Talos Vulnerability Report

TALOS-2020-1062

Nitro Pro PDF JPEG2000 Stripe Sub-sample Decoding Out-of-bounds Write Code Execution Vulnerability

SEPTEMBER 15, 2020

CVE NUMBER

CVE-2020-6112

Summary

An exploitable code execution vulnerability exists in the JPEG2000 Stripe Decoding functionality of Nitro Software, Inc.’s Nitro Pro 13.13.2.242 when decoding sub-samples. While initializing tiles with sub-sample data, the application can miscalculate a pointer for the stripes in the tile which allow for the decoder to write out of-bounds and cause memory corruption. This can result in code execution. A specially crafted image can be embedded inside a PDF and loaded by 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-823 - Use of Out-of-range Pointer Offset

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 allows for creators to embed various image types encoded in a number of different formats, and as such the Nitro PDF application includes support for these. The Nitro PDF application includes support for both the JPEG and JPEG2000 image formats which is facilitated by usage of the Kakadu JPEG2000 library. This allows Nitro PDF to accommodate the user with a number of the capabilities of the JPEG2000 image format.

When Nitro Pro PDF loads a document containing an object stream encoded with the “JPXDecode” filter, the application will use the Kakadu library in order to perform the decoding of said image. Once initializing the library and constructing the decoder object, the application will begin to allocate tiles for the image. These tiles are used to contain sub-sample data for each color component that was described by the image header. After allocating these tiles for the image and the image’s color components, the application will begin to initialize these tiles with sub-sample data. During this decoding, the application will calculate the stride to traverse through each individual sub-sample within a tile that belongs to a particular color component. Due to the calculation not accommodating for whether the width of the stripe can be larger than the tiles themselves, the stride can be made to be larger than the tile for an individual color component. During the decoding of each stripe, this stride can then be used to seek past the boundaries of the tiles themselves and write to memory that is outside their bounds. When the sub-sample data is then written to the stripe, memory can then be corrupted.

When first decoding an image encoded with the `/JPXDecode` filter, the following function is executed to determine the filter. This function will simply read the result of the `/Filter` attribute and use it to determine which filter type to use for decoding. Firstly it will check against the `/ASCIIHexDecode` atom, and continue by checking against each of the other available filter types. Partway through the execution of this function at [1], the application will fetch the atom for the string `/JPXDecode` and then compare the filter against this atom. If this is the case, the function call at [2] will be made in order to perform JPX decoding.

```
npdf!nitro::document_security::security_model::write_encryption_dictionary+0x4010:
5b62f8e0 55              push    ebp
5b62f8e1 8bec            mov     ebp,esp
5b62f8e3 8b0de865c95b    mov     ecx,dword ptr [npdf!CAPContent::`vftable'+0x139ef8 (5bc965e8)]     ; atom(ASCIIHexDecode)
5b62f8e9 8bc1            mov     eax,ecx
5b62f8eb 8b15ec65c95b    mov     edx,dword ptr [npdf!CAPContent::`vftable'+0x139efc (5bc965ec)]     ; atom(ASCIIHexDecode)
5b62f8f1 23c2            and     eax,edx
5b62f8f3 53              push    ebx
5b62f8f4 8b5d14          mov     ebx,dword ptr [ebp+14h]
5b62f8f7 56              push    esi
5b62f8f8 57              push    edi
5b62f8f9 8b7b08          mov     edi,dword ptr [ebx+8]
5b62f8fc 8b730c          mov     esi,dword ptr [ebx+0Ch]
5b62f8ff 83f8ff          cmp     eax,0FFFFFFFFh
5b62f902 751c            jne     npdf!nitro::document_security::security_model::write_encryption_dictionary+0x4050 (5b62f920)
...
npdf!nitro::document_security::security_model::write_encryption_dictionary+0x425c:
5b62fb2c b92066c95b      mov     ecx,offset npdf!CAPContent::`vftable'+0x139f30 (5bc96620)
5b62fb31 e8ca05feff      call    npdf!local_file_handle::write+0x1000 (5b610100)                    ; [1] GetAtomFromString
5b62fb36 3bf8            cmp     edi,eax
5b62fb38 751a            jne     npdf!nitro::document_security::security_model::write_encryption_dictionary+0x4284 (5b62fb54)
5b62fb3a 3bf2            cmp     esi,edx
5b62fb3c 7516            jne     npdf!nitro::document_security::security_model::write_encryption_dictionary+0x4284 (5b62fb54)
5b62fb3e ff751c          push    dword ptr [ebp+1Ch]
5b62fb41 ff7518          push    dword ptr [ebp+18h]
5b62fb44 ff7508          push    dword ptr [ebp+8]
5b62fb47 e804520000      call    npdf!CAPCosStream::SetAttributesDict+0x2900 (5b634d50)             ; [2] Decode JPX stream
```

