CVE-2016-0147: take a look at patched analyzing.

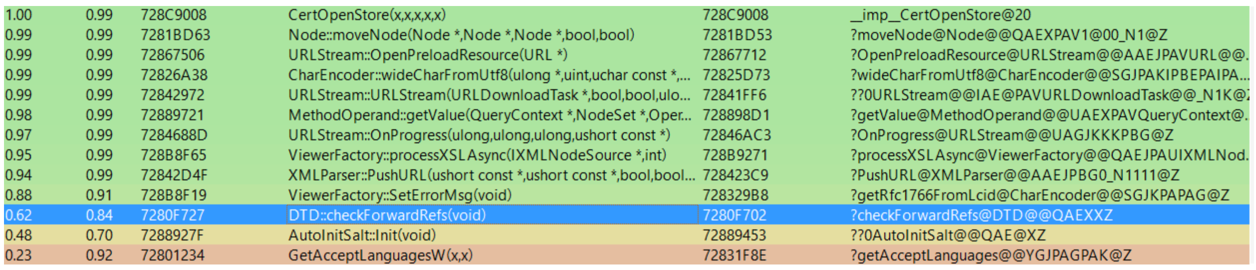
BY nEINEI

On April 12 in 2016, MS has released MS-16-40 bulletin, and mentioned the CVE-2016-147 vulnerability. Which about use-after-free vulnerability of MSXML3, and mentioned the CVE-2016-0147 vulnerability, which is about an UAF vulnerability of MSXML3.The patch caught our attention.

Root cause analysis : According to the MS MS-16-40 bulletin, a use-after-free vulnerability was found in Microsoft Windows that could lead to remote code execution. The issue was found in MSXML3 library. And can be exploited if the XML document parsed by the library is controlled by an attacker.

1) We stared that binary diffing analysis of windows 7 sp1 (unpatched MSXML3.DLL) File version 8.110.7601.18923 VS 8.110.7601.23373 (patched).

Between the 2 versions, we could see that there were very few changes rather than adding or removing functions.

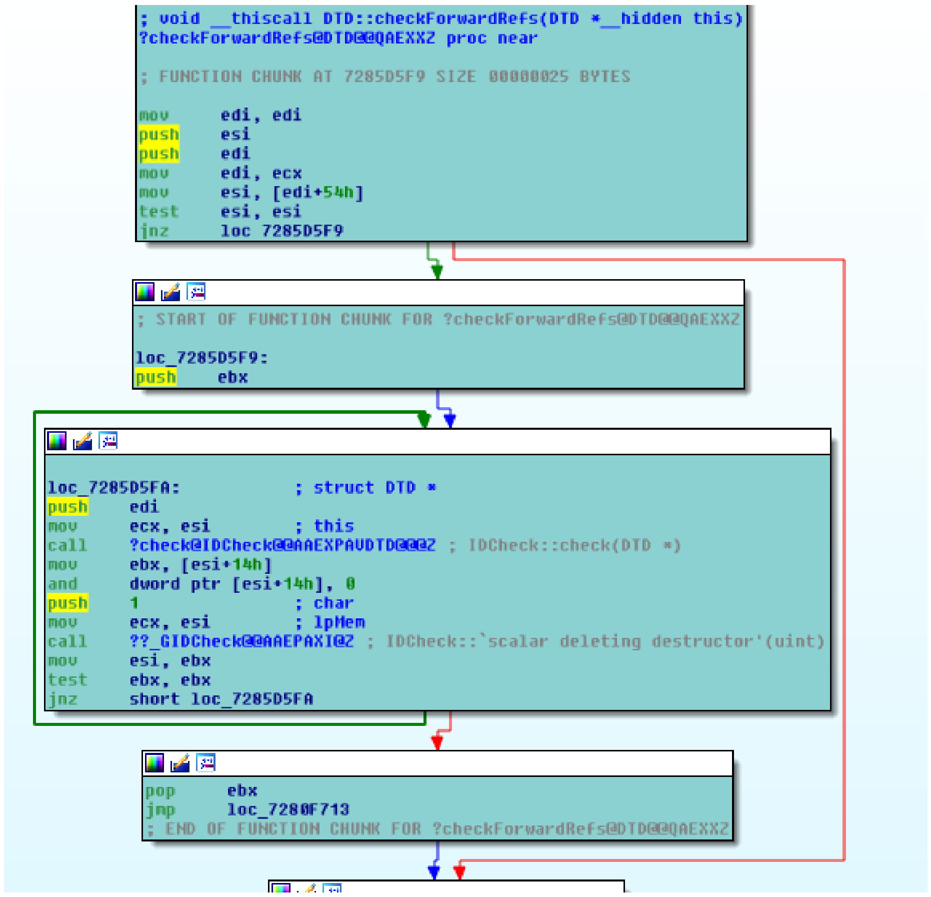


1.00	0.99	728C9008	CertOpenStore(x,x,x,x,x)	728C9008	__imp__CertOpenStore@20
0.99	0.99	7281BD63	Node::moveNode(Node *,Node *,Node *,bool,bool)	7281BD53	?moveNode@Node@@QAEXPAV1@00_N1@Z
0.99	0.99	72867506	URLStream::OpenPreloadResource(URL *)	72867712	?OpenPreloadResource@URLStream@@AAEJPAVURL@@.
0.99	0.99	72826A38	CharEncoder::wideCharFromUtf8(ulong *,uint,uchar const *,...	72825D73	?wideCharFromUtf8@CharEncoder@@SGJPAKIPBEPAIPA...
0.99	0.99	72842972	URLStream::URLStream(URLDownloadTask *,bool,bool,ulo...	72841FF6	??0URLStream@@IAE@PAVURLDownloadTask@@_N1K@Z
0.98	0.99	72889721	MethodOperand::getValue(QueryContext *,NodeSet *,Oper...	728898D1	?getValue@MethodOperand@@UAEXPAVQueryContext@.
0.97	0.99	7284688D	URLStream::OnProgress(ulong,ulong,ulong,ushort const *)	72846AC3	?OnProgress@URLStream@@UAGJKKKPBG@Z
0.95	0.99	728B8F65	ViewerFactory::processXSLAsync(IXMLNodeSource *,int)	728B9271	?processXSLAsync@ViewerFactory@@QAEJPAUIXMLNod.
0.94	0.99	72842D4F	XMLParser::PushURL(ushort const *,ushort const *,bool,bool...	728423C9	?PushURL@XMLParser@@AAEJPBG0_N1111@Z
0.88	0.91	728B8F19	ViewerFactory::SetErrorMsg(void)	728329B8	?getRfc1766FromLcid@CharEncoder@@SGJKPAPAG@Z
0.62	0.84	7280F727	DTD::checkForwardRefs(void)	7280F702	?checkForwardRefs@DTD@@QAEXXZ
0.48	0.70	7288927F	AutoInitSalt::Init(void)	72889453	??0AutoInitSalt@@QAE@XZ
0.23	0.92	72801234	GetAcceptLanguagesW(x,x)	72831F8E	?getAcceptLanguages@@YGJPAGPAK@Z

