Talos Vulnerability Report

TALOS-2021-1267

Nitro Pro PDF JavaScript document.flattenPages JSStackFrame stack-based use-after-free vulnerability

SEPTEMBER 13, 2021

CVE NUMBER

CVE-2021-21798

Summary

An exploitable return of stack variable address vulnerability exists in the JavaScript implementation of Nitro Pro PDF. A specially crafted document can cause a stack variable to go out of scope, resulting in the application dereferencing a stale pointer. This can lead to code execution under the context of the application. An attacker can convince a user to open a document to trigger the 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-562 - Return of Stack Variable Address

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 an `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 `HFTServer` 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 javascript classes available to a user. 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 for the object, there's a number of function calls used to initialize all of the available javascript objects. The function call at [2] is then used to define the "Document" javascript class. At [3], the global `JSClass` definition for a "Document" is passed as an argument to the `js32u.dll!JS_InitClass` function in order to register the class with the SpiderMonkey library. Afterwards, the resulting object will be stored to a global and also appended to a global array.

```
np_java_script+0x511f0:
2beb11f0 55              push    ebp
2beb11f1 8bec            mov     ebp,esp
2beb11f3 83ec40          sub     esp,40h
2beb11f6 a17441fc2b      mov     eax,dword ptr [np_java_script!CxImageJPG::`vftable'+0x4f8b8 (2bfc4174)]
2beb11fb 33c5            xor     eax,ebp
2beb11fd 8945fc          mov     dword ptr [ebp-4],eax
2beb1200 53              push    ebx
2beb1201 8b5d08          mov     ebx,dword ptr [ebp+8]
2beb1204 894dc4          mov     dword ptr [ebp-3Ch],ecx                                 ; this
...
2beb120d e8de9e0000      call    np_java_script!nitro::java_script::create_plugin+0x9890 (2bebb0f0)
2beb1212 8bcb            mov     ecx,ebx
2beb1214 a3dce7fc2b      mov     dword ptr [np_java_script!CxImageJPG::`vftable'+0x59f20 (2bfce7dc)],eax    ; [1] parent plugin object
containing global state
2beb1219 e8c29e0000      call    np_java_script!nitro::java_script::create_plugin+0x9880 (2bebb0e0)
2beb121e 8bcb            mov     ecx,ebx
2beb1220 a3d8e7fc2b      mov     dword ptr [np_java_script!CxImageJPG::`vftable'+0x59f1c (2bfce7d8)],eax    ; root JSContext
2beb1225 e8d69e0000      call    np_java_script!nitro::java_script::create_plugin+0x98a0 (2bebb100)
2beb122a 8bcb            mov     ecx,ebx
2beb122c a3e4e7fc2b      mov     dword ptr [np_java_script!CxImageJPG::`vftable'+0x59f28 (2bfce7e4)],eax    ; pdf_java_script_actions
2beb1231 e8da9e0000      call    np_java_script!nitro::java_script::create_plugin+0x98b0 (2bebb110)
2beb1236 a3e0e7fc2b      mov     dword ptr [np_java_script!CxImageJPG::`vftable'+0x59f24 (2bfce7e0)],eax    ; array of objects
2beb123b e88059fbff      call    np_java_script+0x6bc0 (2be66bc0)                        ; define app object and attach
its private date
2beb1240 85c0            test    eax,eax
2beb1242 0f8450020000    je      np_java_script+0x51498 (2beb1498)
...
2beb1255 e8e639fcff      call    np_java_script+0x14c40 (2be74c40)                       ; sealed "console" object
2beb125a 85c0            test    eax,eax
2beb125c 0f8436020000    je      np_java_script+0x51498 (2beb1498)
...
2beb1262 e8a943fdff      call    np_java_script+0x25610 (2be85610)                       ; [2] \ create "Document" object
2beb1267 85c0            test    eax,eax
2beb1269 0f8429020000    je      np_java_script+0x51498 (2beb1498)
\
np_java_script+0x25610:
2be85610 6a00            push    0                                                       ; (static_fs)
2be85612 6a00            push    0                                                       ; (static_ps)
2be85614 6a00            push    0                                                       ; (fs)
2be85616 6a00            push    0                                                       ; (ps)
2be85618 6a00            push    0                                                       ; (nargs)
2be8561a 6a00            push    0                                                       ; (constructor)
2be8561c 68701afc2b      push    offset np_java_script!CxImageJPG::`vftable'+0x4d1b4 (2bfc1a70)          ; (clasp) "Document" JSClass
2be85621 6a00            push    0                                                       ; (parent_proto)
2be85623 ff35dce7fc2b    push    dword ptr [np_java_script!CxImageJPG::`vftable'+0x59f20 (2bfce7dc)]     ; (parent obj
2be85629 ff35d8e7fc2b    push    dword ptr [np_java_script!CxImageJPG::`vftable'+0x59f1c (2bfce7d8)]     ; (cx) root JSContext
2be8562f ff1568e7f52b    call    dword ptr [np_java_script!CxImageJPG::Encode+0x9c7e8 (2bf5e768)]        ; [3] JS_InitClass
2be85635 83c428          add     esp,28h
2be85638 a340e3fc2b      mov     dword ptr [np_java_script!CxImageJPG::`vftable'+0x59a84 (2bfce340)],eax    ; store "Document" to global
2be8563d 85c0            test    eax,eax
2be8563f 7501            jne     np_java_script+0x25642 (2be85642)
```