Inside the function call at [3], the application will begin opening up the stream by constructing an object representing the JPX image source. This corresponds to the `jpx_source` class from the Kakadu library. After constructing this object from the stream as required by the Kakadu library, the object will be passed to the function call at [4].

```
npdf!CAPCosStream::SetAttributesDict+0x2926:
5b634d76 6a01            push    1
5b634d78 ff7508          push    dword ptr [ebp+8]
5b634d7b 8d4d84          lea     ecx,[ebp-7Ch]                                        // jpx_source object
5b634d7e e88ddeffff      call    npdf!CAPCosStream::SetAttributesDict+0x7c0 (5b632c10)      // [3] jpx_source::open
...
5b634d83 ff7510          push    dword ptr [ebp+10h]
5b634d86 8d4d84          lea     ecx,[ebp-7Ch]                                        // jpx_source object
5b634d89 c745fc00000000  mov     dword ptr [ebp-4],0
5b634d90 ff750c          push    dword ptr [ebp+0Ch]
5b634d93 e808e9ffff      call    npdf!CAPCosStream::SetAttributesDict+0x1250 (5b6336a0)     // [4]
5b634d98 85d2            test    edx,edx
5b634d9a 7c0a            jl      npdf!CAPCosStream::SetAttributesDict+0x2956 (5b634da6)
5b634d9c 7f04            jg      npdf!CAPCosStream::SetAttributesDict+0x2952 (5b634da2)
...
5b634da8 8d4d84          lea     ecx,[ebp-7Ch]                                        // jpx_source object
5b634dab e8a0e5ffff      call    npdf!CAPCosStream::SetAttributesDict+0xf00 (5b633350)      // destructor
```

After initializing some variables to assist with decoding, the application will first perform some sanity checks on some values from the JPX source. At [5], the number of components from the JPEG2000 image header will be checked. After validating that these fields are non-zero, at [6] the application will pass the JPX source object to a constructor for creating the object representing the JPX code-stream. This corresponds to the `kdu_core::kdu_codestream` object from the Kakadu library headers, and the method is suspected to be `kdu_core::kdu_codestream::create`. This method is directly responsible for reading the different parameters as specified within the JPX code-stream such as the contents of the `SIZ` marker, the JPX dimensions [7], and the number of components that will be used later. Upon returning from the construction of the code-stream object, it is suspected by the author that the application calls the method `kdu_core::kdu_codestream::apply_input_restrictions` from the Kakadu library.

```
npdf!CAPCosStream::SetAttributesDict+0x1250:
5b6336a0 55              push    ebp
5b6336a1 8bec            mov     ebp,esp
...
5b6336d6 8bf1            mov     esi,ecx                                   // jpx_source object
...
5b6336f7 837e0800        cmp     dword ptr [esi+8],0
5b6336fb 0f8407060000    je      npdf!CAPCosStream::SetAttributesDict+0x18b8 (5b633d08)
5b633701 837e0400        cmp     dword ptr [esi+4],0                       // [5] ImageHeader.NC
5b633705 0f84fd050000    je      npdf!CAPCosStream::SetAttributesDict+0x18b8 (5b633d08)
...
npdf!CAPCosStream::SetAttributesDict+0x12bb:
5b63370b 6a00            push    0
5b63370d 6a00            push    0
5b63370f ff765c          push    dword ptr [esi+5Ch]                       // jpx_source object
5b633712 8d8d68e1ffff    lea     ecx,[ebp-1E98h]
5b633718 c78568e1ffff00000000 mov dword ptr [ebp-1E98h],0
5b633722 e8d9952d00      call    npdf!CAPContent::Wrap+0x7b510 (5b90cd00)   // [6] kdu_core::kdu_codestream::create \
\
npdf!CAPContent::Wrap+0x77995:
5b909185 8b06            mov     eax,dword ptr [esi]
5b909187 0f108000010000  movups  xmm0,xmmword ptr [eax+100h]               // [7] Dimensions from SIZ marker
5b90918e 0f11804c010000  movups  xmmword ptr [eax+14Ch],xmm0               // Written into kdu_codestream object
5b909195 8b4518          mov     eax,dword ptr [ebp+18h]
5b909198 85c0            test    eax,eax
5b90919a 7412            je      npdf!CAPContent::Wrap+0x779be (5b9091ae)
```