After analyzing of AutoInitStalt::Init and GetAceptLanguagesW , we find they don't request user input XML data, and they seem like without user input XML data. So, let us focus on the DTD::CheckForwardRefs function.

DTD::CheckForwardRefs references to DTD and IDCheck object. Obviously, the function aims to check some DTD object through IDCheck::check (). It will release the IDCheck Object until the IDCheck object no longer to reference any IDCheck object. Let us again simplify two object model.

```
Class DTD {…

    IDCheck *pIDCheckObject // offset 0x54 from the beginning of the DTD object.

    …}

Class IDCheck {…

IDCheck *pNextObject; // offset 0x14, point to the another IDCheck object.

…}
```



```
; void __thiscall DTD::checkForwardRefs(DTD *__hidden this)
?checkForwardRefs@DTD@@QAEXXZ proc near

; FUNCTION CHUNK AT 7285D5F9 SIZE 00000025 BYTES

mov     edi, edi
push    esi
push    edi
mov     edi, ecx
mov     esi, [edi+54h]
test    esi, esi
jnz     loc_7285D5F9
```

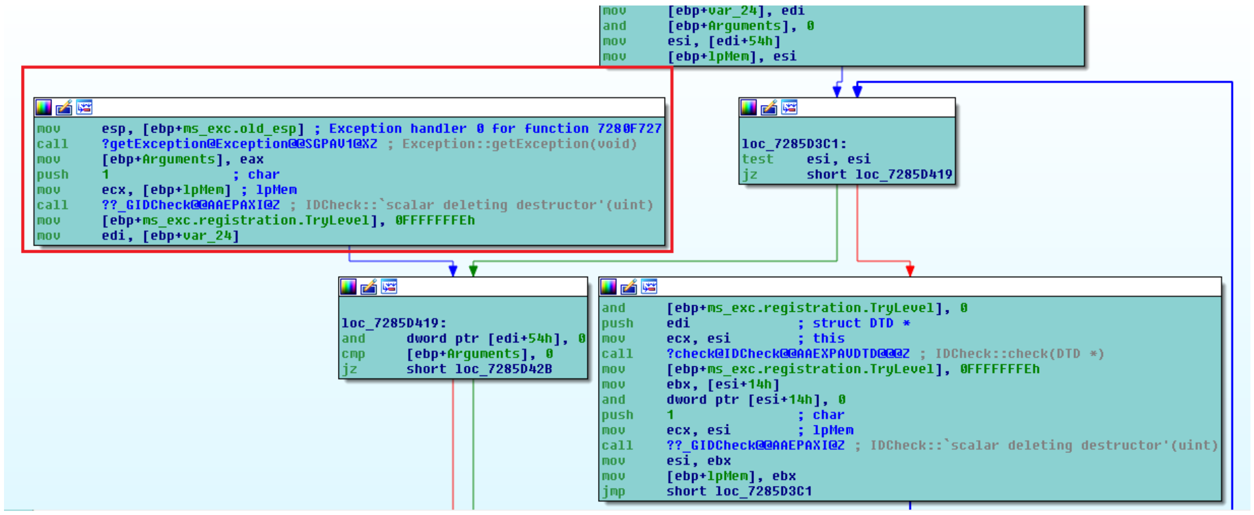
```
; START OF FUNCTION CHUNK FOR ?checkForwardRefs@DTD@@QAEXXZ

loc_7285D5F9:
push    ebx
```

```
loc_7285D5FA:             ; struct DTD *
push    edi
mov     ecx, esi          ; this
call    ?check@IDCheck@@AAEXPAVDTD@@@Z ; IDCheck::check(DTD *)
mov     ebx, [esi+14h]
and     dword ptr [esi+14h], 0
push    1                 ; char
mov     ecx, esi          ; lpMem
call    ??_GIDCheck@@AAEPAXI@Z ; IDCheck::`scalar deleting destructor'(uint)
mov     esi, ebx
test    ebx, ebx
jnz     short loc_7285D5FA
```

```
pop     ebx
jmp     loc_7280F713
; END OF FUNCTION CHUNK FOR ?checkForwardRefs@DTD@@QAEXXZ
```

If DTD::FindID find there is ID existed, it will release the IDCheck object and continue to get next IDCheck object cyclic process until the pNextObject point is null. If any no ID existing, function will raise a custom exception process.

