Talos Vulnerability Report

TALOS-2021-1265

Nitro Pro PDF JavaScript local_file_path Object use-after-free vulnerability

OCTOBER 13, 2021

CVE NUMBER

CVE-2021-21796

Summary

An exploitable use-after-free vulnerability exists in the JavaScript implementation of Nitro Pro PDF. A specially crafted document can cause an object containing the path to a document to be destroyed and then later reused, resulting in a use-after-free vulnerability, which can lead to code execution under the context of the application. An attacker can convince a user to open a document to trigger this vulnerability.

Tested Versions

Nitro Pro 13.31.0.605
Nitro Pro 13.33.2.645

Product URLs

https://www.gonitro.com/nps/product-details/downloads

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

Nitro Software Inc. develops a variety of feature-rich PDF software that allows users to read and manipulate the files. This includes their flagship product, Nitro Pro, as part of their Nitro Productivity Suite. This product allows users to create and modify PDFs and other digital documents. This software includes support for several capabilities via third-party libraries to parse the PDF specification. This includes software produced by Kakadu Software for providing JPEG2000 image file format support, the LibTIFF library for providing support for TIFF image files, and Mozilla's SpiderMonkey for providing JavaScript support within their software.

In order to support many of the features of Nitro PDF, the application implements a local dispatching interface for its various components to be able to communicate with it. Specifically, this dispatching interface is an array of functions that are collectively referred to as the HFT Extension Manager and is of the type `HFTServer`. Upon the application populating this array, various plugins will then be loaded by the application and then register their capabilities with the `HFTServer` host. The Sixth Edition of the Portable Document Format (PDF) specification includes a javascript extension in order to allow document creators to improve the interactivity of their documents. Thus, the application implements this capability by loading a javascript plugin and registering it via the HFT Extension Manager interface. The javascript implementation that is used by the application is based on Mozilla's SpiderMonkey, and includes a number of bindings that allow a content developer to automate various aspects of the document.

The SpiderMonkey library allows a developer to develop their own custom classes and objects via its API in order to enable a document creator to script various parts of the application. When the application initializes its javascript plugin, the application needs to create a number of objects and classes at in order to expose various PDF automation points to the document creator. The following code represents a closure (or an anonymous function) and is responsible for initializing all of the available javascript classes. Firstly at [1], a number of objects that were captured from the encompassing function are extracted and then written to globals that are accessible by the plugin. This includes the parent object, the global root object for garbage collection, and the `pdf_java_script_actions` object. After assigning the captured variables, the function call at [2] is executed to define the "app" object by attaching its private data along with any necessary methods or properties.

```
np_java_script+0x511f0:
2bdd11f0 55              push    ebp
2bdd11f1 8bec            mov     ebp,esp
2bdd11f3 83ec40          sub     esp,40h
2bdd11f6 a17441ee2b      mov     eax,dword ptr [np_java_script!CxImageJPG::`vftable'+0x4f8b8 (2bee4174)]
2bdd11fb 33c5            xor     eax,ebp
2bdd11fd 8945fc          mov     dword ptr [ebp-4],eax
2bdd1200 53              push    ebx
2bdd1201 8b5d08          mov     ebx,dword ptr [ebp+8]
2bdd1204 894dc4          mov     dword ptr [ebp-3Ch],ecx                                         ; this
...
2bdd120d e8de9e0000      call    np_java_script!nitro::java_script::create_plugin+0x9890 (2bddb0f0)
2bdd1212 8bcb            mov     ecx,ebx
2bdd1214 a3dce7ee2b      mov     dword ptr [np_java_script!CxImageJPG::`vftable'+0x59f20 (2beee7dc)],eax    ; [1] parent plugin object
containing global state
2bdd1219 e8c29e0000      call    np_java_script!nitro::java_script::create_plugin+0x9880 (2bddb0e0)
2bdd121e 8bcb            mov     ecx,ebx
2bdd1220 a3d8e7ee2b      mov     dword ptr [np_java_script!CxImageJPG::`vftable'+0x59f1c (2beee7d8)],eax    ; root JSContext
2bdd1225 e8d69e0000      call    np_java_script!nitro::java_script::create_plugin+0x98a0 (2bddb100)
2bdd122a 8bcb            mov     ecx,ebx
2bdd122c a3e4e7ee2b      mov     dword ptr [np_java_script!CxImageJPG::`vftable'+0x59f28 (2beee7e4)],eax    ; pdf_java_script_actions
2bdd1231 e8da9e0000      call    np_java_script!nitro::java_script::create_plugin+0x98b0 (2bddb110)
2bdd1236 a3e0e7ee2b      mov     dword ptr [np_java_script!CxImageJPG::`vftable'+0x59f24 (2beee7e0)],eax    ; array of objects
2bdd123b e88059fbff      call    np_java_script+0x6bc0 (2bd86bc0)                                ; [2] define app object and
attach its private data
2bdd1240 85c0            test    eax,eax
2bdd1242 0f8450020000    je      np_java_script+0x51498 (2bdd1498)
```