After registering the “Document” javascript class, the following function named `JSDocumentConstructObject` will be used by the application to create an instance of “Document” javascript object. This starts with the call to the `js32u.dll!JS_NewObject` function at [4], which will be used with the global `JSClass` definition for a “Document” object to create a new instance of the object. Afterwards at [5], the `JSDocumentConstructObject` function will then register an array of `JSPropertySpec` definitions for the object using the `js32u.dll!JS_DefineProperties` function. At [6], the function will then do the same type of registration with an array of `JSFunctionSpec` definitions using the `js32u.dll!JS_DefineFunctions` function. After the registration of the “Document” object’s properties and functions, accessing any of the defined properties from javascript or calling any of its defined functions will result in the relevant code being executed.

```
np_java_script+0x24be0:
2be84be0 55              push    ebp
2be84be1 8bec            mov     ebp,esp
2be84be3 51              push    ecx
2be84be4 56              push    esi
2be84be5 57              push    edi
2be84be6 6a00            push    0
2be84be8 ff35dce7fc2b    push    dword ptr [np_java_script!CxImageJPG::`vftable'+0x59f20 (2bfce7dc)]    ; parent "app" object
2be84bee 68701afc2b      push    offset np_java_script!CxImageJPG::`vftable'+0x4d1b4 (2bfc1a70)        ; "Document JSClass
2be84bf3 ff35d8e7fc2b    push    dword ptr [np_java_script!CxImageJPG::`vftable'+0x59f1c (2bfce7d8)]    ; root JSContext
2be84bf9 ff1510e7f52b    call    dword ptr [np_java_script!CxImageJPG::Encode+0x9c790 (2bf5e710)]      ; [4] JS_NewObject
2be84bff 8bf0            mov     esi,eax
2be84c01 83c410          add     esp,10h
2be84c04 85f6            test    esi,esi
2be84c06 7520            jne     np_java_script+0x24c28 (2be84c28)
...
np_java_script+0x24c28:
2be84c28 688012fc2b      push    offset np_java_script!CxImageJPG::`vftable'+0x4c9c4 (2bfc1280)        ; "Document" properties
2be84c2d 56              push    esi                                                     ; object
2be84c2e ff35d8e7fc2b    push    dword ptr [np_java_script!CxImageJPG::`vftable'+0x59f1c (2bfce7d8)]    ; root JSContext
2be84c34 ff1518e7f52b    call    dword ptr [np_java_script!CxImageJPG::Encode+0x9c798 (2bf5e718)]      ; [5] JS_DefineProperties
2be84c3a 680014fc2b      push    offset np_java_script!CxImageJPG::`vftable'+0x4cb44 (2bfc1400)        ; "Document" functions
2be84c3f 56              push    esi                                                     ; object
2be84c40 ff35d8e7fc2b    push    dword ptr [np_java_script!CxImageJPG::`vftable'+0x59f1c (2bfce7d8)]    ; root JSContext
2be84c46 ff1538e7f52b    call    dword ptr [np_java_script!CxImageJPG::Encode+0x9c7b8 (2bf5e738)]      ; [6] JS_DefineFunctions
```