After patched, we found that IDCheck::check only added exception handlers rather than other code. So, why fixed the vulnerability in this way?



```
mov     [ebp+var_24], edi
and     [ebp+Arguments], 0
mov     esi, [edi+54h]
mov     [ebp+lpMem], esi
```

```
mov     esp, [ebp+ms_exc.old_esp] ; Exception handler 0 for function 7280F727
call    ?getException@Exception@@SGPAV1@XZ ; Exception::getException(void)
mov     [ebp+Arguments], eax
push    1                 ; char
mov     ecx, [ebp+lpMem] ; lpMem
call    ??_GIDCheck@@AAEPAXI@Z ; IDCheck::`scalar deleting destructor'(uint)
mov     [ebp+ms_exc.registration.TryLevel], 0FFFFFFFEh
mov     edi, [ebp+var_24]
```

```
loc_7285D3C1:
test    esi, esi
jz      short loc_7285D419
```

```
loc_7285D419:
and     dword ptr [edi+54h], 0
cmp     [ebp+Arguments], 0
jz      short loc_7285D42B
```

```
and     [ebp+ms_exc.registration.TryLevel], 0
push    edi               ; struct DTD *
mov     ecx, esi          ; this
call    ?check@IDCheck@@AAEXPAVDTD@@@Z ; IDCheck::check(DTD *)
mov     [ebp+ms_exc.registration.TryLevel], 0FFFFFFFEh
mov     ebx, [esi+14h]
and     dword ptr [esi+14h], 0
push    1                 ; char
mov     ecx, esi          ; lpMem
call    ??_GIDCheck@@AAEPAXI@Z ; IDCheck::`scalar deleting destructor'(uint)
mov     esi, ebx
mov     [ebp+lpMem], ebx
jmp     short loc_7285D3C1
```

2)

Let's look at a simplified vulnerability happening in this scene, repeated loading can be included DTD of XML data.

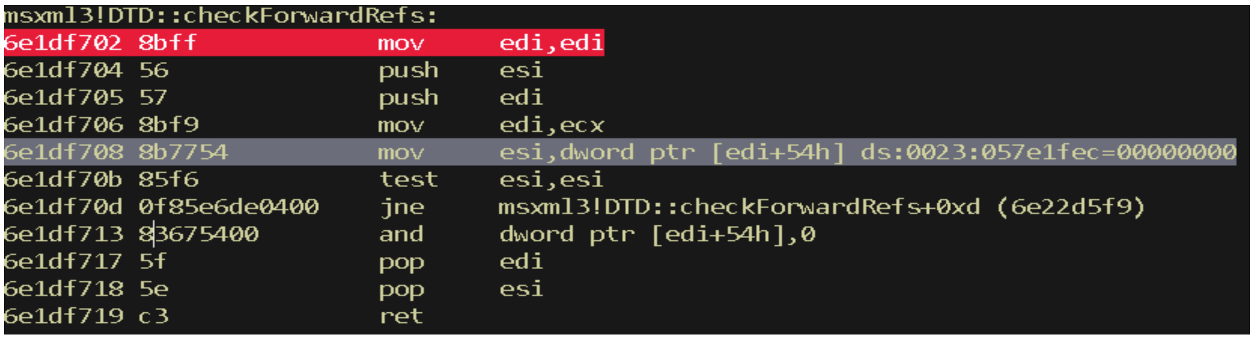
DTD of XML reference：http://xmlwriter.net/xml_guide/doctype_declaration.shtml

The critical functions calling order list like following:

msxml3!Document::loadXML –> msxml3!DTD::checkForwardRefs -> msxml3!DTD::addForwardRef -> msxml3!DTD::addForwardRef-> msxml3!Document::loadXML…

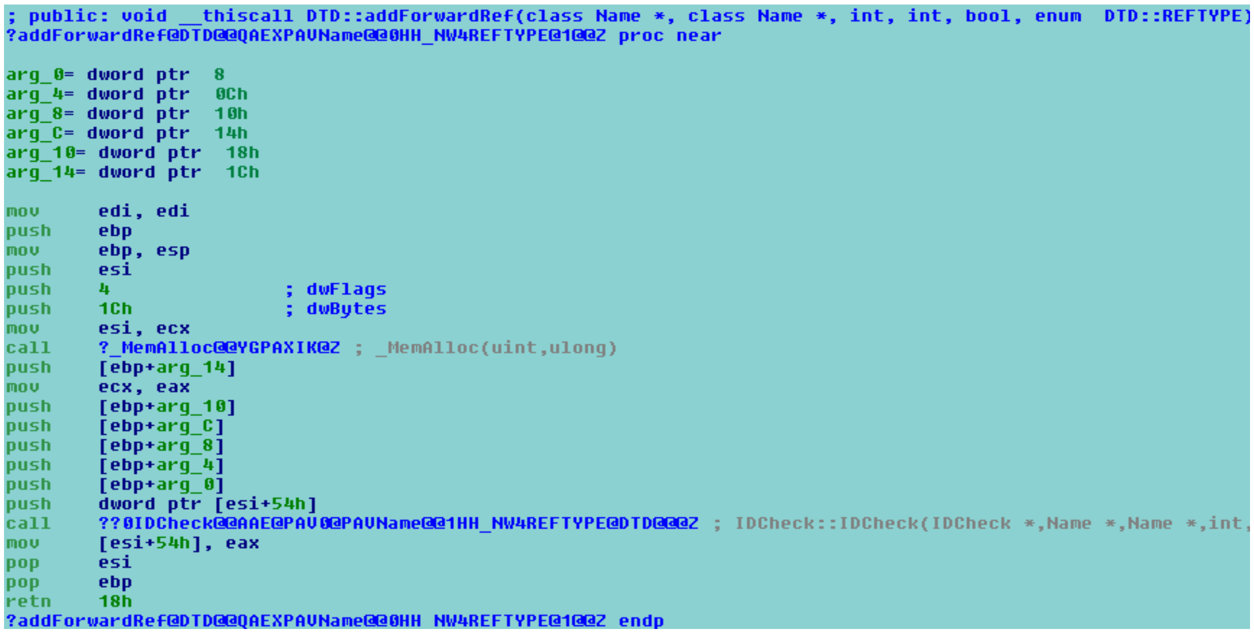
Frist, After the msxml3!DocumentLLloadXML called, it will create a new DTD object through DTD:New(struct Document *a1, struct DTD **a2).

If some object reference two different ID, the program will be called the DTD::CheckForwardRefs function.