In the implementation of the mentioned function call, the methods and properties for the “app” object are assigned and stored into arrays on the stack of both the `JSFunctionSpec` and `JSPropertySpec` type. At [3], the `JSNative` implementations for the “app.newDoc” and “app.openDoc” javascript functions are assigned so that when “app.newDoc” or “app.openDoc” is called, the correct code will be dispatched to. After the `JSFunctionSpec` array has been populated, the function will then proceed to create an instance of the “App” class using its `JSClass` definition and the object’s name “app” as the parameters for the `JS_DefineObject` function call at [4]. Next when the array of `JSFunctionSpec` and `JSPropertySpec` elements have both been initialized, the function will proceed to use both of them by registering the properties with the `JS_DefineProperties` call at [6], and then the functions with the call to the `JS_DefineFunctions` function at [7].

```
np_java_script+0x6bc0:
2bd86bc0 55              push    ebp
2bd86bc1 8bec            mov     ebp,esp
2bd86bc3 81ec94010000    sub     esp,194h
2bd86bc9 a17441ee2b      mov     eax,dword ptr [np_java_script!CxImageJPG::`vftable'+0x4f8b8 (2bee4174)]
2bd86bce 33c5            xor     eax,ebp
2bd86bd0 8945fc          mov     dword ptr [ebp-4],eax
...
np_java_script+0x6e60:
2bd86e60 c7458c2409e82b  mov     dword ptr [ebp-74h],offset np_java_script!CxImageJPG::Encode+0x9e9a4 (2be80924)    ; [3]
JSFunctionSpec.name = "newDoc"
2bd86e67 c745905078d82b  mov     dword ptr [ebp-70h],offset np_java_script+0x7850 (2bd87850)                        ; JSFunctionSpec.call
2bd86e6e c7459402000000  mov     dword ptr [ebp-6Ch],2                                                             ; JSFunctionSpec.nargs
2bd86e75 894598          mov     dword ptr [ebp-68h],eax
2bd86e78 c7459c2c09e82b  mov     dword ptr [ebp-64h],offset np_java_script!CxImageJPG::Encode+0x9e9ac (2be8092c)    ; [3]
JSFunctionSpec.name = "openDoc"
2bd86e7f c745a0607ad82b  mov     dword ptr [ebp-60h],offset np_java_script+0x7a60 (2bd87a60)                        ; JSFunctionSpec.call
2bd86e86 c745a405000000  mov     dword ptr [ebp-5Ch],5                                                             ; JSFunctionSpec.nargs
2bd86e8d 8945a8          mov     dword ptr [ebp-58h],eax
...
np_java_script+0x6e97:
2bd86e97 6a06            push    6                                                                                 ; flags
2bd86e99 50              push    eax                                                                               ; class prototype
2bd86e9a 680010ee2b      push    offset np_java_script!CxImageJPG::`vftable'+0x4c744 (2bee1000)                    ; [4] JSClass for "App"
2bd86e9f 686409e82b      push    offset np_java_script!CxImageJPG::Encode+0x9e9e4 (2be80964)                       ; [4] "app"
2bd86ea4 ff35dce7ee2b    push    dword ptr [np_java_script!CxImageJPG::`vftable'+0x59f20 (2beee7dc)]               ; parent object
containing global state
2bd86eaa c745b0e07fd82b  mov     dword ptr [ebp-50h],offset np_java_script+0x7fe0 (2bd87fe0)
2bd86eb1 ff35d8e7ee2b    push    dword ptr [np_java_script!CxImageJPG::`vftable'+0x59f1c (2beee7d8)]               ; root JSContext
...
np_java_script+0x6f0d:
2bd86f0d ff1514e7e72b    call    dword ptr [np_java_script!CxImageJPG::Encode+0x9c794 (2be7e714)]                  ; [5] call
JS_DefineObject
2bd86f13 8bf0            mov     esi,eax                                                                           ; save new JSObject
2bd86f15 83c418          add     esp,18h
2bd86f18 85f6            test    esi,esi
2bd86f1a 743f            je      np_java_script+0x6f5b (2bd86f5b)
...
2bd86f1c 8d856cfeffff    lea     eax,[ebp-194h]                                                                    ; address of
JSPropertySpec array
2bd86f22 50              push    eax                                                                               ; ps
2bd86f23 56              push    esi                                                                               ; obj
2bd86f24 ff35d8e7ee2b    push    dword ptr [np_java_script!CxImageJPG::`vftable'+0x59f1c (2beee7d8)]               ; root JSContext
2bd86f2a ff1518e7e72b    call    dword ptr [np_java_script!CxImageJPG::Encode+0x9c798 (2be7e718)]                  ; [6]
JS_DefineProperties
2bd86f30 83c40c          add     esp,0Ch
2bd86f33 85c0            test    eax,eax
2bd86f35 7446            je      np_java_script+0x6f7d (2bd86f7d)
2bd86f37 8d85ecfeffff    lea     eax,[ebp-114h]                                                                    ; address of
JSFunctionSpec array
2bd86f3d 50              push    eax                                                                               ; fs
2bd86f3e 56              push    esi                                                                               ; obj
2bd86f3f ff35d8e7ee2b    push    dword ptr [np_java_script!CxImageJPG::`vftable'+0x59f1c (2beee7d8)]               ; root JSContext
2bd86f45 ff1538e7e72b    call    dword ptr [np_java_script!CxImageJPG::Encode+0x9c7b8 (2be7e738)]                  ; [7] JS_DefineFunctions
2bd86f4b 83c40c          add     esp,0Ch
2bd86f4e 85c0            test    eax,eax
2bd86f50 742b            je      np_java_script+0x6f7d (2bd86f7d)
```

