebx ; 最终 eax 返回 1
.text:5DDA5DF7      call     __SEH_epilog
.text:5DDA5DFC      retn     8
```

注意：在这种情况下，没有对返回值的 buf 进行任何赋值，因此刚才传进来的栈上的缓冲区内的值未变

5DD8D73F	53	push	ebx		
5DD8D740	FF75 10	push	dword ptr [ebp+10]		
5DD8D743	FF75 08	push	dword ptr [ebp+8]		
5DD8D746	FF56 20	call	dword ptr [esi+20]		
5DD8D749	3BC3	cmp	eax, ebx		
5DD8D74B	0F8C C7000000	jle	5DD8D818		
5DD8D751	8B45 EC	mov	eax, dword ptr [ebp+14]		
5DD8D754	3BC3	cmp	eax, ebx		
5DD8D756	8BF0	mov	esi, eax		
5DD8D758	74 26	je	short 5DD8D780		
5DD8D75A	FF75 28	push	dword ptr [ebp+28]		
5DD8D75D	8B08	mov	ecx, dword ptr [eax]		
5DD8D75F	FF75 24	push	dword ptr [ebp+24]		
5DD8D762	FF75 20	push	dword ptr [ebp+20]		
5DD8D765	57	push	edi		
5DD8D766	6A 03	push	3		
5DD8D768	FF75 14	push	dword ptr [ebp+14]		
5DD8D76B	68 F8A7D85D	push	GUID_NULL		
5DD8D770	53	push	ebx		
5DD8D771	50	push	eax		
5DD8D772	FF51 18	call	dword ptr [ecx+18]		
5DD8D775	8945 0C	mov	dword ptr [ebp+C], eax		

寄存器 (FPU)

EAX	00000001
ECX	5DDA5DFC nsxml3.5DDA
EDX	00000001
EBX	00000000
ESP	0161D14C
EBP	0161D264
ESI	5DE1F584 offset nsxm
EDI	0161D4A8
EIP	5DD8D749 nsxml3.5DD8
C 0	ES 0023 32位 0(FFFF
P 0	CS 001B 32位 0(FFFF
A 0	SS 0023 32位 0(FFFF
Z 0	DS 0023 32位 0(FFFF
S 0	FS 003B 32位 7FFD70
T 0	GS 0000 NULL
D 0	
0 0	LastErr ERROR_SUCCE
EFL	00000202 (NO,NB,NE,A
ST0	emotu -??? FFFF 0000

下面检查失败与否，这里当 eax 返回小于 0 的情况下表示失败。这里 eax 返回 1，程序继续。（程序在检查参数等出现错误的时候会将 eax 返回小于 0 的值，这里检查成功，只是 get_definition 失败）

.text:5DD8D758	jz	short	loc_5DD8D780
----------------	----	-------	--------------

之后便将其当作对象的 `this` 指针了，获得虚表，当然是错误的了

5DD0D72E	8045 E4	lea	eax, dword ptr [ebp-1C]			EAX	0161D270	
5DD0D731	50	push	eax	OLEAUT32.VariantInit		ECX	50D0A5DFC	msxml3.50D0A5DFC
5DD0D732	FF15 9890E15D	call	dword ptr [__imp__VariantInit]			EDX	00000001	
5DD0D738	53	push	ebx			EBX	00000000	
5DD0D739	8045 E4	lea	eax, dword ptr [ebp-1C]			ESP	0161D148	
5DD0D73C	50	push	eax			EBP	0161D264	
5DD0D73D	6A 02	push	2			ESI	0161D270	
5DD0D73F	53	push	ebx			EDI	0161D4A8	
5DD0D740	FF75 10	push	dword ptr [ebp+10]			EIP	50D0D75D	msxml3.50D0D75D
5DD0D743	FF75 08	push	dword ptr [ebp+8]			C 0	ES 0023 32	0(FFFFFFFF)
5DD0D746	FF56 20	call	dword ptr [esi+20]			P 0	CS 001B 32	0(FFFFFFFF)
5DD0D749	3BC3	cmp	eax, ebx			A 0	SS 0023 32	0(FFFFFFFF)
5DD0D74B	0F8C C7000000	j1	50D0D818			Z 0	DS 0023 32	0(FFFFFFFF)
5DD0D751	8B45 EC	mov	eax, dword ptr [ebp-14]			S 0	FS 003B 32	7FDF0000(FFF
5DD0D754	3BC3	cmp	eax, ebx			T 0	GS 0000 NULL	
5DD0D756	80F0	mov	esi, eax			D 0		