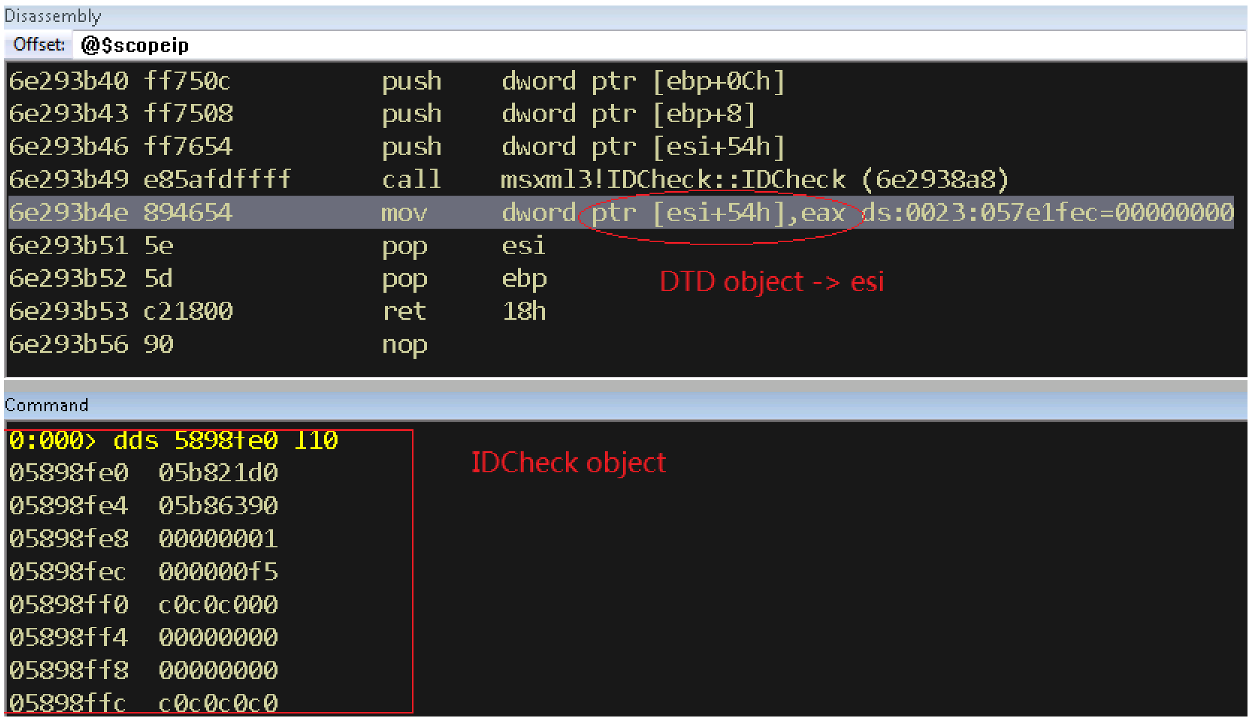
```
msxml3!DTD::checkForwardRefs:
6e1df702 8bff               mov     edi,edi
6e1df704 56                 push    esi
6e1df705 57                 push    edi
6e1df706 8bf9               mov     edi,ecx
6e1df708 8b7754             mov     esi,dword ptr [edi+54h] ds:0023:057e1fec=00000000
6e1df70b 85f6               test    esi,esi
6e1df70d 0f85e6de0400       jne     msxml3!DTD::checkForwardRefs+0xd (6e22d5f9)
6e1df713 83675400           and     dword ptr [edi+54h],0
6e1df717 5f                 pop     edi
6e1df718 5e                 pop     esi
6e1df719 c3                 ret
```

First time, the offset 0x54 of DTD object has no reference IDCheck object. It will continue to call the sxml3!DTD::addForwardRef and create a new IDCheck object.



```
; public: void __thiscall DTD::addForwardRef(class Name *, class Name *, int, int, bool, enum  DTD::REFTYPE)
?addForwardRef@DTD@@QAEXPAVName@@0HH_NW4REFTYPE@1@@Z proc near

arg_0= dword ptr  8
arg_4= dword ptr  0Ch
arg_8= dword ptr  10h
arg_C= dword ptr  14h
arg_10= dword ptr  18h
arg_14= dword ptr  1Ch

mov     edi, edi
push    ebp
mov     ebp, esp
push    esi
push    4               ; dwFlags
push    1Ch             ; dwBytes
mov     esi, ecx
call    ?_MemAlloc@@YGPAXIK@Z ; _MemAlloc(uint,ulong)
push    [ebp+arg_14]
mov     ecx, eax
push    [ebp+arg_10]
push    [ebp+arg_C]
push    [ebp+arg_8]
push    [ebp+arg_4]
push    [ebp+arg_0]
push    dword ptr [esi+54h]
call    ??0IDCheck@@AAE@PAV0@PAVName@@1HH_NW4REFTYPE@DTD@@@Z ; IDCheck::IDCheck(IDCheck *,Name *,Name *,int,
mov     [esi+54h], eax
pop     esi
pop     ebp
retn    18h
?addForwardRef@DTD@@QAEXPAVName@@0HH NW4REFTYPE@1@@Z endp
```

