



SA29941 / CVE-2008-1104

Generated by Secunia

20 May, 2008

5 pages

Table of Contents

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

Introduction:

=====

A vulnerability in Foxit Reader when viewing a PDF document during the processing of the "util.printf()" Javascript method can be exploited by malicious people to compromise a user's system.

Technical Details:

=====

PDF documents can contain Javascript that can be linked to a number of different events. Foxit Reader implements an Adobe Javascript API defining objects and methods to interact with the viewer in many ways. An error in the parsing of parameters to the documented "util.printf()" method may lead to a stack-based buffer overflow allowing arbitrary code execution.

In Foxit Reader, the Javascript engine is implemented in the main executable. The "util" API object is documented in the "Javascript for Acrobat API Reference" available from the Adobe website. Always available, the "util" object exposes a number of utility functions such as "printf".

When "util.printf()" is executed, eventually sub_64FE90() is called to perform the processing and is passed a pointer to an array of objects representing the arguments.

The first argument (the format string) is parsed and for each format specifier, sub_64FD70() is called (not shown in disassembly) to identify the class of format specifier.

```
.text:00650162 loc_650162:          ; CODE XREF: sub_64FE90+2CB
.text:00650162          push   eax                ; lpwzFormatSpec
.text:00650163          call   sub_64FD70         ; get format specifier class
.text:00650163          ; -1: invalid
.text:00650163          ; 0: integer
.text:00650163          ; 1: floating point
.text:00650163          ; 2: string
.text:00650163          ; 3: pointer
.text:00650168          add    esp, 4
.text:0065016B          cmp    eax, 3             ; switch 4 cases
.text:0065016E          ja     loc_650238         ; default
.text:00650174          jmp    ds:off_650460[eax*4] ; switch jump
```

In the case of a floating point format specifier (type 1), execution reaches loc_6501A9 where the current argument is converted to a floating point value and stored on the stack.

```
.text:006501A9 loc_6501A9:          ; CODE XREF: sub_64FE90+2E4
.text:006501A9          ; DATA XREF: .text:off_650460
.text:006501A9          mov    esi, [ebp+lpBS_FormatString+4] ; jumtable 00650174 case 1
.text:006501A9          ; floating point
.text:006501AC          test   esi, esi
.text:006501AE          jnz   short loc_6501B5 ; verify format string is not empty?
...
.text:006501B5 loc_6501B5:          ; CODE XREF: sub_64FE90+31E
.text:006501B5          mov    eax, [ebp+lpStartArgs]
.text:006501B8          mov    ecx, ebx           ; current argument index
.text:006501BA          shl   ecx, 5
.text:006501BD          add   ecx, eax           ; pointer to current argument in array
.text:006501BF          call  sub_693360         ; get floating pointer representation of variable
.text:006501C4          fstp  qword ptr [ebp+var_24]
```

The floating point value and the current format specifier are passed to sub_4B32F0() to be rendered as a string.

```
.text:006501C7      mov     eax, [ebp+var_24+4]
.text:006501CA      mov     ecx, [ebp+var_24]
.text:006501CD      push   eax
.text:006501CE      push   ecx          ; int
.text:006501CF      lea   edx, [ebp+lpwzString]
.text:006501D2      push   esi          ; Format
.text:006501D3      push   edx          ; ppwzString
.text:006501D4      call  sub_4B32F0    ; wrapper for sprintf into correctly sized heap buffer
```

sub_4B32F0() is simply a wrapper for sub_4B2E90().

```
.text:004B32F0 ; int __cdecl sub_4B32F0(void *ppwzString, wchar_t *Format, int)
.text:004B32F0 sub_4B32F0      proc near          ; CODE XREF: sub_63E400+181
.text:004B32F0                                     ; sub_645DB0+20C ...
.text:004B32F0
.text:004B32F0 ppwzString      = dword ptr  4
.text:004B32F0 Format          = dword ptr  8
.text:004B32F0 firstArg       = dword ptr  0Ch
.text:004B32F0
.text:004B32F0      mov     ecx, [esp+Format]
.text:004B32F4      lea   eax, [esp+firstArg]
.text:004B32F8      push   eax          ; int
.text:004B32F9      push   ecx          ; Format
.text:004B32FA      mov   ecx, [esp+8+ppwzString]
.text:004B32FE      call  sub_4B2E90    ; sprintf into correctly sized heap buffer
```