5DD0D759	74 26	je	short 50D0D780			D 0	LastErr ERROR_SUCCESS (00	
5DD0D75A	FF75 28	push	dword ptr [ebp+28]			EFL	00000202	(NO, NB, NE, A, NS, PO
5DD0D75D	8B08	mov	ecx, dword ptr [eax]			S70	empty	~??? FFFF 000000C7 0
5DD0D75F	FF75 24	push	dword ptr [ebp+24]			S11	empty	~??? FFFF 00000000 0
5DD0D762	FF75 20	push	dword ptr [ebp+20]			S12	empty	~??? FFFF 00000000 0
5DD0D765	57	push	edi			S13	empty	791.9999822974205017
5DD0D766	6A 03	push	3			S14	empty	426.9999904558062554
5DD0D768	FF75 14	push	dword ptr [ebp+14]			S15	empty	7777.0000000000000000
5DD0D76B	68 F867D85D	GUITD_NULL				S16	empty	7777.0000000000000000
5DD0D770	53	push	ebx			S17	empty	7777.0000000000000000
5DD0D771	50	push	eax					3 2 1 0 E
5DD0D772	FF51 18	call	dword ptr [ecx+18]					

5DD8D75D	8B08	mov	ecx, dword ptr [eax]	
5DD8D75F	FF75 24	push	dword ptr [ebp+24]	
5DD8D762	FF75 20	push	dword ptr [ebp+20]	
5DD8D765	57	push	edi	
5DD8D766	6A 03	push	3	
5DD8D768	FF75 14	push	dword ptr [ebp+14]	
5DD8D76B	68 F8A7D85D	push	GUID_NULL	
5DD8D770	53	push	ebx	
5DD8D771	50	push	eax	
5DD8D772	FF51 18	call	dword ptr [ecx+18]	
5DD8D775	8945 0C	mov	dword ptr [ebp+C], eax	
5DD8D778	8B06	mov	eax, dword ptr [esi]	
5DD8D77A	56	push	esi	
5DD8D77B	FF50 08	call	dword ptr [eax+8]	
5DD8D77E	EB 79	jmp	short 5DD8D7F9	
5DD8D780	B8 57000070	mov	eax, 80070057	
ds:[00000018]=???				
0161D270	00000000			
0161D274	00000017			
0161D278	00000001			
0161D27C	00000001			
0161D280	0161D4A8			
0161D284	00000000			
0161D288	0161D4B8			

以上分析表明当利用代码中在调用 definition 时非法传入了一个参数，导致走了错误的分支，最终触发了漏洞。在正常情况下，会进入另一个分支

5DD8D6F9	8945 0C	mov	dword ptr [ebp+C], eax	
5DD8D6FC	0F8C 00010000	jl	5DD8D802	
5DD8D702	8B45 F4	mov	eax, dword ptr [ebp-C]	
5DD8D705	8B7D 1C	mov	edi, dword ptr [ebp+1C]	
5DD8D708	395F 08	cmp	dword ptr [edi+8], ebx	参数个数检查
5DD8D70B	8B4E 10	mov	ecx, dword ptr [esi+10]	
5DD8D70E	8D0440	lea	eax, dword ptr [eax+eax*2]	
5DD8D711	8D04C1	lea	eax, dword ptr [ecx+eax*8]	
5DD8D714	76 74	jbe	short 5DD8D78A	没有参数传入，所以这里跳走，进入正常分支
5DD8D716	395D 10	cmp	dword ptr [ebp+10], ebx	
5DD8D719	74 6F	je	short 5DD8D78A	
5DD8D71B	F645 18 01	test	byte ptr [ebp+18], 1	
5DD8D71F	74 69	je	short 5DD8D78A	
5DD8D721	F640 16 02	test	byte ptr [eax+16], 2	
5DD8D725	74 63	je	short 5DD8D78A	
5DD8D727	66:8378 14 09	cmn	word ptr [eax+14], 9	

漏洞弥补：

这里仅提出本人想出的一种补丁修补办法，不一定正确，最后还是要看微软怎么修补的。在上文中提到以下这段代码是用来检查调用的方法的一些属性

.text:5DD8D721 test byte ptr [eax+16h], 2 ; 检查调用的方法的一些属性，并跳转到对应的处理函数，用于处理参数等

.text:5DD8D725 jz short loc_5DD8D78A

.text:5DD8D727 cmp word ptr [eax+14h], 9