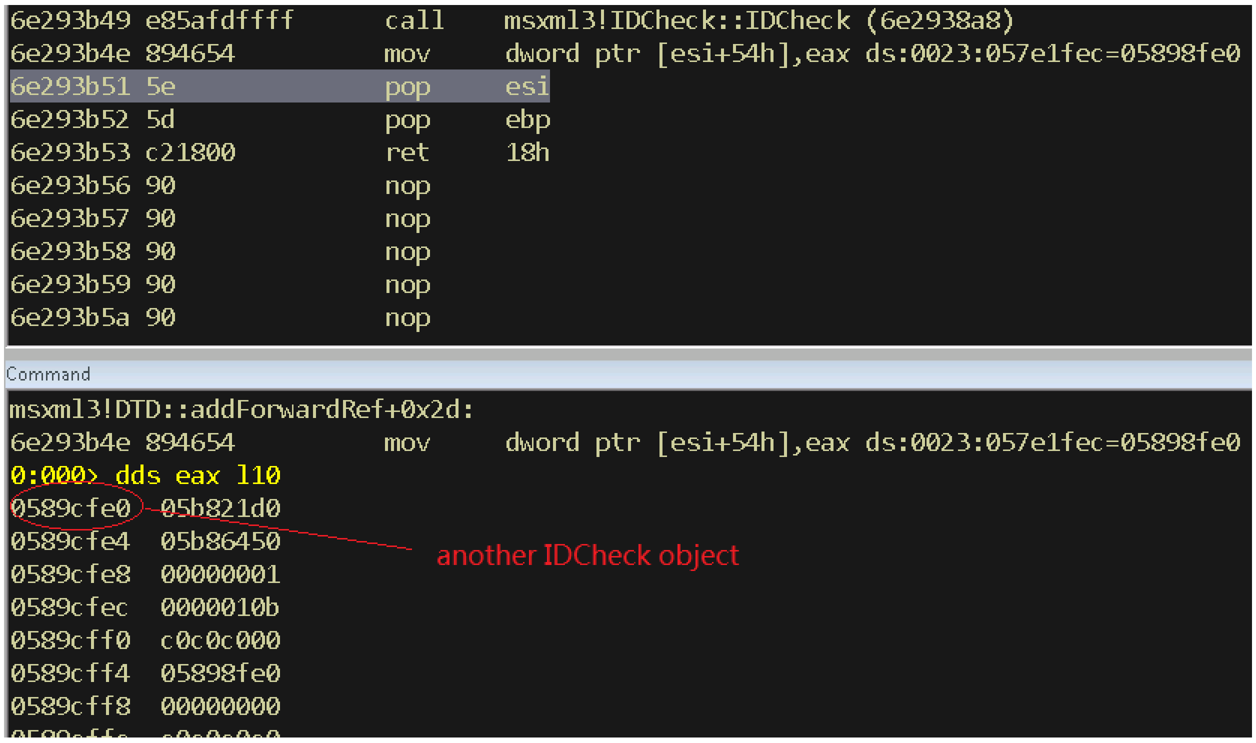
By _MemAlloc function build a new IDCheck object in the memory space 0x598fe0.



```
6e293b40 ff750c          push    dword ptr [ebp+0Ch]
6e293b43 ff7508          push    dword ptr [ebp+8]
6e293b46 ff7654          push    dword ptr [esi+54h]
6e293b49 e85afdffff      call    msxml3!IDCheck::IDCheck (6e2938a8)
6e293b4e 894654          mov     dword ptr [esi+54h],eax ds:0023:057e1fec=00000000
6e293b51 5e              pop     esi
6e293b52 5d              pop     ebp
6e293b53 c21800          ret     18h
6e293b56 90              nop
```

```
0:000> dds 5898fe0 l10
05898fe0  05b821d0
05898fe4  05b86390
05898fe8  00000001
05898fec  000000f5
05898ff0  c0c0c000
05898ff4  00000000
05898ff8  00000000
05898ffc  c0c0c0c0
```

Second time, allocation another IDCheck object in the memory space 0x589cfe0.

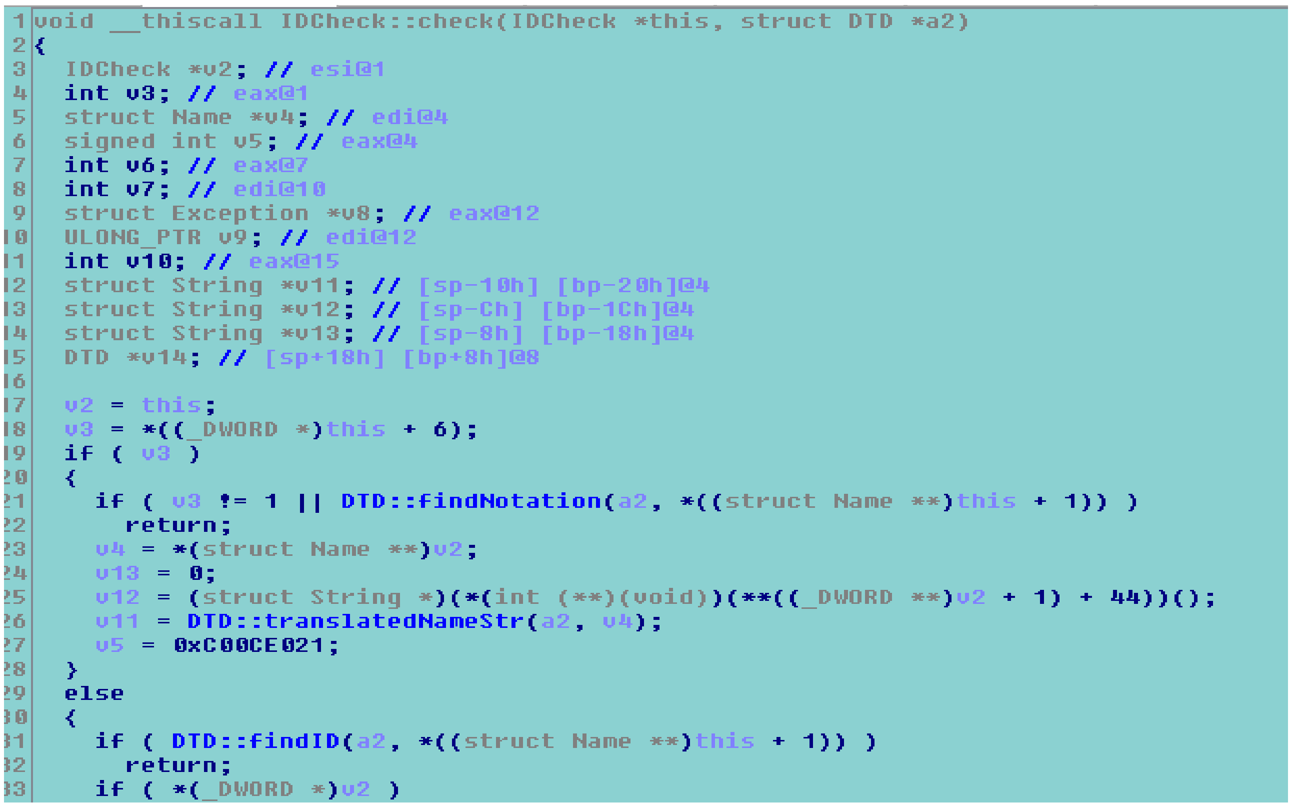


```
6e293b49 e85afdffff      call    msxml3!IDCheck::IDCheck (6e2938a8)
6e293b4e 894654          mov     dword ptr [esi+54h],eax ds:0023:057e1fec=05898fe0
6e293b51 5e              pop     esi
6e293b52 5d              pop     ebp
6e293b53 c21800          ret     18h
6e293b56 90              nop
6e293b57 90              nop
6e293b58 90              nop
6e293b59 90              nop
6e293b5a 90              nop
```

```
msxml3!DTD::addForwardRef+0x2d:
6e293b4e 894654          mov     dword ptr [esi+54h],eax ds:0023:057e1fec=05898fe0
0:000> dds eax l10
0589cfe0  05b821d0
0589cfe4  05b86450
0589cfe8  00000001
0589cfec  0000010b
0589cff0  c0c0c000
0589cff4  05898fe0
0589cff8  00000000
```