The binding for the “app.newDoc” function is implemented by the following code. This function will first get the private data associated with the global “app” object at [8], and then proceed to check the parameters that were passed to the “app.newDoc” function. After the parameters were checked in order to determine whether the width and height or an object containing these properties were passed to it, the function will extract both the width and the height and pass them along with the private app data to the `AppNewDoc` function call at [9]. Inside the `AppNewDoc` function, a structured exception handler is setup and the canary for the frame is initialized. At [10], the function will then pass the width and height that were received from the parameters to the `npdf.dll!PDDocCreate` function and then store the returned document object in the `%ecx` register. This object will then be passed as a parameter to the `npdf.dll!PDDocCreatePage` function call at [11] in order to have a page attached to it. Once the page has been attached, at [12] a call will be made to the HFT Extension Manager function at offset `+0x314` in order to signal the HFT Server to execute the `AVDocOpenFromPDDocWithParams` function.

```
np_java_script+0x7850:
2bd87850 55              push    ebp
2bd87851 8bec            mov     ebp,esp
2bd87853 83ec10          sub     esp,10h
2bd87856 8b4518          mov     eax,dword ptr [ebp+18h]
2bd87859 53              push    ebx
2bd8785a 56              push    esi
2bd8785b 57              push    edi
2bd8785c ff750c          push    dword ptr [ebp+0Ch]
2bd8785f c70001000080    mov     dword ptr [eax],80000001h
2bd87865 e896dcffff      call    np_java_script+0x5500 (2bd85500)      ; [8] get app private data
2bd8786a 83c404          add     esp,4
2bd8786d 8945f0          mov     dword ptr [ebp-10h],eax
...
np_java_script+0x7951:
2bd87951 53              push    ebx                                   ; width
2bd87952 50              push    eax                                   ; height
2bd87953 ff75f0          push    dword ptr [ebp-10h]                   ; app private data
2bd87956 e855cdffff      call    np_java_script+0x46b0 (2bd846b0)      ; [9] \ AppNewDoc
2bd8795b 57              push    edi
2bd8795c 8bf0            mov     esi,eax
\
np_java_script+0x46b0:
2bd846b0 55              push    ebp
2bd846b1 8bec            mov     ebp,esp
...
np_java_script+0x46d9:
2bd846d9 8b750c          mov     esi,dword ptr [ebp+0Ch]               ; width
2bd846dc b864020000      mov     eax,264h
2bd846e1 8b7d10          mov     edi,dword ptr [ebp+10h]               ; height
2bd846e4 85f6            test    esi,esi
2bd846e6 c745fc00000000  mov     dword ptr [ebp-4],0
2bd846ed 0f4ef0          cmovle  esi,eax
2bd846f0 c645fc01        mov     byte ptr [ebp-4],1
2bd846f4 b818030000      mov     eax,318h
2bd846f9 85ff            test    edi,edi
2bd846fb 0f4ef8          cmovle  edi,eax
2bd846fe ff152cf0e72b    call    dword ptr [np_java_script!CxImageJPG::Encode+0x9d0ac (2be7f02c)]   ; [10] npdf.dll!PDDocCreate
2bd84704 660f6ed6        movd    xmm2,esi
2bd84708 8bc8            mov     ecx,eax                               ; store pd_doc
...
np_java_script+0x470e:
2bd8470e 83ec20          sub     esp,20h
2bd84711 f30fe6d2        cvtdq2pd xmm2,xmm2
2bd84715 8bc4            mov     eax,esp
2bd84717 894d0c          mov     dword ptr [ebp+0Ch],ecx
2bd8471a 6aff            push    0FFFFFFFFh
2bd8471c 51              push    ecx                                                                ; document (pd_doc)
2bd8471d f30fe6c9        cvtdq2pd xmm1,xmm1
2bd84721 0f57c0          xorps   xmm0,xmm0
2bd84724 660f14d1        unpcklpd xmm2,xmm1
2bd84728 0f1100          movups  xmmword ptr [eax],xmm0                                             ; page dimensions
2bd8472b 0f115010        movups  xmmword ptr [eax+10h],xmm2                                         ; page dimensions
2bd8472f ff1528f0e72b    call    dword ptr [np_java_script!CxImageJPG::Encode+0x9d0a8 (2be7f028)]   ; [11] npdf.dll!PDDocCreatePage
2bd84735 83c428          add     esp,28h
...
np_java_script+0x473f:
2bd8473f a128ebee2b      mov     eax,dword ptr [np_java_script!CxImageJPG::`vftable'+0x5a26c (2beeeb28)]    ; HFT Extension Manager
2bd84744 83c404          add     esp,4
2bd84747 8b8014030000    mov     eax,dword ptr [eax+314h]                                           ; +0x314
2bd8474d 6a00            push    0
2bd8474f ff750c          push    dword ptr [ebp+0Ch]
2bd84752 ffd0            call    eax                                                                ; [12] HFTServer
```

When using the regular NitroPDF application as an HFT Server, the HFT Extension Manager function at offset `+0x314` is implemented by the following function. This function is simply a wrapper around another function that is used to update the `pd_doc` object that was returned from the prior mentioned function call. At [13], the `pd_doc` that was passed as a parameter is then reused as a parameter to the `AVDocOpenFromPDDocWithParams` function. One characteristic of this function is that it is also registered in the HFT Extension Manager at offset `+0x26c`.

```
NitroPDF!CxIOFile::Write+0x79670:
00452c80 55              push    ebp
00452c81 8bec            mov     ebp,esp
00452c83 6aff            push    0FFFFFFFFh
00452c85 68c81f7900      push    offset NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x2475f8 (00791fc8)
00452c8a 64a100000000    mov     eax,dword ptr fs:[00000000h]
00452c90 50              push    eax
00452c91 83ec54          sub     esp,54h
00452c94 56              push    esi
00452c95 a1a45d9400      mov     eax,dword ptr [NitroPDF!CxImageJPG::`vftable'+0x15c3fc (00945da4)]
00452c9a 33c5            xor     eax,ebp
00452c9c 50              push    eax
00452c9d 8d45f4          lea     eax,[ebp-0Ch]
00452ca0 64a300000000    mov     dword ptr fs:[00000000h],eax
...
NitroPDF!CxIOFile::Write+0x796b8:
00452cc8 8d45a0          lea     eax,[ebp-60h]
00452ccb c745a04c000000  mov     dword ptr [ebp-60h],4Ch
00452cd2 50              push    eax
00452cd3 ff750c          push    dword ptr [ebp+0Ch]
00452cd6 ff7508          push    dword ptr [ebp+8]                             ; pd_doc
00452cd9 e822000000      call    NitroPDF!CxIOFile::Write+0x796f0 (00452d00)   ; [13] AVDocOpenFromPDDocWithParams
00452cde 83c418          add     esp,18h
```

