Abysssec Research

# 1) Advisory information

|  |
| --- |
| Title : Adobe Shockwave player rcsL chunk memory corruption  Version : Shockwave player 11.5.8.612  Discovery : <http://www.abysssec.com>  Vendor : <http://www.adobe.com>  Impact : Critical  Contact : shahin [at] abysssec.com , info [at] abysssec.com  Twitter : @abysssec  CVE : ZeroDay Not Patched |

# 2) Vulnerable version

|  |
| --- |
| [Shockwave Player 11.5.8.612](http://www.filehippo.com/download_shockwave/) last version |

# 3) Vulnerability information

|  |
| --- |
| Class  1- Memory corruption allow command execute  Impact  Successfully exploiting this issue allows remote attackers to execute arbitrary code or cause denial-of-service conditions.  Remotely Exploitable  Yes  Locally Exploitable  Yes |

# 4) Vulnerabilities detail

**Introduction**

Shockwave player is a plug in for loading Adobe Director video files in to the browser. Director movies have DIR or compressed format of DCR. DIR file format is based on RIFF based formats. RIFF formats start with a 4byte RIFX identifier and length of the file. And subsequently chunks come together with format of 4byte chunk identifier + size of chunk + data. Some of the chunk identifiers are tSAC, pami, rcsL.

By help of our simple fuzzer we have manipulated a director movie file and found a vulnerability in part of an existing rcsL chunk.

**Vulnerability explanation**

There is a 4bytes value in the undocumented rcsL chunk in our sample director movie and it may be possible to find similar rcsL chunks in other director samples. The 4bytes so called value can be manipulated to reach the vulnerable part of function 68122990. Here is the function:

|  |
| --- |
| text:68122990 sub\_68122990 proc near ; CODE XREF: sub\_68112120+1A57p  .text:68122990 ; DATA XREF: sub\_68122F30+4AAo  .text:68122990  .text:68122990 var\_8 = dword ptr -8  .text:68122990 var\_4 = dword ptr -4  .text:68122990 arg\_0 = dword ptr 4  .text:68122990 arg\_4 = dword ptr 8  .text:68122990  .text:68122990 sub esp, 8  .text:68122993 mov eax, [esp+8+arg\_4]  .text:68122997 push ebx  .text:68122998 push ebp  .text:68122999 push esi  .text:6812299A mov esi, [esp+14h+arg\_0]  .text:6812299E push edi  .text:6812299F push eax  .text:681229A0 push esi  .text:681229A1 call sub\_680FC6D0  .text:681229A6 mov ecx, [esi+18h]  .text:681229A9 mov edx, [esi+10h]  .text:681229AC mov ebp, [esi+1Ch]  .text:681229AF mov ebx, [esi+20h]  .text:681229B2 add ecx, 0FFFFFFF8h  .text:681229B5 cmp ebp, 3  .text:681229B8 mov [esp+18h+arg\_0], eax  .text:681229BC mov [esi+18h], ecx  .text:681229BF mov eax, [edx]  .text:681229C1 mov edx, [eax+ecx]  .text:681229C4 lea edi, [esi+1Ch]  .text:681229C7 mov [edi], edx  .text:681229C9 mov eax, [eax+ecx+4]  .text:681229CD mov [edi+4], eax  .text:681229D0 mov [esp+18h+var\_8], 4  .text:681229D8 mov [esp+18h+var\_4], 0  .text:681229E0 jz short loc\_681229F6  .text:681229E2 push ebx  .text:681229E3 push ebp  .text:681229E4 push 0Ch  .text:681229E6 push esi  .text:681229E7 call sub\_680FCFB0  .text:681229EC pop edi  .text:681229ED pop esi  .text:681229EE pop ebp  .text:681229EF pop ebx  .text:681229F0 add esp, 8  .text:681229F3 retn 8  .text:681229F6 ; ---------------------------------------------------------------------------  .text:681229F6  .text:681229F6 loc\_681229F6: ; CODE XREF: sub\_68122990+50j  .text:681229F6 mov ecx, [ebx]  .text:681229F8 mov edx, [ecx]  .text:681229FA mov ecx, [esp+18h+arg\_0]  .text:681229FE lea eax, [esp+18h+var\_8]  .text:68122A02 push eax  .text:68122A03 push ecx  .text:68122A04 push ebx  .text:68122A05 push esi  .text:68122A06 call dword ptr [edx+2Ch]  .text:68122A09 mov ecx, [esi+7Ch]  .text:68122A0C test ecx, ecx  .text:68122A0E jz short loc\_68122A22  .text:68122A10 push ebx  .text:68122A11 push ebp  .text:68122A12 push esi  .text:68122A13 call sub\_680FC730  .text:68122A18 pop edi  .text:68122A19 pop esi  .text:68122A1A pop ebp  .text:68122A1B pop ebx  .text:68122A1C add esp, 8  .text:68122A1F retn 8  .text:68122A22 ; ---------------------------------------------------------------------------  .text:68122A22  .text:68122A22 loc\_68122A22: ; CODE XREF: sub\_68122990+7Ej  .text:68122A22 test eax, eax  .text:68122A24 jnz loc\_68122AAC  .text:68122A2A push esi  .text:68122A2B call sub\_680FD9D0  .text:68122A30 push edi  .text:68122A31 push esi  .text:68122A32 mov [edi], ebp  .text:68122A34 mov [edi+4], ebx  .text:68122A37 call sub\_680FC7C0  .text:68122A3C push esi  .text:68122A3D call sub\_680FD9D0  .text:68122A42 mov eax, [esp+18h+arg\_4]  .text:68122A46 mov edx, [esi+28h]  .text:68122A49 mov [esi+0A4h], eax  .text:68122A4F mov dword ptr [esi+20h], 80000001h  .text:68122A56 mov ecx, [edx]  .text:68122A58 lea eax, [eax+eax\*2]  .text:68122A5B push esi  .text:68122A5C call dword ptr [ecx+eax\*8+20h]  .text:68122A60 mov eax, [esi+7Ch]  .text:68122A63 test eax, eax  .text:68122A65 jz short loc\_68122A85  .text:68122A67 cmp eax, 4  .text:68122A6A jnz short loc\_68122ACE  .text:68122A6C mov edx, [esp+18h+arg\_0]  .text:68122A70 push edx  .text:68122A71 push 8  .text:68122A73 push 37h  .text:68122A75 push esi  .text:68122A76 call sub\_680FD040  .text:68122A7B pop edi  .text:68122A7C pop esi  .text:68122A7D pop ebp  .text:68122A7E pop ebx  .text:68122A7F add esp, 8  .text:68122A82 retn 8  .text:68122A85 ; ---------------------------------------------------------------------------  .text:68122A85  .text:68122A85 loc\_68122A85: ; CODE XREF: sub\_68122990+D5j  .text:68122A85 mov eax, [edi]  .text:68122A87 mov ecx, [edi+4]  .text:68122A8A mov edx, [esi+10h]  .text:68122A8D mov [esp+18h+var\_8], eax  .text:68122A91 mov eax, [esi+18h]  .text:68122A94 add eax, 0FFFFFFF8h  .text:68122A97 mov [esp+18h+var\_4], ecx  .text:68122A9B mov [esi+18h], eax  .text:68122A9E mov ecx, [edx]  .text:68122AA0 mov edx, [ecx+eax]  .text:68122AA3 mov [edi], edx  .text:68122AA5 mov eax, [ecx+eax+4]  .text:68122AA9 mov [edi+4], eax  .text:68122AAC  .text:68122AAC loc\_68122AAC: ; CODE XREF: sub\_68122990+94j  .text:68122AAC push ebx  .text:68122AAD push ebp  .text:68122AAE push esi  .text:68122AAF call sub\_680FC730  .text:68122AB4 mov eax, [esi+7Ch]  .text:68122AB7 test eax, eax  .text:68122AB9 jnz short loc\_68122ACE  .text:68122ABB push esi  .text:68122ABC call sub\_680FD9D0  .text:68122AC1 mov ecx, [esp+18h+var\_8]  .text:68122AC5 mov edx, [esp+18h+var\_4]  .text:68122AC9 mov [edi], ecx  .text:68122ACB mov [edi+4], edx  .text:68122ACE  .text:68122ACE loc\_68122ACE: ; CODE XREF: sub\_68122990+DAj  .text:68122ACE ; sub\_68122990+129j  .text:68122ACE pop edi  .text:68122ACF pop esi  .text:68122AD0 pop ebp  .text:68122AD1 pop ebx  .text:68122AD2 add esp, 8  .text:68122AD5 retn 8  .text:68122AD5 sub\_68122990 endp |