.text:5DD8D72C jnz short loc_5DD8D78A

因此我找了其他几种和 get_definition 这个相同的一些方法（属性）。比如 get_firstchild, get_lastchild 这样的函数。

并对比了它们的代码结构和 get_definition 的异同，发现他们在失败的时候，会向返回值缓冲区写入 0。

如：get_firstchild 函数

```

ext:5DDA5AB8      mov     edi, [ebp+retnbuf]
ext:5DDA5ABB      cmp     edi, ebx          ; 检查存放返回值的缓冲区是否存在
ext:5DDA5ABD      jnz     short loc_5DDA5ACD
ext:5DDA5ABF      push    ecx
ext:5DDA5AC0      call   ?g_pfnExit@@3P6GXPATLSDATA@@@Z ; void (*g_pfnExit)(TSLDA
ext:5DDA5AC6      mov     eax, E_INVALIDARG
ext:5DDA5ACB      jmp     short loc_5DDA5B47
ext:5DDA5ACD      ; -----
ext:5DDA5ACD      loc_5DDA5ACD:          ; CODE XREF: DOMNode::get_firstChild(IXMLD
ext:5DDA5ACD      push    eax
ext:5DDA5ACE      push    ecx
ext:5DDA5ACF      lea     ecx, [ebp+var_3C]
ext:5DDA5AD2      call   ??0MReadLock@@QAE@PAUTLSDATA@@PAUDOMNode@@@Z ; OMReadLock
ext:5DDA5AD7      mov     [ebp+ms_exc.disabled], ebx
ext:5DDA5ADA      lea     eax, [ebp+var_24]
ext:5DDA5ADD      push    eax
ext:5DDA5ADE      mov     ecx, esi
ext:5DDA5AE0      call   ?getNodeFirstChild@Node@@QAEPAU1@PAPAX@Z ; Node::getNodeFi
ext:5DDA5AE5      mov     ecx, eax          ; 如果这里失败,eax则返回0
ext:5DDA5AE7      mov     [ebp+var_28], ecx
ext:5DDA5AEA      cmp     ecx, ebx          |
ext:5DDA5AEC      jz      short loc_5DDA5AF7 ; eax为0的情况下,跳转实现
ext:5DDA5AEE      call   ?getDOMNodeWrapper@Node@@QAEPAUDOMNode@@@XZ ; Node::getDOMN
ext:5DDA5AF3      mov     [edi], eax        ; 成功的情况,写入返回值的缓冲区内
ext:5DDA5AF5      jmp     short loc_5DDA5B2F
ext:5DDA5AF7      ; -----
ext:5DDA5AF7      loc_5DDA5AF7:          ; CODE XREF: DOMNode::get_firstChild(IXMLD
ext:5DDA5AF7      mov     [edi], ebx        ; 向存放返回值的缓冲区内写入0
ext:5DDA5AF9      mov     [ebp+var_1C], 1
ext:5DDA5B00      jmp     short loc_5DDA5B2F

```

get_lastchild 函数

```

B8F      mov     [ebp+var_1C], edx
B92      mov     eax, [ebp+arg_0]
B95      mov     esi, [eax+1Ch]
B98      mov     edi, [ebp+retnbuf]
B9B      cmp     edi, ebx          ; 检查返回值的buf
B9D      jnz     short loc_5DDA5BAD
B9F      push    ecx
BA0      call   ?g_pfnExit@@3P6GXPATLSDATA@@@Z ; void (*g_pfnExit)(
BA6      mov     eax, E_INVALIDARG
BAB      jmp     short loc_5DDA5C23
BAD      ; -----
BAD      loc_5DDA5BAD:          ; CODE XREF: DOMNode::get_lastChild(
BAD      push    eax
BAE      push    ecx
BAF      lea     ecx, [ebp+var_38]
BB2      call   ??0MReadLock@@QAE@PAUTLSDATA@@PAUDOMNode@@@Z ; OMRe
BB7      mov     [ebp+ms_exc.disabled], ebx
BBA      mov     ecx, esi