At the beginning of the `AVDocOpenFromPDDocWithParams` function, at [14] a structured exception handler will first be registered prior to checking the function's parameters. This exception handler is directly relevant to the vulnerability described within this document and will later be used to destroy the `local_file_path` object that triggers the vulnerability. After checking the `AVDocOpenFromPDDocWithParams` function's parameters, the function will pass its `pd_doc` parameter to the `npdf.dll!PDDocGetFile` call at [15]. This function will typically return the file associated with the `pd_doc`, however, since this `pd_doc` object was just created using the "app.newDoc" javascript function it will return `NULL` as an error. This will result in the branch that follows being skipped and allow for execution to continue. If there is no file associated with the `pd_doc`, the function at [16] will then be called. This function's responsibility is to generate a temporary filename as a `local_file_path` object and then store it to its parameter.

```
NitroPDF!CxIOFile::Write+0x796f0:
00452d00 55              push    ebp
00452d01 8bec            mov     ebp,esp
00452d03 6aff            push    0FFFFFFFFh
00452d05 6825207900      push    offset NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x247655 (00792025)        ;
[14] exception handler
00452d0a 64a100000000    mov     eax,dword ptr fs:[00000000h]
00452d10 50              push    eax
00452d11 81ec80000000    sub     esp,80h
...
NitroPDF!CxIOFile::Write+0x79761:
00452d71 33db            xor     ebx,ebx
00452d73 c6458f00        mov     byte ptr [ebp-71h],0
00452d77 57              push    edi                                                                                                  ;
pd_doc
00452d78 895d84          mov     dword ptr [ebp-7Ch],ebx
00452d7b c645fc02        mov     byte ptr [ebp-4],2
00452d7f ff159cca7b00    call    dword ptr [NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x2720cc (007bca9c)]    ;
[15] npdf.dll!PDDocGetFile
00452d85 83c404          add     esp,4
00452d88 85c0            test    eax,eax
00452d8a 0f8598000000    jne     NitroPDF!CxIOFile::Write+0x79818 (00452e28)
...
00452d90 8d45d0          lea     eax,[ebp-30h]                                                                                        ;
local_file_path variable
00452d93 50              push    eax                                                                                                  ;
local_file_path parameter
00452d94 e8c7e20100      call    NitroPDF!CxIOFile::Write+0x97a50 (00471060)                                                          ;
[16] generates temporary filename
00452d99 83c404          add     esp,4
...
NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x247655:
00792025 8b542408        mov     edx,dword ptr [esp+8]
00792029 8d420c          lea     eax,[edx+0Ch]
0079202c 8b8a70ffffff    mov     ecx,dword ptr [edx-90h]
00792032 33c8            xor     ecx,eax
00792034 e8ba9be4ff      call    NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x91223 (005dbbf3)
00792039 8b4af8          mov     ecx,dword ptr [edx-8]
0079203c 33c8            xor     ecx,eax
0079203e e8b09be4ff      call    NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x91223 (005dbbf3)
00792043 b8fc458d00      mov     eax,offset NitroPDF!CxImageJPG::`vftable'+0xeac54 (008d45fc)                                         ;
[14] exception handler
00792048 e92afbfdff      jmp     NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x2271a7 (00771b77)              ;
[14] __CxxFrameHandler3
```

The following function's responsibility is to call the constructor for a `local_file_path` object and initialize t with a temporary filename that resides in the user's temporary directory. Once initializing the stack frame, at [17] the function will use the `nitro::create_temp_directory` function to initialize a `boost::filesystem::path` object with the temporary directory name. Afterwards, the application intends to convert this path into a wide-character string so that it can be used to initialize a `local_file_path` object. Prior to this, the application will zero out a wide-character string buffer on the stack. This is done by passing the address of the string and the length of `+0x208` as parameters to a call to `memset` at [18].

```
NitroPDF!CxIOFile::Write+0x97a50:
00471060 55              push    ebp
00471061 8bec            mov     ebp,esp
00471063 6aff            push    0FFFFFFFFh
00471065 68683d7900      push    offset NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x249398 (00793d68)
0047106a 64a100000000    mov     eax,dword ptr fs:[00000000h]
00471070 50              push    eax
00471071 81ec7c020000    sub     esp,27Ch
...
NitroPDF!CxIOFile::Write+0x97b28:
00471138 8d85b8fdffff    lea     eax,[ebp-248h]
0047113e c745fc02000000  mov     dword ptr [ebp-4],2
00471145 50              push    eax
00471146 8d85a0fdffff    lea     eax,[ebp-260h]                                                                                       ;
boost::filesystem::path
0047114c 50              push    eax                                                                                                  ; path parameter
0047114d e80b542200      call    NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x14bb8d (0069655d)              ; [17]
nitro::create_temp_directory
00471152 83c408          add     esp,8
...
00471155 8d8db8fdffff    lea     ecx,[ebp-248h]
0047115b e880b4f5ff      call    NitroPDF+0x5c5e0 (003cc5e0)
00471160 6808020000      push    208h                                                                                                 ; length
00471165 33ff            xor     edi,edi
00471167 8d85e8fdffff    lea     eax,[ebp-218h]                                                                                       ; wchar_t
filename path
0047116d 57              push    edi
0047116e 50              push    eax                                                                                                  ; destination
0047116f e8330a3000      call    NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x2271d7 (00771ba7)              ; [18] memset
00471174 83c40c          add     esp,0Ch
```

After zeroing out the stack buffer that will contain the generated path, at [19] the function will construct a `CString` type, and at [20] will use it to load a string from the resource section of the application using the `CString::LoadStringW` method. This will load the string resource with the identifier 53 which has the value "%s\Untitled%d.pdf". Afterwards at [21], the temporary directory will be converted from a `boost::filesystem::path` to a wide-character string. After building all of the components of the format string, at [22] a call to `vswprintf_s` will render the full path to the temporary filename and write it into a buffer allocated on the stack. At [23], this temporary filename will now be used to construct a `boost::filesystem::path` using the address that was passed as this function's parameter and then return. Upon returning, the application will then use the `local_file_path::SetName` method to assign the temporary filename into a `local_file_path` object.

```
NitroPDF!CxIOFile::Write+0x97b70:
00471180 8d8d78fdffff    lea     ecx,[ebp-288h]                                                                    ;
contains format string
00471186 47              inc     edi
00471187 ff15acbf7b00    call    dword ptr [NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x2715dc (007bbfac)]    ;
[19] constructor
...
0047118d 6a35            push    35h                                                                               ;
0x35
0047118f 8d8d78fdffff    lea     ecx,[ebp-288h]                                                                    ;
format string
00471195 c645fc05        mov     byte ptr [ebp-4],5
00471199 ff15c0bf7b00    call    dword ptr [NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x2715f0 (007bbfc0)]    ;
[20] LoadStringW
...
0047119f 83bdb4fdffff08  cmp     dword ptr [ebp-24Ch],8
004711a6 8db5a0fdffff    lea     esi,[ebp-260h]
004711ac 8d8d78fdffff    lea     ecx,[ebp-288h]
004711b2 0f43b5a0fdffff  cmovae  esi,dword ptr [ebp-260h]
004711b9 ff159cbf7b00    call    dword ptr [NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x2715cc (007bbf9c)]    ;
[21] convert temporary directory
...
004711bf 57              push    edi
004711c0 56              push    esi                                                                               ;
format arguments
004711c1 50              push    eax                                                                               ;
format string
004711c2 8d85e8fdffff    lea     eax,[ebp-218h]                                                                    ;
result
004711c8 6804010000      push    104h                                                                              ;
buffer size
004711cd 50              push    eax                                                                               ;
buffer
004711ce e82dc6ffff      call    NitroPDF!CxIOFile::Write+0x941f0 (0046d800)                                       ;
[22] vswprintf_s
004711d3 83c414          add     esp,14h
...
NitroPDF!CxIOFile::Write+0x97bf0:
00471200 8d85e8fdffff    lea     eax,[ebp-218h]                                                                    ;
temporary filename
00471206 8bcb            mov     ecx,ebx                                                                           ;
from parameter
00471208 50              push    eax                                                                               ;
string
00471209 ff156cca7b00    call    dword ptr [NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x27209c (007bca6c)]    ;
[23] write new boost::filesystem::path containing temporary filename
```