After constructing the code-stream object, the application will initialize an array which can retain a number of objects up to a certain count. These objects contained by this list are of the `kdu_core::kdu_dims` structure from the Kakadu library. As found at [8], this list can hold up to 0x40 elements within itself before the application will use dynamically allocated memory to store its contents.

```
npdf!CAPCosStream::SetAttributesDict+0x12f3:
5b633743 8b4604          mov     eax,dword ptr [esi+4]
5b633746 8d8decfbffff    lea     ecx,[ebp-414h]                ; List elements stored on stack
5b63374c 898de0fbffff    mov     dword ptr [ebp-420h],ecx       ; Pointer to elements of list
5b633752 c785e4fbffff00000000 mov dword ptr [ebp-41Ch],0        ; Next available slot
5b63375c c785e8fbffff40000000 mov dword ptr [ebp-418h],40h      ; [8] Length
5b633766 898d80e1ffff    mov     dword ptr [ebp-1E80h],ecx
5b63376c 8d8de0fbffff    lea     ecx,[ebp-420h]
5b633772 c745fc00000000  mov     dword ptr [ebp-4],0
5b633779 83f840          cmp     eax,40h
5b63377c 7622            jbe     npdf!CAPCosStream::SetAttributesDict+0x1350 (5b6337a0)
```

Once the application has finished allocating the list, the application will begin to initialize it with the elements found by the `kdu_core::kdu_codestream` object. At [9], the application will load the number of components into the `%eax` register, check it against zero, and then enter the loop at [10]. This loop will continue until the number of components as identified by the code-stream object, and read the dimensions into the list that was allocated in the prior block of code. Afterwards, the application will do the similar for the recommended stripe heights list.

```
npdf!CAPCosStream::SetAttributesDict+0x1363:
5b6337b3 8b4604          mov     eax,dword ptr [esi+4]                      ; [9] Number of components from kdu_codestream object
5b6337b6 33ff            xor     edi,edi
5b6337b8 c745fc01000000  mov     dword ptr [ebp-4],1
5b6337bf 85c0            test    eax,eax
5b6337c1 743b            je      npdf!CAPCosStream::SetAttributesDict+0x13ae (5b6337fe)
..
npdf!CAPCosStream::SetAttributesDict+0x1373:
5b6337c3 33c9            xor     ecx,ecx
5b6337c5 898d70e1ffff    mov     dword ptr [ebp-1E90h],ecx
5b6337cb 0f1f440000      nop     dword ptr [eax+eax]
...
npdf!CAPCosStream::SetAttributesDict+0x1380:
5b6337d0 8b85e0fbffff    mov     eax,dword ptr [ebp-420h]                   ; [10] Pointer to elements of list
5b6337d6 03c1            add     eax,ecx
5b6337d8 8d8d68e1ffff    lea     ecx,[ebp-1E98h]                            ; Codestream object
5b6337de 6a01            push    1
5b6337e0 50              push    eax                                        ; Locations to store kdu_dims structure for each component
5b6337e1 57              push    edi
5b6337e2 e839fa2d00      call    npdf!CAPContent::Wrap+0x81a30 (5b913220)   ; [11] kdu_core::kdu_codestream::get_dims
5b6337e7 8b8d70e1ffff    mov     ecx,dword ptr [ebp-1E90h]
5b6337ed 47              inc     edi
5b6337ee 8b4604          mov     eax,dword ptr [esi+4]                      ; Number of components from kdu_codestream object
5b6337f1 83c110          add     ecx,10h                                    ; Seek kdu_dims pointer to next element in list
5b6337f4 898d70e1ffff    mov     dword ptr [ebp-1E90h],ecx                  ; Update pointer
5b6337fa 3bf8            cmp     edi,eax
5b6337fc 72d2            jb      npdf!CAPCosStream::SetAttributesDict+0x1380 (5b6337d0)
```