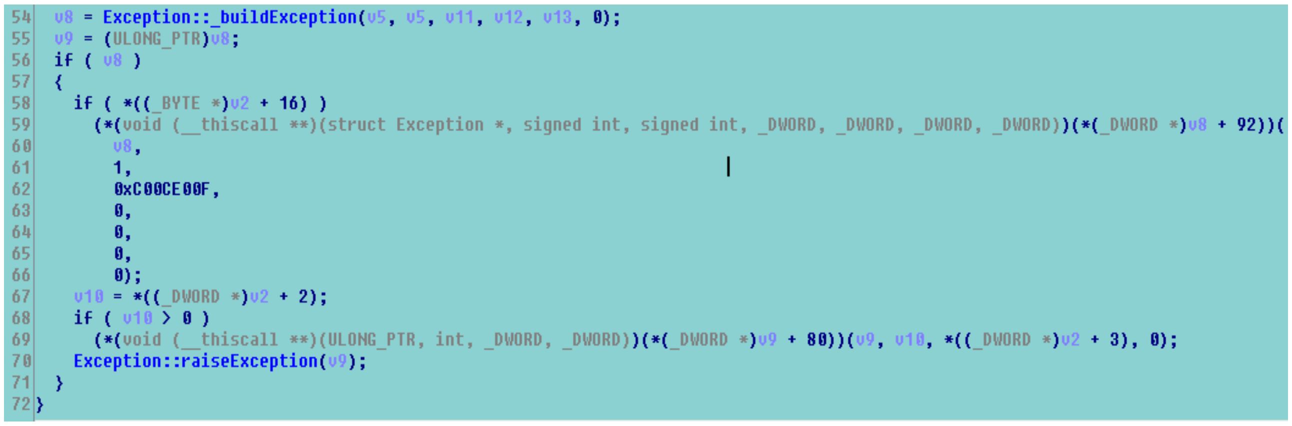
3)

Then, msxml3!DTD::CheckForwardRefs will reset this memory(IDCheck object) to be zero with its strategy accordingly(free up or set to be zero). Which IDCheck::check will be called when DTD::findNotation or DTD::findID will check the ID whether has already been check in a HASHTABLE of structure.



```
void __thiscall IDCheck::check(IDCheck *this, struct DTD *a2)
{
  IDCheck *v2; // esi@1
  int v3; // eax@1
  struct Name *v4; // edi@4
  signed int v5; // eax@4
  int v6; // eax@7
  int v7; // edi@10
  struct Exception *v8; // eax@12
  ULONG_PTR v9; // edi@12
  int v10; // eax@15
  struct String *v11; // [sp-10h] [bp-20h]@4
  struct String *v12; // [sp-Ch] [bp-1Ch]@4
  struct String *v13; // [sp-8h] [bp-18h]@4
  DTD *v14; // [sp+18h] [bp+8h]@8

  v2 = this;
  v3 = *((_DWORD *)this + 6);
  if ( v3 )
  {
    if ( v3 != 1 || DTD::findNotation(a2, *((struct Name **)this + 1)) )
      return;
    v4 = *(struct Name **)v2;
    v13 = 0;
    v12 = (struct String *)(*(int (**)(void))(**((_DWORD **)v2 + 1) + 44))();
    v11 = DTD::translatedNameStr(a2, v4);
    v5 = 0xC00CE021;
  }
  else
  {
    if ( DTD::findID(a2, *((struct Name **)this + 1)) )
      return;
    if ( *(_DWORD *)v2 )
```

If so, it will continue to execution, if not will raise a custom exception handlers. It will free some resources in exception handlers which include IDCheck object.



```
  v8 = Exception::_buildException(v5, v5, v11, v12, v13, 0);
  v9 = (ULONG_PTR)v8;
  if ( v8 )
  {
    if ( *((_BYTE *)v2 + 16) )
      (*(void (__thiscall **)(struct Exception *, signed int, signed int, _DWORD, _DWORD, _DWORD, _DWORD))(*(_DWORD *)v8 + 92))(
        v8,
        1,
        0xC00CE00F,
        0,
        0,
        0,
        0);
    v10 = *((_DWORD *)v2 + 2);
    if ( v10 > 0 )
      (*(void (__thiscall **)(ULONG_PTR, int, _DWORD, _DWORD))(*(_DWORD *)v9 + 80))(v9, v10, *((_DWORD *)v2 + 3), 0);
    Exception::raiseException(v9);
  }
}
```