After the temporary path was assigned to the `local_file_path` object via its `local_file_path::SetName` method, the following code will serialize the document and save it to disk. At [24], both the `local_file_path` object and the `pd_doc` object that were passed as parameters are used with the `npdf.dll!PDDocSave` function. This will save the newly constructed document with the temporary filename that was generated. After saving the document, the function will re-fetch the filename for the document using the `npdf.dll!PDDocGetFile` function. After retrieving the filename, at [25] the function will verify that the global `CNxDocManager` object is initialized. If this global is initialized, then the function call at [26] will store references to both the `local_file_path` and `pd_doc` objects at offset `+0x38` of the `CNxDocManager` object for the `local_file_path`, and offset `+0x40` for the `pd_doc` object. As these properties are not locked or kept track of in any form, these are duplicate references of both the `local_file_path` and `pd_doc` objects. These duplicate references directly relevant to the vulnerability described by this document as if they go out of scope, the references will not be updated. After storing both properties, the function will return and then convert the path back into a wide-character string. Afterwards the global `CNxAvApp` variable will be used to call the `CWinApp::OpenDocument` method at [27].

```
NitroPDF!CxIOFile::Write+0x797c8:
00452dd8 6a00            push    0                                                                                    ;
ASProgressMonitorBase*
00452dda 8d45d0          lea     eax,[ebp-30h]                                                                        ;
local_file_path
00452ddd 50              push    eax                                                                                  ;
[24] nitro::as_layer::file_path
00452dde 0fb705b8459400  movzx   eax,word ptr [NitroPDF!CxImageJPG::`vftable'+0x15ac10 (009445b8)]                    ;
flags
00452de5 50              push    eax                                                                                  ;
PDSaveFlags
00452de6 57              push    edi                                                                                  ;
pd_doc*
00452de7 ff1590ca7b00    call    dword ptr [NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x2720c0 (007bca90)]    ;
[24] call npdf.dll!PDDocSave
00452ded 83c410          add     esp,10h
...
NitroPDF!CxIOFile::Write+0x79827:
00452e37 8b0d2c799600    mov     ecx,dword ptr [NitroPDF!CxImageJPG::`vftable'+0x17df84 (0096792c)]                   ;
[25] CNxDocManager
00452e3d 85c9            test    ecx,ecx
00452e3f 0f84ca000000    je      NitroPDF!CxIOFile::Write+0x798ff (00452f0f)
00452e45 57              push    edi                                                                                  ;
pd_doc*
00452e46 56              push    esi                                                                                  ;
local_file_path
00452e47 e854190200      call    NitroPDF!CxIOFile::Write+0x9b190 (004747a0)                                          ;
[26] \ save path and pd_doc
\
NitroPDF!CxIOFile::Write+0x9b190:
004747a0 55              push    ebp
004747a1 8bec            mov     ebp,esp
004747a3 8b4508          mov     eax,dword ptr [ebp+8]                                                                ;
path name
004747a6 894138          mov     dword ptr [ecx+38h],eax                                                              ;
[26] local file path
004747a9 8b450c          mov     eax,dword ptr [ebp+0Ch]                                                              ;
pd_doc
004747ac c6413c00        mov     byte ptr [ecx+3Ch],0                                                                 ;
004747b0 894140          mov     dword ptr [ecx+40h],eax                                                              ;
[26] pd_doc
004747b3 5d              pop     ebp
004747b4 c20800          ret     8
/
00452e4c 6a00            push    0
00452e4e 56              push    esi
00452e4f 8d4d90          lea     ecx,[ebp-70h]
00452e52 e8d9820800      call    NitroPDF!CxIOFile::Write+0x101b20 (004db130)                                         ;
calls CAPPathName::SetPath
...
NitroPDF!CxIOFile::Write+0x79880:
00452e90 50              push    eax                                                                                  ;
[27] filename
00452e91 b9d0789600      mov     ecx,offset NitroPDF!CxImageJPG::`vftable'+0x17df28 (009678d0)                        ;
CNxAvApp
00452e96 e8dde13100      call    NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x2266a8 (00771078)    ;
[27] CWinApp::OpenDocumentFile
```