After allocating the list for the recommended stripe heights, the application will execute the following code which is responsible for constructing an object for decompressing stripes. Referring the headers from the Kakadu library, it is suspected that the function at [12] is the constructor for the `kdu_core::kdu_stripe_decompressor` object. Once constructing the object, the application will initialize it by calling one of its methods at [13]. Inside this method, the application will copy some fields from the `kdu_core:kdu_codestream` such as the number of components at [14], and copy them into the stripe decompressor object. At [15], the application will call a method that will calculate the dimensions of the grid that will be used later. The results of these dimensions are then stored into the stripe decompressor object.

```
npdf!CAPCosStream::SetAttributesDict+0x1444:
5b633894 8d8d2ce0ffff    lea     ecx,[ebp-1FD4h]
5b63389a c645fc05        mov     byte ptr [ebp-4],5
5b63389e e8bd9d3000      call    npdf!CAPContent::Wrap+0xabe70 (5b93d660)   ; [12] kdu_core::kdu_stripe_decompressor
...
5b6338a3 6a00            push    0
5b6338a5 6aff            push    0FFFFFFFFh
5b6338a7 6a00            push    0
5b6338a9 6a00            push    0
5b6338ab 6a00            push    0
5b6338ad 6a00            push    0
5b6338af 6a01            push    1
5b6338b1 ffb568e1ffff    push    dword ptr [ebp-1E98h]                      ; kdu_core::kdu_codestream object
5b6338b7 8d8d2ce0ffff    lea     ecx,[ebp-1FD4h]                            ; kdu_core::kdu_stripe_decompreessor object
5b6338bd c645fc06        mov     byte ptr [ebp-4],6
5b6338c1 e8cabf3000      call    npdf!CAPContent::Wrap+0xae0a0 (5b93f890)   ; [13] \
\
npdf!CAPContent::Wrap+0xae0a0:
5b93f890 55              push    ebp
5b93f891 8bec            mov     ebp,esp
5b93f893 83ec24          sub     esp,24h
5b93f896 53              push    ebx
5b93f897 8bd9            mov     ebx,ecx                                    ; this
5b93f899 837b3400        cmp     dword ptr [ebx+34h],0
5b93f89d 7561            jne     npdf!CAPContent::Wrap+0xae110 (5b93f900)
...
npdf!CAPContent::Wrap+0xae12b:
5b93f91b 8d4d08          lea     ecx,[ebp+8]                                ; kdu_core::kdu_codestream object
5b93f91e 884308          mov     byte ptr [ebx+8],al
5b93f921 8a4510          mov     al,byte ptr [ebp+10h]
5b93f924 6a01            push    1
5b93f926 884309          mov     byte ptr [ebx+9],al
5b93f929 e8c23cfdff      call    npdf!CAPContent::Wrap+0x81e00 (5b9135f0)   ; [14] return the number of components
5b93f92e 89430c          mov     dword ptr [ebx+0Ch],eax                    ; Store the number of components into the
kdu_stripe_decompressor object
5b93f931 8d4d08          lea     ecx,[ebp+8]                                ; kdu_core::kdu_codestream object
5b93f934 8d45dc          lea     eax,[ebp-24h]                              ; kdu_core::kdu_dims result
5b93f937 0f57c0          xorps   xmm0,xmm0
5b93f93a 50              push    eax
5b93f93b 0f1145dc        movups  xmmword ptr [ebp-24h],xmm0
5b93f93f e85c40fdff      call    npdf!CAPContent::Wrap+0x821b0 (5b9139a0)   ; [15] calculate the dimensions of the grid
...
5b93f944 8b4ddc          mov     ecx,dword ptr [ebp-24h]                    ; kdu_dims::pos.y
5b93f947 8b45e8          mov     eax,dword ptr [ebp-18h]                    ; kdu_dims::size.x
5b93f94a 8b55e4          mov     edx,dword ptr [ebp-1Ch]                    ; kdu_dims::size.y
5b93f94d 894b14          mov     dword ptr [ebx+14h],ecx                    ; store y position into kdu_decompressor object
5b93f950 898bb0000000    mov     dword ptr [ebx+0B0h],ecx                   ; "
5b93f956 8b4d14          mov     ecx,dword ptr [ebp+14h]
5b93f959 894320          mov     dword ptr [ebx+20h],eax                    ; store width into kdu_decompressor object
5b93f95c 8b45e0          mov     eax,dword ptr [ebp-20h]                    ; kdu_dims::pos.x
5b93f95f 89531c          mov     dword ptr [ebx+1Ch],edx                    ; store height into kdu_decompressor object
5b93f962 894318          mov     dword ptr [ebx+18h],eax                    ; store x position into kdu_decompressor object
5b93f965 8983b4000000    mov     dword ptr [ebx+0B4h],eax                   ; "
5b93f96b 8993b8000000    mov     dword ptr [ebx+0B8h],edx                   ; store height into kdu_decompressor object
```