Whenever the SpiderMonkey library needs to execute a registered javascript binding, the following `js32u.dll!js_Invoke` function will be used. This function is responsible for pushing a `JSStackFrame` structure onto the execution stack for the current `JSContext` and updating its state prior to invoking the desired binding. After the function has allocated space on the stack for its stack frame, the function will load its parameters into registers prior to determining how to invoke the javascript function that is being requested. Throughout this function at [8], the properties of the `JSStackFrame` structure on the stack will be initialized. During the initialization of the fields belonging to the `JSStackFrame`, at [9] the function will load the previous value of the `JSContext.fp` field into the `%eax` register so that it can be used to append the current `JSStackFrame` to the `JSContext` being used. After loading the previous value of the `JSContext.fp` field, the address of the `JSStackFrame` being initialized will be loaded into the `%ecx` register. Then at [10], this address in `%ecx` will be assigned to the `JSContext.fp` field thus switching the `JSContext` into using the new `JSStackFrame`. After updating the `JSContext.fp` field with the newly allocated and initialized `JSStackFrame`, the previous `JSStackFrame` for the `JSContext` that was loaded into the `%eax` register can then be written into the `JSStackFrame.down` field for the new `JSStackFrame`. This will append the new `JSStackFrame` to the linked list that is stored in the `JSContext.fp` field. It is prudent to note that the newly allocated `JSStackFrame` is allocated on the stack, thus if this function returns then this `JSStackFrame` will go out of scope and will be destroyed.

```
js32u!js_Invoke:
2c0c2f00 55              push    ebp
2c0c2f01 8bec            mov     ebp,esp
2c0c2f03 81ec94000000    sub     esp,94h                 ; allocate space on stack
2c0c2f09 53              push    ebx
2c0c2f0a 56              push    esi
2c0c2f0b 8b7510          mov     esi,dword ptr [ebp+10h]    ; vp
2c0c2f0e 57              push    edi
2c0c2f0f 8b7d08          mov     edi,dword ptr [ebp+8]      ; JSContext cx
...
js32u!js_Invoke+0x51e:
2c0c341e 8b4510          mov     eax,dword ptr [ebp+10h]    ; vp
2c0c3421 8b55e4          mov     edx,dword ptr [ebp-1Ch]    ; argv
2c0c3424 8955a0          mov     dword ptr [ebp-60h],edx    ; [7] frame.argv
2c0c3427 8b55f0          mov     edx,dword ptr [ebp-10h]    ; hookdata
2c0c342a f6c201          test    dl,1
2c0c342d 8b4804          mov     ecx,dword ptr [eax+4]
2c0c3430 8b45e8          mov     eax,dword ptr [ebp-18h]    ; script
2c0c3433 894590          mov     dword ptr [ebp-70h],eax    ; [8] frame.script
2c0c3436 8b45d4          mov     eax,dword ptr [ebp-2Ch]    ; funobject
2c0c3439 89458c          mov     dword ptr [ebp-74h],eax    ; [8] frame.callee
2c0c343c 8b450c          mov     eax,dword ptr [ebp+0Ch]    ; argc
2c0c343f 89459c          mov     dword ptr [ebp-64h],eax    ; [8] frame.argc
2c0c3442 b801000080      mov     eax,80000001h
2c0c3447 0f45c1          cmovne  eax,ecx
2c0c344a 894d98          mov     dword ptr [ebp-68h],ecx    ; [8] frame.thisp
2c0c344d 894df8          mov     dword ptr [ebp-8],ecx      ; nslots
2c0c3450 8b4df4          mov     ecx,dword ptr [ebp-0Ch]    ; funflags
2c0c3453 8945a4          mov     dword ptr [ebp-5Ch],eax    ; [8] frame.rval
2c0c3456 0bca            or      ecx,edx
2c0c3458 8b45ec          mov     eax,dword ptr [ebp-14h]    ; nvars
2c0c345b 8945a8          mov     dword ptr [ebp-58h],eax    ; [8] frame.nvars
2c0c345e c1e002          shl     eax,2
2c0c3461 2bf0            sub     esi,eax
2c0c3463 894dc4          mov     dword ptr [ebp-3Ch],ecx    ; frame.flags
2c0c3466 8b4754          mov     eax,dword ptr [edi+54h]    ; [9] read previous JSContext.fp
2c0c3469 8d8d78ffffff    lea     ecx,[ebp-88h]              ; fetch address of current frame
2c0c346f 894f54          mov     dword ptr [edi+54h],ecx    ; [10] write address of current frame to JSContext.fp
2c0c3472 8b8f70010000    mov     ecx,dword ptr [edi+170h]   ; JSContext.debugHooks
2c0c3478 8945b0          mov     dword ptr [ebp-50h],eax    ; [11] write previous JSContext.fp to frame.down
2c0c347b 33c0            xor     eax,eax
2c0c347d c7458800000000  mov     dword ptr [ebp-78h],0      ; [8] frame.varobj
2c0c3484 c7458400000000  mov     dword ptr [ebp-7Ch],0      ; [8] frame.argsobj
2c0c348b c7458000000000  mov     dword ptr [ebp-80h],0      ; [8] frame.callobj
2c0c3492 895d94          mov     dword ptr [ebp-6Ch],ebx    ; [8] frame.fun
2c0c3495 8975ac          mov     dword ptr [ebp-54h],esi    ; [8] frame.vars
2c0c3498 c745b400000000  mov     dword ptr [ebp-4Ch],0      ; [8] frame.annotation
2c0c349f 8945b8          mov     dword ptr [ebp-48h],eax    ; [8] frame.scopeChain
2c0c34a2 898578ffffff    mov     dword ptr [ebp-88h],eax    ; [8] frame.regs
2c0c34a8 89857cffffff    mov     dword ptr [ebp-84h],eax    ; [8] frame.spbase
2c0c34ae 8945bc          mov     dword ptr [ebp-44h],eax    ; [8] frame.sharpDepth
2c0c34b1 8945c0          mov     dword ptr [ebp-40h],eax    ; [8] frame.sharpArray
2c0c34b4 8945c8          mov     dword ptr [ebp-38h],eax    ; [8] frame.dormantNext
2c0c34b7 8945cc          mov     dword ptr [ebp-34h],eax    ; [8] frame.xmlNamespace
2c0c34ba 8945d0          mov     dword ptr [ebp-30h],eax    ; [8] frame.blockchain
2c0c34bd 8b5130          mov     edx,dword ptr [ecx+30h]    ; JSDebugHooks.callHook
```

