Talos Vulnerability Report

TALOS-2020-1063

Nitro Pro PDF Object Stream Parsing Number of Objects Remote Code Execution Vulnerability

SEPTEMBER 15, 2020

CVE NUMBER

CVE-2020-6113

Summary

An exploitable vulnerability exists in the object stream parsing functionality of Nitro Software, Inc.'s Nitro Pro 13.13.2.242 when updating its cross-reference table. When processing an object stream from a PDF document, the application will perform a calculation in order to allocate memory for the list of indirect objects. Due to an error when calculating this size, an integer overflow may occur which can result in an undersized buffer being allocated. Later when initializing this buffer, the application can write outside its bounds which can cause a memory corruption that can lead to code execution. A specially crafted document can be delivered to a victim in order to trigger this vulnerability.

Tested Versions

Nitro Pro 13.13.2.242
Nitro Pro 13.16.2.300

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-190 - Integer Overflow or Wraparound

Details

Nitro Software, Inc. includes their flagship product, Nitro Pro as part of their Nitro Productivity Suite. Nitro Pro is Nitro Software's PDF editor and flagship product. This product allows users to create and modify documents that follow the Portable Document Format (PDF) specification and other digital documents.

The PDF format contains a number of different objects which are then referenced indirectly by the various parts of the file in order to render the document to the user. Each object is identified by a numerical identifier and their offset within the file which is stored within a cross-reference table. This allows the application to quickly locate where a particular object is so that it can be used by the document.

Within a document, the PDF format allows one to store what are known as "Object Streams". These object streams allow for a document author to pack a number of objects within a single object that may be referenced by the document. These objects are then described by a cross-reference table in order to allow the application to quickly locate any object required to render a document. Cross-reference tables within a document can be stored in only a few ways. The most common way is within what is known as the "trailer", and is typically located by a numerical file offset found near the end of the document. This "trailer" contains a list of individual segments describing the file offset of each individual object and its identifier which corresponds to the row within the segment. In order to allow for a more compact way of storing cross-references, the PDF format allows one to store the cross-reference table in a binary format which is then decoded as part of the overall cross-reference table for the "trailer".

When processing the cross-reference tables, the application will iterate through each of its items in order to build an internal lookup table to locate an individual object during the rendering process. When an object of type `/ObjStm` is identified through the cross-reference table, the application will begin to parse its attributes in order to determine what the application needs to do in order to update its lookup table. In the following code, the application will enter a function to begin parsing the attributes of the object. At [1], the object's type is checked to ensure it is not a referenced object. If it isn't, the application will resume execution until the next check. At [2], the application will verify that the object's type is a stream.

```
npdf!CosNewBoolean+0x740:
58ae6080 55              push    ebp
58ae6081 8bec            mov     ebp,esp
58ae6083 6aff            push    0FFFFFFFFh
58ae6085 68e837fe58      push    offset npdf!CAPContent::Wrap+0x281ff8 (58fe37e8)
58ae608a 64a100000000    mov     eax,dword ptr fs:[00000000h]
58ae6090 50              push    eax
58ae6091 83ec50          sub     esp,50h
58ae6094 53              push    ebx
58ae6095 56              push    esi
58ae6096 57              push    edi
58ae6097 a15cfd1759      mov     eax,dword ptr [npdf!CAPAction::smFitTypes+0x69dc (5917fd5c)]
58ae609c 33c5            xor     eax,ebp
58ae609e 50              push    eax
58ae609f 8d45f4          lea     eax,[ebp-0Ch]
58ae60a2 64a300000000    mov     dword ptr fs:[00000000h],eax
58ae60a8 8965f0          mov     dword ptr [ebp-10h],esp
58ae60ab 8b7508          mov     esi,dword ptr [ebp+8]                   // Object
58ae60ae 85f6            test    esi,esi
58ae60b0 0f842c040000    je      npdf!CosNewBoolean+0xba2 (58ae64e2)
...
npdf!CosNewBoolean+0x780:
58ae60c0 8a06            mov     al,byte ptr [esi]                       // Object
58ae60c2 3c09            cmp     al,9                                    // [1] PDFReference
58ae60c4 754d            jne     npdf!CosNewBoolean+0x7d3 (58ae6113)
...
npdf!CosNewBoolean+0x7d3:
58ae6113 3c08            cmp     al,8                                    // [2] PDFStream
58ae6115 741b            je      npdf!CosNewBoolean+0x7f2 (58ae6132)
```