In the above function we have direct control on the second argument of the function. By manipulating the argument in rcsL chunk we reach to an indirect call that is based on our arguments:

|  |
| --- |
| .text:68122A42 mov eax, [esp+18h+arg\_4]  .text:68122A46 mov edx, [esi+28h]  .text:68122A49 mov [esi+0A4h], eax  .text:68122A4F mov dword ptr [esi+20h], 80000001h  .text:68122A56 mov ecx, [edx]  .text:68122A58 lea eax, [eax+eax\*2]  .text:68122A5B push esi  .text:68122A5C call dword ptr [ecx+eax\*8+20h] |

The above code is our vulnerable part. EAX register is set with second argument that we have control on it and ESI is first argument of the function and is a pointer to a dynamic allocated structure in heap. Value of offset 28h of the structure that is unknown is set in ECX register and finally an indirect call to the 'ECX+EAX\*24+20h' is done. Because result of EAX\*24 is a large value and we have complete control on EAX register we can almost control first byte of our indirect call pointer without the need of ECX register.

**Exploit:**

For exploitation purpose because we don't have a fixed address in our call we cannot control the execution flow to an exact value but we can jump to a specific range because we have control on first bytes of the pointer of indirect call. So here by abusing javascript we can use old-school heap spray technic to fill memory with nops+shellcode and call to this range.

To control the 4 bytes EAX register in our exploit we manipulated 4bytes at offset 4C4B of the file to value FFF00267.

An important hint here is that because we call the indirect pointer the EIP is set to nops itself. As you know an EIP of 90909090 is invalid. But we can use other opcodes as nopslides that doesn’t have any effect. In our test sample we used 0a0a0a0a as both base range of heap spray and nopslides because 0a0a opcode is an OR instruction on some unimportant registers.

The sample + exploit are tested on patched windows XP service pack 3.