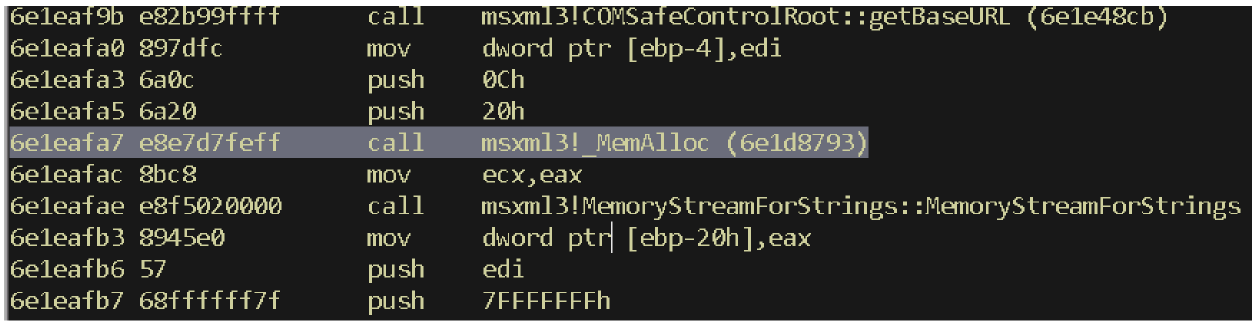
```
0:000> p

eax=00000000 ebx=00000000 ecx=0589cfe0 edx=0000000d esi=0589cfe0 edi=00000000
```

```
eip=6e1d2983 esp=0016e460 ebp=0016e468 iopl=0         nv up ei pl nz na po nc

cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202

msxml3!MemFreeObject+0x1d:

6e1d2983 e809000000      call    msxml3!MpHeapFree (6e1d2991)

0:000> kp

ChildEBP RetAddr

0016e468 6e293937 msxml3!MemFreeObject+0x1d

0016e478 6e22d625 msxml3!IDCheck::`scalar deleting destructor'+0x19

0016e488 6e1e4947 msxml3!DTD::clear+0x43

0016e494 6e205265 msxml3!Document::clear+0x2e

0016e4cc 6e205526 msxml3!Document::abort+0x95

0016e4dc 6e2054f9 msxml3!Document::HandleParseError+0x23

0016e514 6e1e45f5 msxml3!Document::HandleEndDocument+0xb9

0016e548 6e1e451b msxml3!Document::run+0xda

0016e584 6e1eafe5 msxml3!Document::_load+0x18e
```

4)

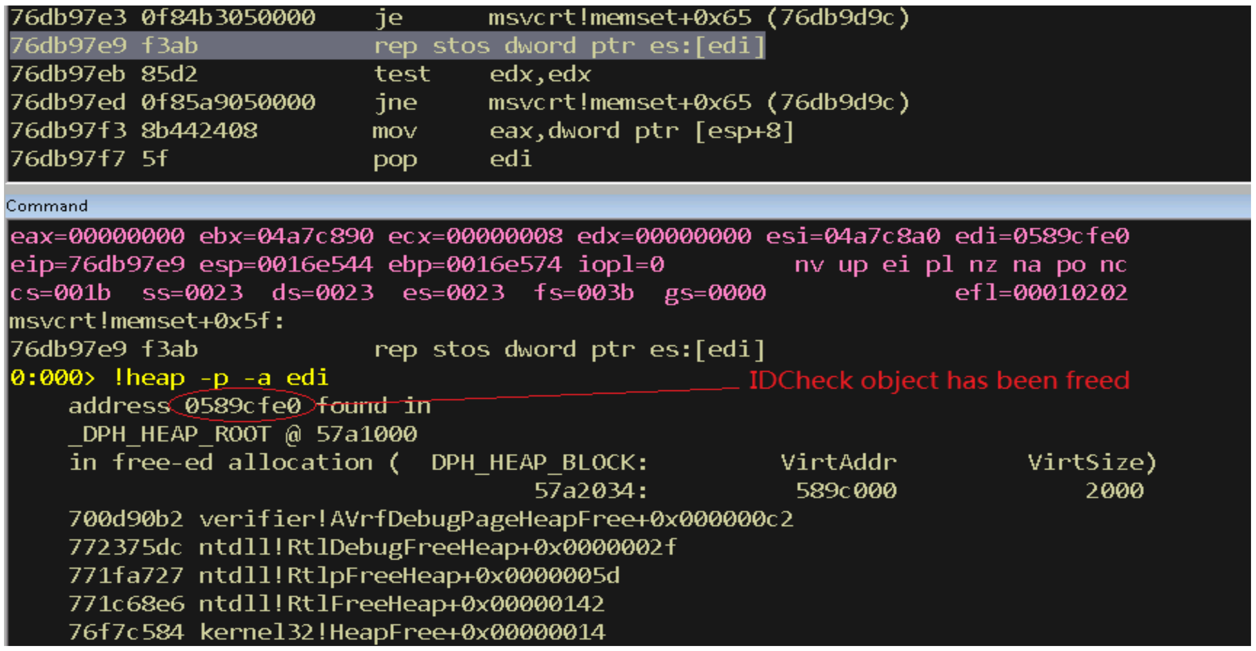
When prgram loaded a including DTD of XML data again, it will trigger a UAF . because msxml3!_MemAlloc will return the freed IDCheck object to user.

```
0:000> p

eax=0016e584 ebx=00000101 ecx=6e1e48f2 edx=00000000 esi=057c7ed8 edi=00000000

eip=6e1eafa7 esp=0016e594 ebp=0016e5d4 iopl=0         nv up ei pl nz na po nc

cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202

msxml3!Document::loadXML+0xa4:

6e1eafa7 e8e7d7feff      call    msxml3!_MemAlloc (6e1d8793)
```



```
6e1eaf9b e82b99ffff      call    msxml3!COMSafeControlRoot::getBaseURL (6e1e48cb)
6e1eafa0 897dfc          mov     dword ptr [ebp-4],edi
6e1eafa3 6a0c            push    0Ch
6e1eafa5 6a20            push    20h
6e1eafa7 e8e7d7feff      call    msxml3!_MemAlloc (6e1d8793)
6e1eafac 8bc8            mov     ecx,eax
6e1eafae e8f5020000      call    msxml3!MemoryStreamForStrings::MemoryStreamForStrings
6e1eafb3 8945e0          mov     dword ptr [ebp-20h],eax
6e1eafb6 57              push    edi
6e1eafb7 68ffffff7f      push    7FFFFFFFh
```