After confirming that the object is a stream and not a reference to another object, the following code will be executed. This code will first convert the “Type” string into an atom at [3]. This atom is used to get the object type from the stream’s dictionary. The stream’s dictionary is fetched at [4], and then at [5] the atom for “Type” is used to get its value out of the stream’s dictionary. Finally at [6], the “ObjStm” string is converted into an atom which is then compared against the “ObjStm” atom. If all of these conditionals are satisfied, the application then knows that it is to process the object as an object stream and it needs to update its lookup table with any objects found within.

```
npdf!CosNewBoolean+0x822:
58ae6162 8d45e4          lea     eax,[ebp-1Ch]                                           // Atom result
58ae6165 b920601659      mov     ecx,offset npdf!CAPContent::`vftable'+0x139930 (59166020)  // "Type"
58ae616a 50              push    eax
58ae616b e8909fffff      call    npdf!local_file_handle::write+0x1000 (58ae0100)           // [3] GetAtomFromString
58ae6170 52              push    edx
58ae6171 50              push    eax
58ae6172 ff7508          push    dword ptr [ebp+8]
58ae6175 e8761a0000      call    npdf!CosNewDict+0x420 (58ae7bf0)                        // [4] Get the object's dictinary
58ae617a 83c410          add     esp,10h
58ae617d 84c0            test    al,al
58ae617f 751f            jne     npdf!CosNewBoolean+0x860 (58ae61a0)
...
npdf!CosNewBoolean+0x860:
58ae61a0 8d45c4          lea     eax,[ebp-3Ch]                                           // Atom
58ae61a3 50              push    eax
58ae61a4 ff75e4          push    dword ptr [ebp-1Ch]                                     // atom "/Type"
58ae61a7 e8442e0000      call    npdf!CosNewName+0x1b0 (58ae8ff0)                        // [5] Get value of name
58ae61ac 83c408          add     esp,8
58ae61af 84c0            test    al,al
58ae61b1 751f            jne     npdf!CosNewBoolean+0x892 (58ae61d2)
...
npdf!CosNewBoolean+0x892:
58ae61d2 b9c06f1659      mov     ecx,offset npdf!CAPContent::`vftable'+0x13a8d0 (59166fc0)  // "ObjStm"
58ae61d7 e8249fffff      call    npdf!local_file_handle::write+0x1000 (58ae0100)           // [6] GetAtomFromString
58ae61dc 3945c4          cmp     dword ptr [ebp-3Ch],eax                                 // Compare Atom to "/ObjStm"
58ae61df 0f85e5020000    jne     npdf!CosNewBoolean+0xb8a (58ae64ca)
58ae61e5 3955c8          cmp     dword ptr [ebp-38h],edx                                 // Compare Atom to "/ObjStm"
58ae61e8 0f85dc020000    jne     npdf!CosNewBoolean+0xb8a (58ae64ca)
```

The application will then proceed to collect different attributes about the object and store them to local variables. One such attribute is the fetched and used by the following code. At [7], the string “N” is converted to an atom. This atom represents the number of indirect objects stored within the stream. At [8], the value for this atom is fetched. This value is then converted into an integer at [9]. After fetching the value for the number of indirect objects and storing it at [10], the application will fetch the value again in order to prepend it to the allocated buffer.

```
npdf!CosNewBoolean+0x8ae:
58ae61ee 8d45e0          lea     eax,[ebp-20h]                                           // Atom result
58ae61f1 b9a0611659      mov     ecx,offset npdf!CAPContent::`vftable'+0x139ab0 (591661a0)  // "N"
58ae61f6 50              push    eax
58ae61f7 e8049fffff      call    npdf!local_file_handle::write+0x1000 (58ae0100)           // [7] GetAtomFromString
58ae61fc 52              push    edx
58ae61fd 50              push    eax
58ae61fe ff7508          push    dword ptr [ebp+8]
58ae6201 e8ea190000      call    npdf!CosNewDict+0x420 (58ae7bf0)                        // [8] Get the object's value from the
dictionary
58ae6206 83c410          add     esp,10h
58ae6209 84c0            test    al,al
58ae620b 751f            jne     npdf!CosNewBoolean+0x8ec (58ae622c)
...
npdf!CosNewBoolean+0x8ec:
58ae622c ff75e0          push    dword ptr [ebp-20h]
58ae622f c745fc00000000  mov     dword ptr [ebp-4],0
58ae6236 c645fc01        mov     byte ptr [ebp-4],1
58ae623a e821270000      call    npdf!CosIntegerValue (58ae8960)                         // [9] Convert to an integer
58ae623f 83c404          add     esp,4
58ae6242 c745fcffffffff  mov     dword ptr [ebp-4],0FFFFFFFFh
58ae6249 8bf8            mov     edi,eax                                                 // [10] Store to %edi
58ae624b b910671659      mov     ecx,offset npdf!CAPContent::`vftable'+0x13a020 (59166710)
...
npdf!CosNewBoolean+0x910:
58ae6250 8d45dc          lea     eax,[ebp-24h]
58ae6253 897de8          mov     dword ptr [ebp-18h],edi                                 // [10] Store to local variable
58ae6256 50              push    eax
58ae6257 e8a49effff      call    npdf!local_file_handle::write+0x1000 (58ae0100)         // GetAtomFromString
58ae625c 52              push    edx
58ae625d 50              push    eax
58ae625e ff7508          push    dword ptr [ebp+8]
58ae6261 e88a190000      call    npdf!CosNewDict+0x420 (58ae7bf0)                        // Get the object's value from the
dictionary
58ae6266 83c410          add     esp,10h
58ae6269 84c0            test    al,al
58ae626b 751f            jne     npdf!CosNewBoolean+0x94c (58ae628c)
...
npdf!CosNewBoolean+0x94c:
58ae628c ff75dc          push    dword ptr [ebp-24h]
58ae628f c745fc04000000  mov     dword ptr [ebp-4],4
58ae6296 c645fc05        mov     byte ptr [ebp-4],5
58ae629a e831270000      call    npdf!CosInteger64Value (58ae89d0)                       // Convert to a 64-bit integer
58ae629f 83c404          add     esp,4
58ae62a2 8945d8          mov     dword ptr [ebp-28h],eax                                 // Store low 32-bits
58ae62a5 8955d0          mov     dword ptr [ebp-30h],edx                                 // Store high 32-bits
58ae62a8 c745fcffffffff  mov     dword ptr [ebp-4],0FFFFFFFFh
```