After assigning the different dimensions retrieved from the code-stream object, the application will allocate an array of objects to store tile information based on the number of color components that were identified by the code-stream object. The number of components and the size of these object are passed to the function call at [16]. Afterwards at [17], the application will clear the array using the `memset` function.

```
npdf!CAPContent::Wrap+0xae1a2:
5b93f992 56              push    esi
5b93f993 57              push    edi
5b93f994 898bcc000000    mov     dword ptr [ebx+0CCh],ecx                  ; width of tile
5b93f99a 8b730c          mov     esi,dword ptr [ebx+0Ch]                   ; number of components
5b93f99d 8b4b34          mov     ecx,dword ptr [ebx+34h]                   ; stripe decompressor object
5b93f9a0 56              push    esi                                       ; count
5b93f9a1 6a08            push    8
5b93f9a3 6a50            push    50h                                       ; size of tile object
5b93f9a5 c6430a00        mov     byte ptr [ebx+0Ah],0
5b93f9a9 e812deffff      call    npdf!CAPContent::Wrap+0xabfd0 (5b93d7c0)  ; [16] allocate array of tile objects
...
5b93f9ae 8d0cb6          lea     ecx,[esi+esi*4]
5b93f9b1 8bf8            mov     edi,eax                                   ; result from allocation
5b93f9b3 c1e104          shl     ecx,4
5b93f9b6 51              push    ecx
5b93f9b7 6a00            push    0
5b93f9b9 57              push    edi                                       ; tile array
5b93f9ba e8b11c1d00      call    npdf!CAPContent::Wrap+0x27fe80 (5bb11670) ; [17] memset
5b93f9bf 897b10          mov     dword ptr [ebx+10h],edi
5b93f9c2 83c40c          add     esp,0Ch
```