After appending the `JSStackFrame` that was allocated on the stack to the linked list stored in the `JSContext.fp` field, the following code will be used to dispatch into the `JSNative` binding that the `js32u.dll!js_Invoke` function was used to invoke. After the function has prepared the current `JSContext` with the newly allocated frame to execute the binding with. The function will use an address of some free space on the stack for containing a result, a pointer to the binding's arguments, the count or number of arguments, the object that will be used, and the current `JSContext` as parameters to the binding at [12]. This will result in the `JSNative` binding being invoked to execute the binding using the new `JSStackFrame`, and write its result to the address that was passed in its parameters.

```
js32u!js_Invoke+0x618:
2c0c3518 85c0            test    eax,eax
2c0c351a 0f4445e0        cmove   eax,dword ptr [ebp-20h]            ; parent scope
2c0c351e 8945b8          mov     dword ptr [ebp-48h],eax            ; JSStackFrame.scopeChain
2c0c3521 8d45a4          lea     eax,[ebp-5Ch]                      ; frame.rval
2c0c3524 50              push    eax                                ; rval
2c0c3525 56              push    esi                                ; argv
2c0c3526 ff750c          push    dword ptr [ebp+0Ch]                ; argc
2c0c3529 ff75f8          push    dword ptr [ebp-8]                  ; obj
2c0c352c 57              push    edi                                ; (cx) current JSContext
2c0c352d ffd2            call    edx                                ; [12] dispatch to JSNative binding
2c0c352f 83c414          add     esp,14h
2c0c3532 eb3e            jmp     js32u!js_Invoke+0x672 (2c0c3572)
```

Due to the application registering a `JSFunctionSpec` for the "Document.flattenPages" javascript function, the following code will be executed when the "Document.flattenPages" function is invoked. This function is used to flatten all of the pages in the current document and heavily depends on the `npdf.dll` library to facilitate its logic. Prior to allocating the necessary space for its stack frame, at [13] the function will first register a structured exception handler. After checking the types of its arguments and extracting the start and end pages that were passed therein, the determined parameters will be passed to the closure at [14] in order to actually flatten the pages of the document. If an exception occurs while the "Document.flattenPages" function is being invoked, the exception handler at [15] will be executed. This exception handler will then dispatch to the C++ exception handlers at [16] in order to properly handle any exceptions that were raised by the implementation.

```
np_java_script+0x17ee0:
2be77ee0 55              push    ebp
2be77ee1 8bec            mov     ebp,esp
2be77ee3 6aff            push    0FFFFFFFFh
2be77ee5 681056f52b      push    offset np_java_script!CxImageJPG::Encode+0x93690 (2bf55610)       ; [13] address of structured exception
handler
2be77eea 64a100000000    mov     eax,dword ptr fs:[00000000h]
2be77ef0 50              push    eax
2be77ef1 83ec0c          sub     esp,0Ch
2be77ef4 53              push    ebx
2be77ef5 56              push    esi
2be77ef6 57              push    edi
2be77ef7 a17441fc2b      mov     eax,dword ptr [np_java_script!CxImageJPG::`vftable'+0x4f8b8 (2bfc4174)]
2be77efc 33c5            xor     eax,ebp
2be77efe 50              push    eax
2be77eff 8d45f4          lea     eax,[ebp-0Ch]
2be77f02 64a300000000    mov     dword ptr fs:[00000000h],eax
2be77f08 8965f0          mov     dword ptr [ebp-10h],esp
...
np_java_script+0x181ad:
2be781ad 8b7518          mov     esi,dword ptr [ebp+18h]
2be781b0 ff75e8          push    dword ptr [ebp-18h]                                        ; flags
2be781b3 53              push    ebx                                                        ; nEnd
2be781b4 56              push    esi                                                        ; nStart
2be781b5 57              push    edi                                                        ; private data
2be781b6 e8a5770000      call    np_java_script+0x1f960 (2be7f960)                          ; [14] flatten pages of document
2be781bb 83c410          add     esp,10h
2be781be 0fb6c0          movzx   eax,al
...
np_java_script!CxImageJPG::Encode+0x93690:
2bf55610 8b542408        mov     edx,dword ptr [esp+8]                                      ; [15] structured exception handler
2bf55614 8d420c          lea     eax,[edx+0Ch]
2bf55617 8b4ae4          mov     ecx,dword ptr [edx-1Ch]
2bf5561a 33c8            xor     ecx,eax
2bf5561c e85bd4f6ff      call    np_java_script!CxImageJPG::Encode+0xafc (2bec2a7c)         ; __security_check_cookie(x)
2bf55621 b8544cfa2b      mov     eax,offset np_java_script!CxImageJPG::`vftable'+0x30398 (2bfa4c54) ; [16] C++ Exception Handlers
2bf55626 e986b7ffff      jmp     np_java_script!CxImageJPG::Encode+0x8ee31 (2bf50db1)       ; __CxxFrameHandler3
```