sub_4B2E90() allocates a 256-byte stack buffer and then parses the format specifier by iterating a loop once for each '%' found.

```
.text:004B2E90 ; int __stdcall sub_4B2E90(wchar_t *Format, int)
.text:004B2E90 sub_4B2E90      proc near          ; CODE XREF: sub_4B32F0+E
.text:004B2E90
.text:004B2E90 var_12C      = qword ptr -12Ch
.text:004B2E90 var_114      = dword ptr -114h
.text:004B2E90 var_110      = dword ptr -110h
.text:004B2E90 runningTotalChars= dword ptr -10Ch
.text:004B2E90 var_108      = dword ptr -108h
.text:004B2E90 var_104      = dword ptr -104h
.text:004B2E90 szTemp      = byte ptr -100h      ; char[256]
.text:004B2E90 Format      = dword ptr  4
.text:004B2E90 arg_4       = dword ptr  8
.text:004B2E90
.text:004B2E90      sub     esp, 114h
.text:004B2E96      push   ebx
.text:004B2E97      push   ebp
.text:004B2E98      mov   ebp, [esp+11Ch+arg_4]
.text:004B2E9F      push   esi
.text:004B2EA0      mov   esi, [esp+120h+Format]
.text:004B2EA7      push   edi
.text:004B2EA8      xor   ebx, ebx
.text:004B2EAA      mov   edi, ecx
.text:004B2EAC      cmp   [esi], bx
.text:004B2EAF      mov   [esp+124h+var_104], edi
.text:004B2EB3      mov   [esp+124h+var_108], ebp
.text:004B2EB7      mov   [esp+124h+runningTotalChars], ebx
.text:004B2EBB      jz    loc_4B322A
.text:004B2EC1
.text:004B2EC1 @loop_parse_format_specifier: ; CODE XREF: sub_4B2E90+394
.text:004B2EC1      cmp   word ptr [esi], '%'
.text:004B2EC5      jnz   loc_4B320E
```

After parsing any flags (not shown in disassembly), the optional width and precision values are parsed and saved.

```
.text:004B2F34 loc_4B2F34: ; CODE XREF: sub_4B2E90+55
.text:004B2F34      push   esi          ; Str
.text:004B2F35      call  __wtoi
.text:004B2F3A      mov   [esp+128h+uWidth], eax
...
.text:004B2F87 loc_4B2F87: ; CODE XREF: sub_4B2E90+E6
.text:004B2F87      push   esi          ; Str
.text:004B2F88      call  __wtoi
.text:004B2F8D      mov   [esp+128h+uPrecision], eax
```

Eventually, the format specifier type field is read and if an 'f' is encountered, execution reaches loc_4B31A2. The previously read width and precision are used to render the floating point value into the 256-byte stack buffer with no bounds check.

```
.text:004B31A2 loc_4B31A2:                ; CODE XREF: sub_4B2E90+1C2
.text:004B31A2                        ; DATA XREF: .text:off_4B329C
.text:004B31A2      fld      qword ptr [ebp+0] ; jumptable 004B3052 case 31 ('f')
.text:004B31A5      mov     eax, [esp+124h+uPrecision]
.text:004B31A9      mov     ecx, [esp+124h+uWidth]
.text:004B31AD      add     ebp, 8
.text:004B31B0      sub     esp, 8
.text:004B31B3      fstp   [esp+12Ch+var_12C]
.text:004B31B6      add     eax, 6
.text:004B31B9      lea   edx, [esp+12Ch+szTemp] ; char[256]
.text:004B31BD      push  eax
.text:004B31BE      push  ecx
.text:004B31BF      push  offset Format      ; "%. *F"
.text:004B31C4      push  edx                ; Dest
.text:004B31C5      call  _sprintf
```

Exploitation:

=====

A PDF document containing specially crafted Javascript code that is executed when the document is viewed may trigger the vulnerability. A stack-based buffer overflow will ensue if the resulting value exceeds 256 characters.

Exploiting the vulnerability for code execution is mitigated by the use of a stack cookie protecting the return address. However, a structured exception handler can easily be overwritten. Limits on the values that can be written (spaces, digits, and period) provide a hurdle to the attacker, but code execution has been proved via heap-spraying.

Secunia Research has developed a PoC and working exploit for the vulnerability. These are available to customers on Secunia Proof of Concept and Exploit Code Services.

Characteristics:

=====

Detection:

Look for a PDF containing Javascript code that calls the "util.printf()" method with a "%f" floating point specification where the size would exceed 256 characters.

Verification:

Create a PDF document containing an element with an action that calls some Javascript code. Set the Javascript code to 'util.printf("%50000f",1);'. When the action is fired, a vulnerable Foxit Reader crashes due to the stack corruption.

Identification:

The vulnerability has been confirmed in Foxit Reader 2.3 build 2825. The default installation location of "Foxit Reader.exe" is "%ProgramFiles%\Foxit Software\Foxit Reader"

Tested Versions:

=====

The vulnerability was analysed on Windows XP SP2 with Foxit Reader 2.2 build 2129.

Fixed Versions:

=====

The vulnerability is patched by upcoming version 2.3 build 2912.

References:

=====

SA29941:

<http://secunia.com/advisories/29941/>

CVE-2008-1104:

http://secunia.com/cve_reference/CVE-2008-1104/

Secunia Research:

http://secunia.com/secunia_research/2008-18/