After initializing the array, the application will enter the following loop. This loop will initialize the respective sub-sample tile for each component. At [18], the application will get the dimensions and store them into the tile object. At [19], the bit-depth will be retrieved by calling the `kdu_core::kdu_codestream::get_bit_depth` method and then also stored into the tile. Finally at [20], the sub-sampling information will be fetched directly into the tile object. After getting these parameters, the application will initialize a number of its properties prior to assigning them. One of these properties, at [21], will be later populated with the buffer that the tile is to be decoded into. After fetching the tile's dimensions, the application will then write these dimensions into the tile object. This loop will continue until there are no components left at [22].

```
npdf!CAPContent::Wrap+0xae1e5:
5b93f9d5 8b7310          mov     esi,dword ptr [ebx+10h]                   ; tile array
5b93f9d8 8d4d08          lea     ecx,[ebp+8]                               ; codestream
5b93f9db 03f0            add     esi,eax                                   ; offset into tile array
5b93f9dd 8d45ec          lea     eax,[ebp-14h]                             ; result to write dimensions into
5b93f9e0 6a01            push    1
5b93f9e2 50              push    eax
5b93f9e3 0f57c0          xorps   xmm0,xmm0
5b93f9e6 893e            mov     dword ptr [esi],edi
5b93f9e8 57              push    edi
5b93f9e9 0f1145ec        movups  xmmword ptr [ebp-14h],xmm0
5b93f9ed e82e38fdff      call    npdf!CAPContent::Wrap+0x81a30 (5b913220)  ; [18] kdu_core::kdu_codestream::get_dims
...
5b93f9f2 8b45f0          mov     eax,dword ptr [ebp-10h]
5b93f9f5 8d4d08          lea     ecx,[ebp+8]
5b93f9f8 6a00            push    0
5b93f9fa 6a01            push    1
5b93f9fc 894604          mov     dword ptr [esi+4],eax
5b93f9ff 8b45f8          mov     eax,dword ptr [ebp-8]
5b93fa02 57              push    edi
5b93fa03 894608          mov     dword ptr [esi+8],eax
5b93fa06 e82537fdff      call    npdf!CAPContent::Wrap+0x81940 (5b913130)  ; [19] kdu_core::kdu_codestream::get_bit_depth
5b93fa0b 89460c          mov     dword ptr [esi+0Ch],eax
...
5b93fa17 6a01            push    1
5b93fa19 8d4610          lea     eax,[esi+10h]                             ; subsample member of object
5b93fa1c 50              push    eax
5b93fa1d 57              push    edi
5b93fa1e 8d4d08          lea     ecx,[ebp+8]                               ; codestream object
5b93fa21 e87a3cfdff      call    npdf!CAPContent::Wrap+0x81eb0 (5b9136a0)  ; [20] kdu_core::kdu_codestream::get_subsampling
...
5b93fa26 6a01            push    1
5b93fa28 8d55ec          lea     edx,[ebp-14h]
5b93fa2b c7462000000000  mov     dword ptr [esi+20h],0                     ; horizontal stide
5b93fa32 52              push    edx
5b93fa33 c7461c00000000  mov     dword ptr [esi+1Ch],0                     ; projected stride
5b93fa3a c7461800000000  mov     dword ptr [esi+18h],0                     ; stripe height
5b93fa41 c7462c00000000  mov     dword ptr [esi+2Ch],0                     ; [21] buffer
5b93fa48 c74628ffffffff  mov     dword ptr [esi+28h],0FFFFFFFFh
5b93fa4f c7463000000000  mov     dword ptr [esi+30h],0
5b93fa56 c7463400000000  mov     dword ptr [esi+34h],0                     ; projected stripe height
...
5b93fa74 8b45f4          mov     eax,dword ptr [ebp-0Ch]                   ; tile height
5b93fa77 894638          mov     dword ptr [esi+38h],eax                   ; store height to tile object
5b93fa7a 8b431c          mov     eax,dword ptr [ebx+1Ch]
5b93fa7d 894648          mov     dword ptr [esi+48h],eax
5b93fa80 8b45dc          mov     eax,dword ptr [ebp-24h]                   ; y position
5b93fa83 89464c          mov     dword ptr [esi+4Ch],eax                   ; store y position to tile object
5b93fa86 c7463c00000000  mov     dword ptr [esi+3Ch],0
5b93fa8d 8b45f4          mov     eax,dword ptr [ebp-0Ch]                   ; image height
5b93fa90 894640          mov     dword ptr [esi+40h],eax                   ; store image height to tile object
...
npdf!CAPContent::Wrap+0xae2d1:
5b93fac1 8b450c          mov     eax,dword ptr [ebp+0Ch]
5b93fac4 47              inc     edi                                       ; next index
5b93fac5 83c050          add     eax,50h                                   ; offset into tile array
5b93fac8 c7464400000000  mov     dword ptr [esi+44h],0
5b93facf 89450c          mov     dword ptr [ebp+0Ch],eax
5b93fad2 3b7b0c          cmp     edi,dword ptr [ebx+0Ch]                   ; [22] loop while index is less than number of components
5b93fad5 0f8cfafeffff    jl      npdf!CAPContent::Wrap+0xae1e5 (5b93f9d5)
```