If there is an error while flattening the pages of the document (which can occur if the annotations for the pages are referenced incorrectly), an exception will be raised which will result in the structured exception handler for the “Document.flattenPages” binding being executed. This will then proceed to execute the C++ exception handlers that were registered during compilation. The first C++ exception handler will execute the following code in which the function call at [17] will capture the exception that was raised and then proceed to convert it into a different exception with an error message describing the type of the exception that was caught. Upon that function returning, the new exception that was returned will then be raised at [18]. This exception being raised will then skip over the next exception handler that was defined within this function so that it can be caught by the next structured exception handler that was registered by another function. This raising of an exception that may be caught by other C++ exception handlers is directly relevant to the vulnerability described within this document.

```
np_java_script+0x18228:
2be78228 68f40e0000      push    0EF4h
2be7822d 685468f62b      push    offset np_java_script!CxImageJPG::Encode+0xa48d4 (2bf66854)
2be78232 e8290fffff      call    np_java_script+0x9160 (2be69160)                           ; [17] Convert caught exception
2be78237 83c408          add     esp,8
2be7823a 6a00            push    0
2be7823c 6a00            push    0
2be7823e e8688b0d00      call    np_java_script!CxImageJPG::Encode+0x8ee2b (2bf50dab)       ; [18] _CxxThrowException
...
np_java_script+0x1820a:
2be7820a 51              push    ecx
2be7820b 8bcc            mov     ecx,esp
2be7820d 682468f62b      push    offset np_java_script!CxImageJPG::Encode+0xa48a4 (2bf66824)        ; "document.flattenPages()"
2be78212 ff1550e9f52b    call    dword ptr [np_java_script!CxImageJPG::Encode+0x9c9d0 (2bf5e950)]   ; CString
2be78218 6a0b            push    0Bh
2be7821a e871520300      call    np_java_script+0x4d490 (2bead490)                          ; raise javascript exception
2be7821f 83c408          add     esp,8
2be78222 b86080e72b      mov     eax,offset np_java_script+0x18060 (2be78060)
```

During normal execution, the `JSNative` binding should return to the following code inside the implementation of the `js32u.dll!js_Invoke`. This code is responsible for popping the `JSStackFrame` that was used to execute the `JSNative` binding, and restore it from the `JSStackFrame` that triggered the invocation of the binding. At [19], the `JSStackFrame.down` field containing the previous value of `JSContext.fp` prior to the execution of the `js32u.dll!js_Invoke` function is loaded into the `%eax` register. After loading its previous value, the `JSContext.fp` field representing the current `JSStackFrame` is then restored so that javascript execution may resume after the execution of the “Document.flattenPages” binding. Due to the implementation of the `JSNative` “Document.flattenPages” binding raising a C++ exception, the following code is completely skipped over leaving the `JSStackFrame` that was created during the `js32u.dll!js_Invoke` function for “Document.flattenPages” still being stored to the `JSContext.fp` field. As the call to `js32u.dll!js_Invoke` was responsible for allocating the `JSStackFrame`, the raised of an exception results in the `JSStackFrame` going out of scope whilst still being stored within the `JSContext`.

```
js32u!js_Invoke+0x6cf:
2c0c35cf 8b5d10          mov     ebx,dword ptr [ebp+10h]            ; vp
2c0c35d2 8b45a4          mov     eax,dword ptr [ebp-5Ch]            ; frame.rval
2c0c35d5 8903            mov     dword ptr [ebx],eax
2c0c35d7 8b45b0          mov     eax,dword ptr [ebp-50h]            ; [19] frame.down
2c0c35da 894754          mov     dword ptr [edi+54h],eax            ; [20] JSContext.fp
2c0c35dd e969fbffff      jmp     js32u!js_Invoke+0x24b (2c0c314b)
```