When calling the `CWinApp::OpenDocumentFile` function, the application will dispatch through the implementation of the `CNxAVApp::OpenDocumentFile` method and eventually call the `CNxDocManager::OpenDocumentFile` method that is listed in the following code. This method will first assign the filename received in its parameters to a variable on the stack and then check to ensure that that it is non-null. Due to the parent document being corrupted, the function call at [28] will fail resulting in an exception being explicitly raised by the call at [29]. This exception will get caught by the `CNxAVApp::OpenDocumentFile` method, but will continue to chain back to the exception handler that was created earlier in the `AVDocOpenFromPDDocWithParams` function.

```
NitroPDF!CxIOFile::Write+0x9a1c0:
004737d0 55              push    ebp
004737d1 8bec            mov     ebp,esp
004737d3 6aff            push    0FFFFFFFFh
004737d5 689d427900      push    offset NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x2498cd (0079429d)
004737da 64a100000000    mov     eax,dword ptr fs:[00000000h]
004737e0 50              push    eax
004737e1 81ec6c030000    sub     esp,36Ch
...
00473801 8bc1            mov     eax,ecx
00473803 8985ccfcffff    mov     dword ptr [ebp-334h],eax                                                                 ; [28] this
...
NitroPDF!CxIOFile::Write+0x9a1f9:
00473809 8b7d08          mov     edi,dword ptr [ebp+8]                                                                    ; [28] filename
from parameters
0047380c 8b4d0c          mov     ecx,dword ptr [ebp+0Ch]
0047380f 8985a4fcffff    mov     dword ptr [ebp-35Ch],eax
00473815 89bdc0fcffff    mov     dword ptr [ebp-340h],edi                                                                 ; [28] store
filename from parameters
0047381b 898db8fcffff    mov     dword ptr [ebp-348h],ecx
00473821 c785b0fcffff00000000 mov dword ptr [ebp-350h],0
0047382b 85ff            test    edi,edi
0047382d 0f84bb0e0000    je      NitroPDF!CxIOFile::Write+0x9b0de (004746ee)
00473833 66833f00        cmp     word ptr [edi],0
00473837 0f84b10e0000    je      NitroPDF!CxIOFile::Write+0x9b0de (004746ee)
...
0047383d 8985bcfcffff    mov     dword ptr [ebp-344h],eax                                                                 ; this
00473843 83c044          add     eax,44h
00473846 50              push    eax                                                                                      ; CNxDocManager
mutex
00473847 89859cfcffff    mov     dword ptr [ebp-364h],eax                                                                 ; mutex
0047384d e82ddf2f00      call    NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x226daf (0077177f)    ; [29] _Mtx_lock
00473852 83c404          add     esp,4
00473855 85c0            test    eax,eax
00473857 7409            je      NitroPDF!CxIOFile::Write+0x9a252 (00473862)
00473859 50              push    eax
0047385a e856df2f00      call    NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x226de5 (007717b5)    ; [30] raise a C
exception
```

Returning back to the `AVDocOpenFromPDDocWithParams` function, its structured exception handler will execute the following code when an exception was raised. After checking that both of the stack canaries were appended to the stack frame, at [31] the `__CxxFrameHandler3` function will be executed in order to process the exceptions registered by Microsoft's C++ exception handling implementation. This function call will then execute the two handlers that were implemented for the caught exception. One of the exception handlers that were registered is responsible for destroying the `local_file_path` object. At [32], this handler will take the `local_file_path` object and pass it to the `npdf.dll!ASFileSysReleasePath` function in order to destroy the object. It is prudent to note that this object was stored directly to a property belonging to the global `CNxDocManager` object. This destruction thus invalidates the scope of the `local_file_path` that was previously stored within the global `CxDocManager` object.

```
NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x247655:
00792025 8b542408        mov     edx,dword ptr [esp+8]
00792029 8d420c          lea     eax,[edx+0Ch]
0079202c 8b8a70ffffff    mov     ecx,dword ptr [edx-90h]
00792032 33c8            xor     ecx,eax
00792034 e8ba9be4ff      call    NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x91223 (005dbbf3)                ;
check canary
00792039 8b4af8          mov     ecx,dword ptr [edx-8]
0079203c 33c8            xor     ecx,eax
0079203e e8b09be4ff      call    NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x91223 (005dbbf3)                ;
check canary
00792043 b8fc458d00      mov     eax,offset NitroPDF!CxImageJPG::`vftable'+0xeac54 (008d45fc)
00792048 e92afbfdff      jmp     NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x2271a7 (00771b77)               ;
[31] __CxxFrameHandler3
...
NitroPDF!CxIOFile::Write+0x79932:
00452f42 8b4584          mov     eax,dword ptr [ebp-7Ch]                                                                              ;
local_file_path
00452f45 85c0            test    eax,eax
00452f47 740a            je      NitroPDF!CxIOFile::Write+0x79943 (00452f53)
00452f49 50              push    eax
00452f4a ff15bcca7b00    call    dword ptr [NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x2720ec (007bcabc)]    ;
[32] npdf.dll!ASFileSysReleasePath
00452f50 83c404          add     esp,4
```

One place that the two stored properties of the `CNxDocManager` are used are in the implementation of the `CNxDocManager::OpenDocumentFile` method. This method will be used the next time another document is opened. In the following code from the `CNxDocManager::OpenDocumentFile` method, the `CNxDocManager` is fetched from the `%esi` register and is then used to call the method at [33]. In this method, a global will be checked and then at [34] the destroyed `local_file_path` will be fetched from the `CNxDocManager`. At [35], the virtual method table of the released object will be dereferenced, and finally at [36] the `local_file_path::get_file_system` method will be called from the dereferenced virtual method table. As the object has been previously destroyed, if another allocation has occurred and reoccupies the destroyed object's space, this dereference can load user-controlled data which can allow for code execution under the context of the application.

```
NitroPDF!CxIOFile::Write+0x9aeea:
004744fa 0f94c0          sete    al
004744fd 8bce            mov     ecx,esi                                         ; CNxDocManager (this)
004744ff 0fb6c0          movzx   eax,al
00474502 50              push    eax
00474503 e888effffff     call    NitroPDF!CxIOFile::Write+0x99e80 (00473490)    ; [33] \
\
NitroPDF!CxIOFile::Write+0x99e80:
00473490 55              push    ebp
00473491 8bec            mov     ebp,esp
00473493 83ec08          sub     esp,8
00473496 833ddc80960000  cmp     dword ptr [NitroPDF!CxImageJPG::`vftable'+0x17e734 (009680dc)],0
0047349d 57              push    edi
0047349e 8bf9            mov     edi,ecx
004734a0 7448            je      NitroPDF!CxIOFile::Write+0x99eda (004734ea)
004734a2 8b4f38          mov     ecx,dword ptr [edi+38h]                         ; [34] local file path
004734a5 85c9            test    ecx,ecx
004734a7 7441            je      NitroPDF!CxIOFile::Write+0x99eda (004734ea)
004734a9 8b01            mov     eax,dword ptr [ecx]                             ; [35] dereference virtual method table
004734ab 56              push    esi
004734ac ff5004          call    dword ptr [eax+4]                               ; [36] call local_file_path::get_file_system
```