After initializing the tile array object, the application will return to the caller and execute the function at [23]. According to the headers from the Kakadu library, it is suspected that this function is named `kdu_stripe_decompressor::get_recommended_stripe_heights`. After determining the recommended stripe heights, the application will fetch the number of bits for each component at [24]. This value comes directly from the `ImageHeader` belonging to the image file. This value is used in a number of places to calculate the length of different aspects of the stripe decoding process. At [25], the number of bits per component for the current component is fetched and then rounded to a multiple of 8. This is done to convert the number of bits to a number of bytes that will be used per component. After this has been determined, the bytes per component is multiplied by the stripe height as per the array initialized by the mentioned call to `kdu_stripe_decompresesor:get_recommended_stripe_heights`. At [26], the application calculates the stride for each stripe. This value is first calculated by taking the width of the image as defined in the `ImageHeader` of the image. This width is then multiplied by the stripe height per component that was calculated by [25]. At [27], the application checks that the product of the stride and the stripe height per component is larger than 32-bits.

```
npdf!CAPCosStream::SetAttributesDict+0x14bc:
5b63390c ffb5c8f3ffff    push    dword ptr [ebp-0C38h]                       ; first get_recommended_stripe_heights array
5b633912 8d8d2ce0ffff    lea     ecx,[ebp-1FD4h]                             ; stripe decompressor object
5b633918 c645fc07        mov     byte ptr [ebp-4],7
5b63391c ffb5d4f7ffff    push    dword ptr [ebp-82Ch]                        ; get_recommended_stripe_heights array
5b633922 6800040000      push    400h                                        ; absolute maximum height
5b633927 6a08            push    8                                           ; preferred minimum height
5b633929 e802ad3000      call    npdf!CAPContent::Wrap+0xace40 (5b93e630)   ; [23]
...
npdf!CAPCosStream::SetAttributesDict+0x14de:
5b63392e 8b460c          mov     eax,dword ptr [esi+0Ch]                     ; [24] BitsPerComponent array from jpx_source
5b633931 8b8de0fbffff    mov     ecx,dword ptr [ebp-420h]                    ; pointer to kdu_dims for each component
5b633937 8b7e04          mov     edi,dword ptr [esi+4]                       ; number of components from jpx_source
5b63393a 8b00            mov     eax,dword ptr [eax]                         ; [25] ImageHeader.BPC + 1
5b63393c 8b490c          mov     ecx,dword ptr [ecx+0Ch]                     ; [26] ImageHeader.width
5b63393f 83c007          add     eax,7
5b633942 99              cdq
5b633943 83e207          and     edx,7                                       ; [25] Round ImageHeader.BPC + 1 to multiple of 8
5b633946 898d80e1ffff    mov     dword ptr [ebp-1E80h],ecx                   ; [26] Store ImageHeader.width as stride
5b63394c 8b8dc8f3ffff    mov     ecx,dword ptr [ebp-0C38h]                   ; [25] get_recommended_stripe_heights array
5b633952 03c2            add     eax,edx
5b633954 c1f803          sar     eax,3                                       ; [25] Divide bits per component by 8 to convert to bytes
5b633957 0faff8          imul    edi,eax
5b63395a 898570e1ffff    mov     dword ptr [ebp-1E90h],eax                   ; Store bytes per component
5b633960 0faf39          imul    edi,dword ptr [ecx]                         ; [25] Multiply bytes per component by stripe height
5b633963 83c8ff          or      eax,0FFFFFFFFh
5b633966 8b8d80e1ffff    mov     ecx,dword ptr [ebp-1E80h]                   [ [26] Load stride
5b63396c 33d2            xor     edx,edx
5b63396e f7f7            div     eax,edi                                     ; [27] check that stripe height per component is not larger than
32-bits
5b633970 3bc8            cmp     ecx,eax
5b633972 0f8710080000    ja      npdf!CAPCosStream::SetAttributesDict+0x1d38 (5b634188)
5b633978 64a12c000000    mov     eax,dword ptr fs:[0000002Ch]
5b63397e 0faff9          imul    edi,ecx                                     ; [26] multiply stride by stripe height per component
```