Later, when the SpiderMonkey library tries to compile and evaluate javascript code, the library will attempt to setup a stack frame. This is done using the `js32u.dll!MaybeSetupFrame` function which will update the `JSContext` with a new `JSContext.fp` pointer that was passed as one of its parameters. Later in this function at [21], the new `JSStackFrame` passed as its parameters will have its `JSStackFrame.down` field assigned with the old `JSStackFrame` that was received. Afterwards at [22], the linked list of `JSStackFrame` structures will be walked looking for a `JSStackFrame` with the right flags in order to use when initializing the properties of the new `JSStackFrame`. As the `JSStackFrame` has gone out of scope due to the exception handler for the “Document.flattenPages” binding skipping over the restoration of the `JSContext.fp` field by the `js32u.dll!js_Invoke` function, this will result in a released object on the stack being written into the new linked list of `JSStackFrame` which will then later be written to. The usage of this released `JSStackFrame` can be used to cause memory corruption which can allow for code execution under the context of the application.

```
js32u!js_FindProperty+0xd880:
2c0d8cd0 55              push    ebp
2c0d8cd1 8bec            mov     ebp,esp
2c0d8cd3 56              push    esi
2c0d8cd4 8b7510          mov     esi,dword ptr [ebp+10h]                    ; oldfp
2c0d8cd7 85f6            test    esi,esi
2c0d8cd9 7418            je      js32u!js_FindProperty+0xd8a3 (2c0d8cf3)
2c0d8cdb 837e1000        cmp     dword ptr [esi+10h],0
2c0d8cdf 7412            je      js32u!js_FindProperty+0xd8a3 (2c0d8cf3)
...
js32u!js_FindProperty+0xd8e6:
2c0d8d36 897738          mov     dword ptr [edi+38h],esi                    ; [21] Assign JSStackFrame.down
2c0d8d39 85f6            test    esi,esi
2c0d8d3b 743c            je      js32u!js_FindProperty+0xd929 (2c0d8d79)
...
js32u!js_FindProperty+0xd900:
2c0d8d50 8b7638          mov     esi,dword ptr [esi+38h]                    ; [22] JSStackFrame.down
2c0d8d53 85f6            test    esi,esi
2c0d8d55 7422            je      js32u!js_FindProperty+0xd929 (2c0d8d79)
2c0d8d57 f6464c18        test    byte ptr [esi+4Ch],18h                     ; JSStackFrame.flags
2c0d8d5b 75f3            jne     js32u!js_FindProperty+0xd900 (2c0d8d50)
2c0d8d5d a818            test    al,18h
2c0d8d5f 7418            je      js32u!js_FindProperty+0xd929 (2c0d8d79)
```

The base addresses for the modules that were described are as follows.

```
Browse full module list
start    end        module name
2c090000 2c11b000   js32u      (export symbols)       C:\Program Files\Nitro\Pro\13\js32u.dll
2be60000 2c077000   np_java_script   (export symbols)       C:\Program Files\Nitro\Pro\13\np_java_script.dll
53370000 53e7c000   npdf       (export symbols)       C:\Program Files\Nitro\Pro\13\npdf.dll
```

Crash Information

Upon starting up the application in a debugger, open up a regular PDF document. When the document has been completely opened, close it and then break into the debugger. Set the following breakpoint which is at the entrypoint of the `Document::flattenPages` closure. Resume execution, and then open up the provided proof-of-concept. The debugger should break at the entrypoint of `Document.flattenPages`.

```
0:000> bp np_java_script+17ee0
0:000> g

eax=01ffd07c ebx=0ecc0310 ecx=309ab014 edx=2d007ee0 esi=309ab08c edi=306fee88
eip=2d007ee0 esp=01ffd020 ebp=01ffd0d8 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+0x17ee0:
2d007ee0 55              push    ebp
```