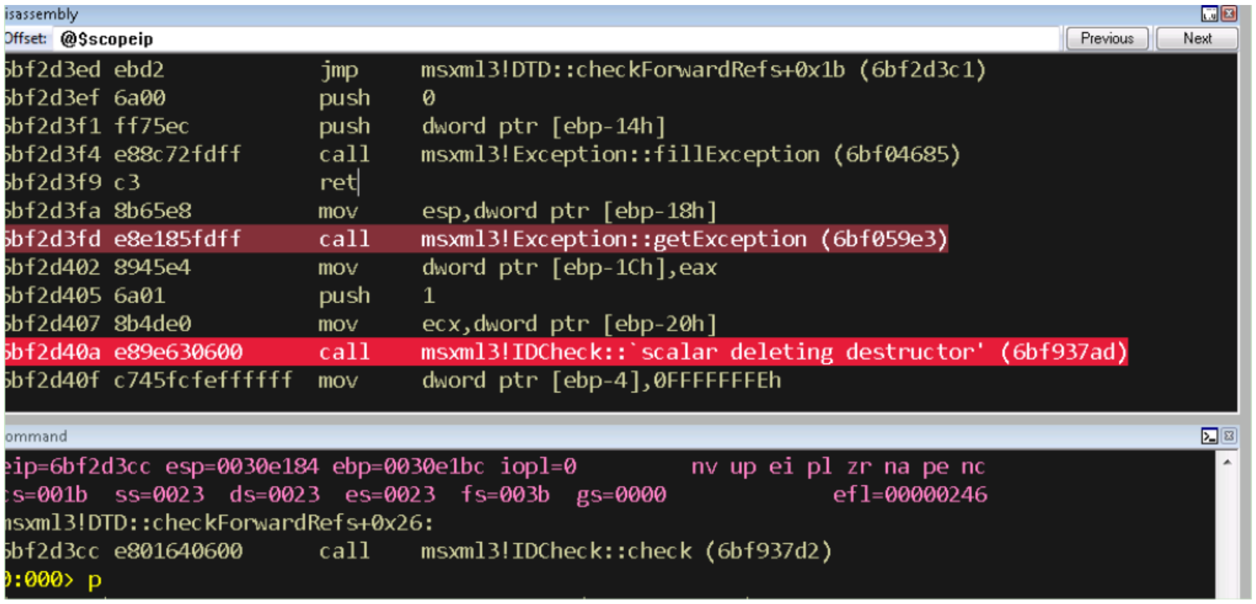
```
0:000> g

(5ac.95c): Access violation - code c0000005 (!!! second chance !!!)

eax=00000000 ebx=04a7c890 ecx=00000008 edx=00000000 esi=04a7c8a0 edi=0589cfe0

eip=76db97e9 esp=0016e544 ebp=0016e574 iopl=0         nv up ei pl nz na po nc

cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000         efl=00010202

msvcrt!memset+0x5f:

76db97e9 f3ab            rep stos dword ptr es:[edi]
```



```
76db97e3 0f84b3050000    je      msvcrt!memset+0x65 (76db9d9c)
76db97e9 f3ab            rep stos dword ptr es:[edi]
76db97eb 85d2            test    edx,edx
76db97ed 0f85a9050000    jne     msvcrt!memset+0x65 (76db9d9c)
76db97f3 8b442408        mov     eax,dword ptr [esp+8]
76db97f7 5f              pop     edi
```

```
Command
eax=00000000 ebx=04a7c890 ecx=00000008 edx=00000000 esi=04a7c8a0 edi=0589cfe0
eip=76db97e9 esp=0016e544 ebp=0016e574 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
msvcrt!memset+0x5f:
76db97e9 f3ab            rep stos dword ptr es:[edi]
0:000> !heap -p -a edi
    address 0589cfe0 found in
    _DPH_HEAP_ROOT @ 57a1000
    in free-ed allocation (  DPH_HEAP_BLOCK:         VirtAddr         VirtSize)
                                 57a2034:          589c000             2000
    700d90b2 verifier!AVrfDebugPageHeapFree+0x000000c2
    772375dc ntdll!RtlDebugFreeHeap+0x0000002f
    771fa727 ntdll!RtlpFreeHeap+0x0000005d
    771c68e6 ntdll!RtlFreeHeap+0x00000142
    76f7c584 kernel32!HeapFree+0x00000014
```

After patched version, msxml3!DTD::CheckForwardRefs added an exception handlers, When IDCheck::check raises an exception event, the exception will be capture by msxml3!DTD::CheckForwardRefs. So IDCheck object will be set to zero and memory will not be free, like the following msxml3!Exception::getException the following figure.



```
isassembly
Offset: @$scopeip
6bf2d3ed ebd2            jmp     msxml3!DTD::checkForwardRefs+0x1b (6bf2d3c1)
6bf2d3ef 6a00            push    0
6bf2d3f1 ff75ec          push    dword ptr [ebp-14h]
6bf2d3f4 e88c72fdff      call    msxml3!Exception::fillException (6bf04685)
6bf2d3f9 c3              ret
6bf2d3fa 8b65e8          mov     esp,dword ptr [ebp-18h]
6bf2d3fd e8e185fdff      call    msxml3!Exception::getException (6bf059e3)
6bf2d402 8945e4          mov     dword ptr [ebp-1Ch],eax
6bf2d405 6a01            push    1
6bf2d407 8b4de0          mov     ecx,dword ptr [ebp-20h]
6bf2d40a e89e630600      call    msxml3!IDCheck::`scalar deleting destructor' (6bf937ad)
6bf2d40f c745fcfeffffff  mov     dword ptr [ebp-4],0FFFFFFFEh
```

```
ommand
eip=6bf2d3cc esp=0030e184 ebp=0030e1bc iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
msxml3!DTD::checkForwardRefs+0x26:
6bf2d3cc e801640600      call    msxml3!IDCheck::check (6bf937d2)
0:000> p
```

This is why, program calls msxml3!Document::loadXML again and return the memory (IDCheck object 0589cfe0)to invoker.

```
0:000> u 6e1eafa7

msxml3!Document::loadXML+0xa4:

6e1eafa7 e8e7d7feff     call   msxml3!_MemAlloc (6e1d8793)  // return 0589cfe0

6e1eafac 8bc8           mov    ecx,eax

6e1eafae e8f5020000     call   msxml3!MemoryStreamForStrings::MemoryStreamForStrings (6e1eb2a8)

6e1eafb3 8945e0         mov    dword ptr [ebp-20h],eax

6e1eafb6 57             push   edi

6e1eafb7 68ffffff7f     push   7FFFFFFFh

6e1eafbc ff7508         push   dword ptr [ebp+8]
```

This is another way to repair UAF vulnerability. Hope that we can help you understand the root cause for CVE-2016-0147 vulnerability.