



SA28083 / CVE-2007-6019

Generated by Secunia

9 April, 2008

5 pages

Table of Contents

Introduction	2
Technical Details	2
Exploitation	4
Characteristics	4
Affected Versions	5
Fixed Versions	5
References	5

Introduction:

=====

A vulnerability in Adobe Flash Player when processing Adobe ActionScript Virtual Machine functions can be exploited to cause memory corruption and potentially compromise a vulnerable system.

Technical Details:

=====

The Adobe (formerly Macromedia) Flash Player is frequently installed as a web browser plugin and interprets Flash (SWF) files embedded in web pages. SWF files detail how objects such as text or images should be displayed and can also contain Adobe ActionScript Virtual Machine (AVM) bytecode for scripting. An error when parsing the bytecode for a function declaration for the AVM can lead to an undersized array being allocated resulting in memory corruption and potential arbitrary code execution.

An SWF file is comprised of a number of tags each containing the type, length, and content of the tag. Some tags contain an array of "actions" (AVM bytecodes) representing the script action. An action with an opcode of 0x8E ("Declare Function (V7)") was introduced in SWF version 7 and allows a function to be declared. The function declaration specifies the function name, the number of arguments it accepts, the number of registers that will be used in the function, and a set of flags specifying (among other things) which built-in variables will be preloaded into the registers.

When an ActionScript function is called, sub_3009D358() is called. If the function content has to be parsed from the SWF file, then execution reaches 3009D966h where a buffer object is initialised with a pointer to the SWF data immediately following the function name.

```
.text:3009D966      push    dword ptr [edi+3Ch]
.text:3009D969      lea    ecx, [esi+1Ch]
.text:3009D96C      call   sub_3004A766
.text:3009D971      lea    ecx, [ebp+buffer]
.text:3009D974      call   sub_30001F79 ; buffer object constructor
.text:3009D979      push  7FFFFFFh ; uiBufferSize
.text:3009D97E      push  0 ; uiOffset
.text:3009D980      push  dword ptr [edi+38h] ; lpData, points to DeclareFunction content immediately after function name string
.text:3009D983      lea    ecx, [ebp+buffer] ; int
.text:3009D986      call   sub_30082175 ; initialise buffer object
```

The argument count is parsed from the file content and the pointer incremented to the next field. If the function declaration is SWF version 7 or later (of interest to this vulnerability), then the register count field is read and saved into a function structure.

```
.text:3009D98B      mov    eax, [ebp+buffer.offset]
.text:3009D98E      mov    ecx, [ebp+buffer.lpData]
.text:3009D991      and    [ebp+arg_14], 0 ; function flags
.text:3009D995      add    eax, ecx
.text:3009D997      push  2
.text:3009D999      xor    edx, edx
.text:3009D99B      pop    ebx
.text:3009D99C      add    [ebp+buffer.offset], ebx ; skip over argument count
.text:3009D99F      mov    dh, [eax+1] ; read argument count
.text:3009DA2      mov    dl, [eax]
.text:3009DA4      mov    al, [edi+4Ch]
.text:3009DA7      test   al, al
.text:3009DA9      mov    byte ptr [ebp+arg_4+3], al
```

```
.text:3009D9AC      mov     [ebp+arg_count], edx ; arg count
.text:3009D9AF      jz     short loc_3009D9EE ; pre V7 DeclareFunction?
.text:3009D9B1      mov     eax, [ebp+buffer.offset]
.text:3009D9B4      mov     al, [eax+ecx] ; reg count
.text:3009D9B7      inc     [ebp+buffer.offset] ; skip reg count
.text:3009D9BA      mov     [esi+FunctionStruct.reg_count], al
```

Following the register count is a 16-bit value containing a number of flags indicating the use of internal variables and whether they should be preloaded into registers.

```
.text:3009D9BD      mov     ecx, [ebp+buffer.offset]
.text:3009D9C0      mov     eax, [ebp+buffer.lpData]
.text:3009D9C3      add     [ebp+buffer.offset], ebx
.text:3009D9C6      add     eax, ecx
.text:3009D9C8      xor     ecx, ecx
.text:3009D9CA      mov     ch, [eax+1]
.text:3009D9CD      mov     cl, [eax]
.text:3009D9CF      mov     al, [esi+FunctionStruct.reg_count]
.text:3009D9D2      test   al, al
.text:3009D9D4      mov     [ebp+arg_14], ecx ; flags
.text:3009D9D7      jz     short loc_3009D9EE ; register count is zero?
```

If the register count was non-zero, an array of 32-bit values is allocated from an internal heap by a call to `sub_30093592()` (not shown in disassembly).

```
.text:3009D9D9      mov     ecx, [ebp+this] ; int
.text:3009D9DC      movzx  eax, al
.text:3009D9DF      push   eax ; reg_count
.text:3009D9E0      call  sub_30093592 ; allocate array for registers
.text:3009D9E5      push   eax ; lpArray
.text:3009D9E6      lea   ecx, [esi+FunctionStruct.field_40]
.text:3009D9E9      call  sub_3004A766 ; save pointer to array
```

Function arguments can optionally be preloaded into the function's registers. This is now accomplished in a loop (not shown in disassembly) and correctly ignores requested register indexes larger than the size of the register array.