When examining its parameters on the stack, the first parameter is of the type `JSContext`. At offset `+0x54` of this structure is the `JSContext.fp` field which if we examine shows the address of the `JSContextFrame` being on the stack, and is still within scope. We then save the address to the `JSContext` in the `$t1` register in order to reference later. Afterwards, we can dump the exception chain to see which exception handlers have been setup prior to the ones in this function installed.

```
0:000> dc 4 + @esp L5
01ffd024  306fee88 308912e0 00000000 309ab08c  ..o0...0.......0
01ffd034  01ffd07c                             |...

0:000> r @$t1=@$p

0:000> dc @$p+54 L4
306feedc  01ffd050 00000000 306feef0 306feef0  P.........o0..o0

0:000> r @esp
esp=01ffd020

0:000> ? poi(@$p) > @esp
Evaluate expression: 1 = 00000001

0:000> !exchain
01ffd52c: np_java_script!CxImageJPG::Encode+97bee (2d0e9b6e)
01ffd898: np_java_script!CxImageJPG::Encode+9712a (2d0e90aa)
01ffd8e4: np_java_script!CxImageJPG::Encode+96901 (2d0e8881)
01ffd920: np_java_script!CxImageJPG::Encode+917f0 (2d0e3770)
01ffd984: NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+247c71 (00792641)
01ffd9ac: NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+23b568 (00785f38)
01ffdab0: NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+246df8 (007917c8)
01ffdb7c: NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+246b40 (00791510)
01ffdba0: NitroPDF!nitro::filenames_provider::workflow::get_from_program_data+23e148 (00788b18)
01ffdbcc: np_annotations!CxImageJPG::Encode+abd28 (2a7a8a98)
...
```

Afterwards, we will set a breakpoint where the exception handler will be installed. Set the following breakpoint, and resume execution. Upon the debugger encountering the breakpoint, we can use the `!exchain` command to output the exception chain. Set a breakpoint on the last address that was registered (first result from `!exchain`), so that we can know when its exception handler will be dispatched.

```
0:000> bp np_java_script+17f08
0:000> g

eax=01ffd010 ebx=0ecc0310 ecx=309ab014 edx=2d007ee0 esi=309ab08c edi=306fee88
eip=2d007f08 esp=01ffcff4 ebp=01ffd01c 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+0x17f08:
2d007f08 8965f0          mov     dword ptr [ebp-10h],esp ss:0023:01ffd00c=309ab098

0:000> !exchain
01ffd010: np_java_script!CxImageJPG::Encode+93690 (2d0e5610)
01ffd52c: np_java_script!CxImageJPG::Encode+97bee (2d0e9b6e)
01ffd898: np_java_script!CxImageJPG::Encode+9712a (2d0e90aa)
01ffd8e4: np_java_script!CxImageJPG::Encode+96901 (2d0e8881)
01ffd920: np_java_script!CxImageJPG::Encode+917f0 (2d0e3770)

0:000> bp np_java_script+f5610
```

The next breakpoint is where the implementation of `Document.flattenPages` is called. When setting the following breakpoint, resume execution and the debugger will break at the function call that enters the `Document.flattenPages` implementation. If we examine its parameters, we can see that the pages which will be flattened are using `-1` for both page boundaries. When stepping over this function, an exception will get raised which will dispatch the structured exception handler that was registered by this function.

```
0:000> bp np_java_script+181b6
0:000> g

eax=00000000 ebx=ffffffff ecx=70b55983 edx=2d007e00 esi=ffffffff edi=39018ff0
eip=2d0081b6 esp=01ffcfe4 ebp=01ffd01c iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
np_java_script+0x181b6:
2d0081b6 e8a5770000      call    np_java_script+0x1f960 (2d00f960)

0:000> dc @esp L4
01ffcfe4  39018ff0 ffffffff ffffffff 1cf6d300  ...9............
```

Next, we can step over the implementation of `Document.flattenPages` which will result in exceptions being dispatched. Eventually, the breakpoint of the structured exception handler that was registered will be encountered. If we dump the address of the `JSContext` that we had saved, and print out the `JSContext.fp` field at offset `+0x54` of the structure we will see the `JSStackContext` that was allocated on the stack by the `js32u!js_Invoke` function. Comparing this to the value of the `%esp` register shows that this stack address is still valid.

```
0:000> p
(894.1678): C++ EH exception - code e06d7363 (first chance)
(894.1678): C++ EH exception - code e06d7363 (first chance)
(894.1678): C++ EH exception - code e06d7363 (first chance)
...

eax=00000000 ebx=00000000 ecx=2d0e5610 edx=77ba3b20 esi=00000000 edi=00000000
eip=2d0e5610 esp=01ffcae0 ebp=01ffcb00 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
np_java_script!CxImageJPG::Encode+0x93690:
2d0e5610 8b542408        mov     edx,dword ptr [esp+8] ss:0023:01ffcae8=01ffd010

0:000> ? @$t1
Evaluate expression: 812641928 = 306fee88

0:000> dc @$t1+54 L4
306feedc  01ffd050 00000000 306feef0 306feef0  P.........o0..o0

0:000> r @esp
esp=01ffcae0

0:000> ? poi(@$t1+54) > @esp
Evaluate expression: 1 = 00000001
```