BBC      call   ?getNodeLastChild@Node@@QAEPAU1@XZ ; Node::getNodeLa
BC1      mov     ecx, eax
BC3      mov     [ebp+var_24], ecx
BC6      cmp     ecx, ebx
BC8      jz      short loc_5DDA5BD3 ; 失败则跳转
BCA      call   ?getDOMNodeWrapper@Node@@QAEPAUDOMNode@@@XZ ; Node::g
BCF      mov     [edi], eax
BD1      jmp     short loc_5DDA5C0B
BD3      ; -----
BD3      loc_5DDA5BD3:          ; CODE XREF: DOMNode::get_lastChild(
BD3      mov     [edi], ebx        ; 向返回值的buf中写入0
BD5      mov     [ebp+var_1C], 1

```

而 get_definition 则明显漏掉了这一步

```

.text:5DDA5D60      mov     ecx, [ebp+retbuf]
.text:5DDA5D6F      cmp     edi, ebx
.text:5DDA5D71      jnz     short loc_5DDA5D89 ; 查看retbuf是否存在
.text:5DDA5D74      lea     ecx, [ebp+var_40]
.text:5DDA5D79      call    ???OMReadLock@@QAE@XZ ; OMReadLock::~OMReadLock(void)
.text:5DDA5D7C      push    [ebp+var_1C]
.text:5DDA5D7E      call    ?g_pfnExit@@3P6GXPAUTLSDATA@@@Z ; void (*g_pfnExit)(TLSDATA *)
.text:5DDA5D82      mov     eax, E_INVALIDARG
.text:5DDA5D87      jmp     short loc_5DDA5DF7
.text:5DDA5D89      ; -----
.text:5DDA5D89      loc_5DDA5D89:      ; CODE XREF: DOMNode::get_definition(IXHLDOMNo
.text:5DDA5D89      mov     [ebp+ms_exc.disabled], ebx
.text:5DDA5D8C      mov     ecx, [esi+1Ch]
.text:5DDA5D8F      mov     [ebp+var_20], ecx ; NODE对象
.text:5DDA5D92      mov     [ebp+var_24], ecx
.text:5DDA5D95      call    ?getDefinition@Node@@QAEPAU1@XZ ; Node::getDefinition(void)
.text:5DDA5D9A      mov     ecx, eax ; 如果成功, eax则不为0
.text:5DDA5D9C      mov     [ebp+var_28], ecx
.text:5DDA5D9F      cmp     ecx, ebx
.text:5DDA5DA1      jz      short loc_5DDA5DAC ; 跳
.text:5DDA5DA3      call    ?getDOMNodeWrapper@Node@@QAEPAUDOMNode@@XZ ; Node::getDOMNodeW
.text:5DDA5DA8      mov     [edi], eax
.text:5DDA5DAA      jmp     short loc_5DDA5DE0
.text:5DDA5DAC      ; -----
.text:5DDA5DAC      loc_5DDA5DAC:      ; CODE XREF: DOMNode::get_definition(IXHLDOMNo
.text:5DDA5DAC      xor     ebx, ebx ; 漏掉了向返回值的buf写入0
.text:5DDA5DAE      inc     ebx
.text:5DDA5DAF      mov     [ebp+var_2C], ebx

```

因此可以在原 0x5DDA5DAC 这一处加上一句“mov [edi], ebx”达到修补漏洞的目的。这样在函数返回的时候，检查返回值是否为 0 则可以跳走，不再会将起当作一对象指针而执行之后的代码。

当然这样修补可以避免出现漏洞，但对于 definition 这个属性还不是完全了解，是否会出现其他问题还有待考量，仅仅是本人的一点拙见而已。

漏洞利用：

可以在栈上事先布置好数据，如 0C0C0C0C 之类的地址（为了控制之后从栈上取得的返回值），然后调用 definition 属性触发漏洞，再结合 Heap Spray 等方法在 0C0C0C0C 布置好 shellcode。网上已经有大牛给出了方法，这里就不班门弄斧了。

总结：

导致该漏洞的直接原因是，脚本调用 definition 属性时错误的传入了一个参数，导致处理函数内部走错了分支；根本原因是 get_definition 函数在失败的时候未向 retbuf 中写入数据，从而造成了漏洞。