The module addresses of the disassembly used in this advisory have the following base addresses.

```
start    end        module name
00370000 00dd6000   NitroPDF   (export symbols)       NitroPDF.exe
2bd80000 2bf97000   np_java_script   (deferred)
53370000 53e7c000   npdf       (deferred)
```

Crash Information

Run the application with a debugger, and set the following breakpoint. This breakpoint is set at the address of the binding for the "app.newDoc" javascript function. Resume execution of the application, and then open up the provided proof-of-concept.

```
0:000> bp np_java_script+7850
0:000> g
```

Upon the debugger returning control, the application will be at the address of the "app.newDoc" javascript function.

```
Breakpoint 0 hit
eax=0201d53c ebx=30a92690 ecx=30991014 edx=2ce67850 esi=30991094 edi=306f0e88
eip=2ce67850 esp=0201d4e0 ebp=0201d598 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
np_java_script+0x7850:
2ce67850 55              push    ebp
```

To resume execution, set a breakpoint at the specified address and continue execution. When the debugger breaks, the application should be at the `np_java_script.dll!AppNewDoc` function which takes a width and a height as its parameters. If we dump out the values on the stack, the width and height are both set to `0x100`. This function will use the `npdf.dll!PDDocCreate` function to create a new document, create a page, and then generate a temporary file name for the resulting document. When creating the document, the application will dispatch into the HFT Extension Manager which will eventually be used to generate a temporary filename. Once generating the temporary filename, its string will be passed to a constructor for a `local_file_path` object.

```
0:000> bp np_java_script+46b0
0:000> g

Breakpoint 1 hit
eax=00000100 ebx=00000100 ecx=1e637698 edx=0201d4d8 esi=2d0942f0 edi=306f0e88
eip=2ce646b0 esp=0201d4b0 ebp=0201d4dc iopl=0         nv up ei pl nz na pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000207
np_java_script+0x46b0:
2ce646b0 55              push    ebp

0:000> dc @esp+4 L3
0201d4b4  30adcfd0 00000100 00000100           ...0........
```

Next, we will set a breakpoint at the point of the application where the temporary filename has been generated, and is being passed as a parameter to the constructor for the `local_file_path` object. Set the specified breakpoint, and resume execution. When the debugger stops, the current program counter will be at the address of the described constructor. If we print the address of its single parameter, and then dump out the string it points to, we will see the temporary filename of the new document that is being created.

```
0:000> bp NitroPDF+101209
0:000> g

Breakpoint 2 hit
eax=0201d134 ebx=0201d3c4 ecx=0201d3c4 edx=0201d00c esi=41340fa0 edi=00000001
eip=00471209 esp=0201d0b0 ebp=0201d34c iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
NitroPDF!CxIOFile::Write+0x97bf9:
00471209 ff156cca7b00    call    dword ptr [NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x27209c (007bca6c)]
ds:0023:007bca6c={npdf!local_file_path::local_file_path (5346d0e0)}

0:000> dc @esp L1
0201d0b0  0201d134                             4...

0:000> du @$p
0201d134  "C:\Users\user\AppData\Local\Temp"
0201d174  "\NitroPDF\2324\Untitled1.pdf"
```

Next we will set a breakpoint at a key point of this vulnerability where the address of the `local_file_path` object that we created is passed by reference to a function which assigns the object into the global `CNxDocManager` object. This will duplicate the reference of the `local_file_path` object without the `CNxDocManager` object knowing when it will go out of scope. To examine this, set a breakpoint at the specified address, and resume execution. Upon returning to the debugger, the `local_file_path` object is located at the address specified by the `%eax` register. If we dump the values at its address, the virtual method table will be the first address, and the string that it contains is at the second address. In the example that follows, the address of the `local_file_path` is at `0x28d41fe0`. If we examine the address that our object will be written by using the sum of the `%ecx` register and the offset `0x38`, we can see where our `local_file_path` object will get stored which in this example will be `0x29644fb8`. We will now set a breakpoint at the destructor for the object, and continue execution.

```
0:000> bp NitroPDF+1047a6
0:000> g

Breakpoint 3 hit
eax=28d41fe0 ebx=00000000 ecx=29644f80 edx=02000000 esi=28d41fe0 edi=1eb1aef0
eip=004747a6 esp=0201d348 ebp=0201d348 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
NitroPDF!CxIOFile::Write+0x9b196:
004747a6 894138          mov     dword ptr [ecx+38h],eax ds:0023:29644fb8=1dd1afe0

0:000> r @eax,@ecx
eax=28d41fe0 ecx=29644f80

0:000> dc @eax L8
28d41fe0  539eeacc 3eb46f80 c0c0c0c0 c0c0c0c0  ...S.o.>........
28d41ff0  c0c0c0c0 0000003c 0000003f d0d0d0d0  ....<...?.......

0:000> dds @$p L0x10
539eeacc  5346b180 npdf!local_file_path::operator=+0x170
539eead0  5346fb20 npdf!local_file_path::get_file_system
539eead4  5346f720 npdf!local_file_path::clone
539eead8  53470120 npdf!local_file_path::open
539eeadc  5346fdb0 npdf!local_file_path::is_same_as
539eeae0  5346fb30 npdf!local_file_path::get_modification_date
539eeae4  5346fa00 npdf!local_file_path::get_creation_date
539eeae8  5346fd40 npdf!local_file_path::is_directory
539eeaec  5346df20 npdf!local_file_path::exists
539eeaf0  5346fab0 npdf!local_file_path::get_file_size_bytes
539eeaf4  53470640 npdf!local_file_path::remove
539eeaf8  5346b600 npdf!nitro::as_layer::file_path::legacy_display_string
539eeafc  5346b620 npdf!nitro::as_layer::file_path::legacy_file_name
539eeb00  5346f6e0 npdf!local_file_path::base_name
539eeb04  53471540 npdf!local_file_path::user_friendly_string
539eeb08  5346b600 npdf!nitro::as_layer::file_path::legacy_display_string

0:000> du poi(@eax+4)
3eb46f80  "C:\Users\user\AppData\Local\Temp"
3eb46fc0  "\NitroPDF\2324\Untitled1.pdf"

0:000> dc @ecx+38 29644fb8 L0x11
29644fb8  28d41fe0 c0c0c000 1eb1aef0 00000002  ...(............
29644fc8  6c586048 00000001 c0c0c0c0 c0c0c0c0  H`Xl............
29644fd8  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0  ................
29644fe8  c0c0c0c0 000000c4 00000001 19839fd8  ................
29644ff8  1983fff0

0:000> r@$t1=@ecx+38
0:000> bp NitroPDF+e2f4a
0:000> g
```