Execution eventually reaches `3009DC89h` where the function flags are checked to see which internal variables should be preloaded into registers. The internal variables are preloaded into registers in a predetermined order, starting from index one with the "this" variable. The actual value written is either a small constant (such as two meaning undefined) or a pointer to an object representing the variable.

```
.text:3009DC89      test   byte ptr [ebp+arg_14], 1 ; function flags
.text:3009DC8D      mov     byte ptr [ebp+arg_1C+3], 1 ; register index
.text:3009DC91      jz     short loc_3009DCA3 ; don't preload 'this'?
.text:3009DC93      push   [ebp+value] ; "this" object
.text:3009DC96      mov     ecx, esi
.text:3009DC98      push   1 ; index
.text:3009DC9A      call  sub_3006A6E4 ; write "this" register to index 1
.text:3009DC9F      mov     byte ptr [ebp+arg_1C+3], 2 ; update register index to next free
```

In `sub_3006A6E4()`, the pointer to the array is retrieved and the supplied value written to it at the specified index with no bounds checking.

```
.text:3006A72D      mov     eax, [ebp+this]
.text:3006A730      mov     eax, [eax+40h] ; pointer to array
.text:3006A733      mov     ecx, [ebp+index]
.text:3006A736      mov     edx, [ebp+value]
.text:3006A739      pop     edi
.text:3006A73A      pop     esi
.text:3006A73B      mov     [eax+ecx*4], edx ; write to array
```

The "arguments" internal variable is then preloaded into the next register if the function flags indicate that it should be. The value of the variable is retrieved via a

call to sub_300978CA() (not shown in disassembly). The value is again written via a call to sub_3006A6E4() with no bounds checking.

```
.text:3009DCA3      test    byte ptr [ebp+arg_14], 4 ; function flags
.text:3009DCA7      jz     short loc_3009DCCF ; don't preload "arguments"?
.text:3009DCA9      mov    ebx, [ebp+this]
.text:3009DACAC     lea   eax, [ebp+arg_0] ; sub_300978CA() will write "arguments" here
.text:3009DCAF      push  eax
.text:3009DCB0      mov    [ebp+arg_0], 2 ; initialise to undefined
.text:3009DCB7      call  sub_300978CA ; get "arguments"
.text:3009DCBC      movzx  eax, byte ptr [ebp+arg_1C+3]
.text:3009DCC0      pop    ecx
.text:3009DCC1      push  [ebp+arg_0] ; value
.text:3009DCC4      mov    ecx, esi
.text:3009DCC6      push  eax ; index
.text:3009DCC7      call  sub_3006A6E4 ; write "arguments" to register array
.text:3009DCCC      inc   byte ptr [ebp+arg_1C+3] ; increment register index to next free
```

If the flags indicate that they should be preloaded, then the "super", "_root", "_parent", and "_global" internal variables are stored into subsequent indexes in the register array via calls to sub_3006A6E4(). If the register count field specified a number smaller than the number of registers required by the preload flags, then memory corruption will occur when the values are written past the end of the array.

Exploitation:

=====

By tricking a user into viewing a maliciously crafted SWF file in their browser, an attacker can trigger memory corruption via manipulated fields in a function declaration. By specifying that more internal variables should be preloaded than the indicated size of the register array, values will be written past the end of the array in the program controlled heap.

The values written to the heap are not directly user-controlled, however, they appear to be somewhat controllable via scripting. While code execution has not been proven, function table pointers have been successfully overwritten and there are many unexplored permutations available via scripting.

Secunia has developed a PoC for the vulnerability. This is available to customers on Secunia Proof of Concept and Exploit Code Services.

Characteristics:

=====

Detection:

Look for a SWF file containing a tag that contains an action of type 0x8E ("Declare Function (V7)") where the number of registers indicated by the "f_reg_count" field is less than the number of registers required by the preload flags.

Verification:

View a maliciously crafted SWF file in a browser with the Flash Player installed. The malicious SWF file should contain an object (such as text) that contains an action (such as onSelfEvent()) that declares a function. The function declaration should use the 0x8E action, the "f_reg_count" field should be 3, and all register preload flags should be

set. If the function is parsed a number of times, the browser will crash due to memory corruption.

Identification:

Flash9e.ocx version 9.0.115.0 has been confirmed vulnerable. The default installation location is "%WinDir%\system32\Macromed\Flash\".

Affected Versions:

=====

The vulnerability was analysed on Windows XP SP2 with Adobe Flash Player 9.0.115.0 in Internet Explorer 7.

Fixed Versions:

=====

The vulnerability is fixed in Adobe Flash Player 9.0.124.0 (Flash9f.ocx). In the patched version, the size of the register array is verified to be large enough before each variable is written to it.

References:

=====

SA28083:

<http://secunia.com/advisories/28083/>

CVE-2008-6019:

http://secunia.com/cve_reference/CVE-2008-6019/

Secunia Research:

http://secunia.com/secunia_research/2007-103