After resuming execution, the exception handlers will be executed which will result in the `JSStackContext` going out of scope and may be overwritten by another function that is being executed. As the value will still be stored in the `JSContext.fp` field, the next time the SpiderMonkey library uses the `JSContext` it will use that written value in order to facilitate any required logic by the javascript code within the document. If we resume execution, the application will crash in a number of places while using this value such as in the following function call.

```
0:000> g
(894.1678): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00200000 ebx=00000000 ecx=00000005 edx=00000000 esi=00200000 edi=306fee88
eip=2d25b1a2 esp=01ffd334 ebp=01ffd35c iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
js32u!js_LookupProperty+0x1f42:
2d25b1a2 8b4008          mov     eax,dword ptr [eax+8] ds:0023:00200008=????????
```

If we examine the value we have previously saved, we can compare it to the arguments of the function call that we've crashed in. Confirming they match proves that the first parameter to this function is the `JSContext` with our out-of-scope `JSStackContext`. Printing out the value at offset `+0x54` of this structure shows the same stack pointer, and then comparing it to our current stack pointer shows that it has completely gone out of scope. Stacks grow downwards on the Intel architecture, and so if a value on the stack is below the `%esp` register, it should not exist anymore. If we output the disassembly of the surrounding instructions, we can see that later during execution an address will be dereferenced off of our pointer, and then called. Thus the stack variable going out of scope is then used which demonstrates that code execution can be achieved if the `JSStackContext.fp` field is properly controlled.

```
0:000> ? @$t1
Evaluate expression: 812641928 = 306fee88

0:000> dc @ebp+8 L3
01ffd364  306fee88 021c87f4 00000005           ..o0........

0:000> dc @$p+54 L4
306feedc  01ffd4f8 3d72fbe8 306feef0 306feef0  ......r=..o0..o0

0:000> ? poi(@$p) > @esp
Evaluate expression: 0 = 00000000

0:000> u . L0n14
js32u!js_LookupProperty+0x1f42:
2d25b1a2 8b4008          mov     eax,dword ptr [eax+8]
2d25b1a5 83e0f8          and     eax,0FFFFFFF8h
2d25b1a8 75f6            jne     js32u!js_LookupProperty+0x1f40 (2d25b1a0)
2d25b1aa 8b460c          mov     eax,dword ptr [esi+0Ch]
2d25b1ad 83e0fc          and     eax,0FFFFFFFCh
2d25b1b0 f7400400000100  test    dword ptr [eax+4],10000h
2d25b1b7 7410            je      js32u!js_LookupProperty+0x1f69 (2d25b1c9)
2d25b1b9 8b4050          mov     eax,dword ptr [eax+50h]
2d25b1bc 85c0            test    eax,eax
2d25b1be 7409            je      js32u!js_LookupProperty+0x1f69 (2d25b1c9)
2d25b1c0 56              push    esi
2d25b1c1 57              push    edi
2d25b1c2 ffd0            call    eax
2d25b1c4 83c408          add     esp,8


Browse full module list
start    end        module name
2d220000 2d2ab000   js32u      (export symbols)       C:\Program Files\Nitro\Pro\13\js32u.dll
2cff0000 2d207000   np_java_script   (export symbols)       C:\Program Files\Nitro\Pro\13\np_java_script.dll
00370000 00dd6000   NitroPDF   (export symbols)       C:\Program Files\Nitro\Pro\13\NitroPDF.exe
```

Exploit Proof of Concept

In the provided proof-of-concept, it is object 110 revision 0 that contains the javascript that 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 - Follow up w/ vendor

2021-06-07 - 2nd follow up w/ vendor

2021-06-11 - Advisories reissued to vendor

2021-06-22 - Talos and vendor discussion; Vendor acknowledged reports

2021-08-26 - Talos granted final disclosure extension

2021-09-10 - Vendor patched

2021-09-13 - Public release

CREDIT

Discovered by a member of Cisco Talos.