Upon the application arriving at the next breakpoint, we should be at the destructor for the `local_file_path`. If we dump out the first parameter for this call, we can see that it corresponds to the address that was assigned into the `CNxDocManager` object previously. Just to verify, we can dump out the property belonging to the `CNxDocManager` which was at `0x29644fb8`. In this example, it was at the address `0x28d41fe0`. If we step over the execution of this function, the object will be released. Just to verify the reference is still there, we can dump out the address containing the reference again, followed by the address that was just released. If we are using the “Full Page Heap” Global Flags debugging option, the address will be completely released.

```
Breakpoint 4 hit
eax=28d41fe0 ebx=0201d3e8 ecx=00000100 edx=00000000 esi=0201c9ec edi=0201cbd4
eip=00452f4a esp=0201c898 ebp=0201d3f4 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
NitroPDF!CxIOFile::Write+0x7993a:
00452f4a ff15bcca7b00    call    dword ptr [NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+0x2720ec (007bcabc)]
ds:0023:007bcabc={npdf!ASStmClose (5346c330)}

0:000> dc @esp L1
0201c898  28d41fe0                              ...(

0:000> dc @$t1 L4
29644fb8  28d41fe0 c0c0c000 1eb1aef0 00000002

0:000> p
eax=28d41fe0 ebx=0201d3e8 ecx=5245ed55 edx=02000000 esi=0201c9ec edi=0201cbd4
eip=00452f50 esp=0201c898 ebp=0201d3f4 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
NitroPDF!CxIOFile::Write+0x79940:
00452f50 83c404          add     esp,4

0:000> dc @$t1 L4
29644fb8  28d41fe0 c0c0c000 1eb1aef0 00000002  ...(............

0:000> dc @$p L1
28d41fe0  ????????                              ????
```

As the `local_file_path` object in question was released, any instruction that accesses the address to it stored in the `CNxDocManager` can be considered a use-after-free vulnerability. To catch the next access to this released object, we can use a hardware breakpoint on the address containing our reference. In this example, the address inside the global `CNxDocManager` is at `0x29644fb8`. Set a breakpoint-on-access when reading from the given address, and then resume execution. There is a chance that a message box will be displayed in the application during this time due to the corruption of the proof-of-concept. Go to the application, click “Ok”, then wait for the debugger to dereference the address that is being watched.

```
0:000> ba r4 @$t1
0:000> g
```

Upon the debugger encountering the breakpoint, and then disassembling backwards, we can see that the application has read an address from the `%edi` register into the `%ecx` register. Printing their values shows that the `%edi` register was pointing to our address inside the global `CNxDocManager` object, and the `%ecx` register contains the address of the `local_file_path` object that was released. If we disassemble forwards, the application will load the address of the virtual method table for the released object into the `%eax` register, and then call a method from it. As the object has gone out of scope due to being released, this can leveraged by an attacker in order to earn code execution under the context of the application.

```
Breakpoint 5 hit
eax=00000000 ebx=00000000 ecx=28d41fe0 edx=02000000 esi=29644f80 edi=29644f80
eip=004734a5 esp=0201ed44 ebp=0201ed50 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
NitroPDF!CxIOFile::Write+0x99e95:
004734a5 85c9            test    ecx,ecx

0:000> ub . L1
NitroPDF!CxIOFile::Write+0x99e92:
004734a2 8b4f38          mov     ecx,dword ptr [edi+38h]

0:000> r @edi,@ecx
edi=29644f80 ecx=28d41fe0

0:000> u . L5
NitroPDF!CxIOFile::Write+0x99e95:
004734a5 85c9            test    ecx,ecx
004734a7 7441            je      NitroPDF!CxIOFile::Write+0x99eda (004734ea)
004734a9 8b01            mov     eax,dword ptr [ecx]
004734ab 56              push    esi
004734ac ff5004          call    dword ptr [eax+4]
```

We can now resume execution to let it crash while trying to dereference the released address.

```
0:000> g
(914.c4): 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=28d41fe0 edx=02000000 esi=29644f80 edi=29644f80
eip=004734a9 esp=0201ed44 ebp=0201ed50 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
NitroPDF!CxIOFile::Write+0x99e99:
004734a9 8b01            mov     eax,dword ptr [ecx]  ds:0023:28d41fe0=????????
```

In this example, the modules that are referred to are at the following base addresses.

```
Browse full module list
start    end        module name
00370000 00dd6000   NitroPDF   (export symbols)       C:\Program Files\Nitro\Pro\13\NitroPDF.exe
53370000 53e7c000   npdf       (export symbols)       C:\Program Files\Nitro\Pro\13\npdf.dll
2ce60000 2d077000   np_java_script   (export symbols)       C:\Program Files\Nitro\Pro\13\np_java_script.dll
```

Exploit Proof of Concept

In the provided proof-of-concept, it is object 110 revision 0 that contains the javascript which triggers this vulnerability.

Mitigation

To mitigate this vulnerability, one can visit the “JavaScript” item in the preferences and click the “Disable JavaScript” checkbox. As this vulnerability depends on the execution of JavaScript, enabling this option will prevent the vulnerability from being triggered.

Timeline

2021-03-15 - Vendor Disclosure
2021-04-20 - 30 day follow up with vendor
2021-06-08 - Copies of advisories issued
2021-06-22 - Granted disclosure extension
2021-07-19 - Final disclosure extension granted
2021-10-14 - Public Release

CREDIT

Discovered by a member of Cisco Talos.