After retrieving the integer for the value for “/N” from the object’s dictionary, the application will verify that the value is unsigned and then adjust it before using the value to allocate memory. At [11], the application will add 1 to its value and then multiply it by 0x10 (16) prior to passing it to the `malloc` function call. Due to the application not checking whether this calculation can overflow, this multiply can result in an undersized allocation if its value when multiplied by 0x10 (16) is larger than 32-bits. The result of this calculation is then stored into a local variable at [12]. At the beginning of the first 0x10 bytes of memory of the allocation, some fields are assigned at [13] which includes its 64-bit value as well as its actual 32-bit value. Once this is done, the size will be loaded again at [14] and then used to initialize the buffer. At [15], the value is again multiplied by 0x10 (16) and then the value 0x10 is added to the pointer to shift it past the initial fields. Finally at [16], the `memset` function is called to zero its memory. Due to the application using a length for its allocation that differs from the call to `memset`, this can result in a buffer overflow which causes memory corruption. Under the proper conditions, this can lead to code execution under the context of the application.

```
npdf!CosNewBoolean+0x99f:
58ae62df b802000000      mov     eax,2
58ae62e4 85ff            test    edi,edi
58ae62e6 7e03            jle     npdf!CosNewBoolean+0x9ab (58ae62eb)
58ae62e8 8d4701          lea     eax,[edi+1]                                     // [11] Add 1 to N
58ae62eb c1e004          shl     eax,4                                           // [11] Multiply by 0x10
58ae62ee 50              push    eax
58ae62ef ff1580f6ff58    call    dword ptr [npdf!CAPContent::Wrap+0x29de90 (58fff680)]  // malloc
...
58ae62f5 8b4dd8          mov     ecx,dword ptr [ebp-28h]
58ae62f8 8945ec          mov     dword ptr [ebp-14h],eax                         // [12] Store buffer that was allocated
58ae62fb 894808          mov     dword ptr [eax+8],ecx                           // [13] Low 32-bits
58ae62fe 8b4dd0          mov     ecx,dword ptr [ebp-30h]
58ae6301 89480c          mov     dword ptr [eax+0Ch],ecx                         // [13] High 32-bits
...
58ae6304 8bcf            mov     ecx,edi                                         // [14] Load value from "/N"
58ae6306 c1e104          shl     ecx,4                                           // [15] Multiply by 0x10
58ae6309 51              push    ecx
58ae630a 8938            mov     dword ptr [eax],edi                             // [13] Size
58ae630c 83c010          add     eax,10h                                         // [15] Add 0x10 to pointer
58ae630f 6a00            push    0
58ae6311 50              push    eax
58ae6312 e859b34f00      call    npdf!CAPContent::Wrap+0x27fe80 (58fe1670)       // [16] memset
```

This report uses the following base addresses for the libraries in the application.

```
0:000> lm m nitropdf
Browse full module list
start    end        module name
002c0000 00b41000   NitroPDF   (deferred)
58a30000 59477000   npdf       (export symbols)       npdf.dll
```

Crash Information

The provided proof-of-concept sets the relevant fields to cause the application to allocate a zero-sized buffer. This results in any attempt to set fields in the header to write outside the bounds of the buffer. When opening up the provided proof-of-concept within the application, the following crash will occur.

```
(165c.12d8): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=3253cff8 ebx=3cce0fe0 ecx=00000002 edx=00000000 esi=312c6fc8 edi=0fffffff
eip=58ae62fb esp=01c7d7b4 ebp=01c7d824 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00210202
npdf!CosNewBoolean+0x9bb:
58ae62fb 894808          mov     dword ptr [eax+8],ecx ds:0023:3253d000=????????
```

Adding 1 to the length from the `%edi` register and multiplying it by 0x10 will result in an integer overflow where the lower 32-bits will be set to 0.

```
0:000> ? (@edi+1)*10
Evaluate expression: 4294967296 = 00000001`00000000
```

Timeline

2020-05-06 - Vendor Disclosure

2020-07-27 - Vendor Patched

2020-09-15 - Public Release

CREDIT

Discovered a member of Cisco Talos.