After the information about each tile for each component has been calculated, the application will branch to the following block of code. At [28], the application will execute the `GetSystemInfo` function. This function will fetch the page size from the system (4096) and then store it at [29]. This will later be used to allocate space for all of the tiles belonging to the each individual color component prior to decoding the tile’s stripes.

```
npdf!CAPCosStream::SetAttributesDict+0x1d59:
5b6341a9 680861cc5b      push    offset npdf!CAPCosObj::smEnumProc+0x1c (5bcc6108)
5b6341ae e894aa4d00      call    npdf!CAPContent::Wrap+0x27d457 (5bb0ec47)
5b6341b3 83c404          add     esp,4
5b6341b6 833d0861cc5bff  cmp     dword ptr [npdf!CAPCosObj::smEnumProc+0x1c (5bcc6108)],0FFFFFFFFh
5b6341bd 0f85def7ffff    jne     npdf!CAPCosStream::SetAttributesDict+0x1551 (5b6339a1)
...
5b6341c3 8d8508e0ffff    lea     eax,[ebp-1FF8h]
5b6341c9 c645fc08        mov     byte ptr [ebp-4],8
5b6341cd 50              push    eax
5b6341ce ff15a0f2b25b    call    dword ptr [npdf!CAPContent::Wrap+0x29dab0 (5bb2f2a0)]          ; [28] GetSystemInfo
...
5b6341d4 8b850ce0ffff    mov     eax,dword ptr [ebp-1FF4h]
5b6341da 680861cc5b      push    offset npdf!CAPCosObj::smEnumProc+0x1c (5bcc6108)
5b6341df a30461cc5b      mov     dword ptr [npdf!CAPCosObj::smEnumProc+0x18 (5bcc6104)],eax     ; [29] Store page size
```

After calculating the page size for the system, the application will then take the total tile size that was previously calculated at [30] and use it to calculate the number of pages that needs to be allocated to decode the tile. At [31], the stride value that was calculated is divided by the page size. If there was a remainder, the page size will then be rounded upwards. Before calling the `VirtualAlloc` function, the number of page will then be increased by one more. After the number of pages have been determined, they will be multiplied by the page size [32]. This will then be passed to `VirtualAlloc` at [33], and then stored into the tile for the current color component.

```
npdf!CAPCosStream::SetAttributesDict+0x1551:
5b6339a1 33d2            xor     edx,edx
5b6339a3 8bc7            mov     eax,edi                                                        ; [30] Stride for tile component
5b6339a5 f7350461cc5b    div     eax,dword ptr [npdf!CAPCosObj::smEnumProc+0x18 (5bcc6104)]     ; [31] Divide by page size
5b6339ab 6a04            push    4
5b6339ad 85d2            test    edx,edx
5b6339af ba00000000      mov     edx,0
5b6339b4 6800100000      push    1000h
5b6339b9 0f95c2          setne   dl
5b6339bc 03d0            add     edx,eax                                                        ; [31] Add one if there is a remainder
5b6339be 899578e1ffff    mov     dword ptr [ebp-1E88h],edx
5b6339c4 8d4201          lea     eax,[edx+1]                                                    ; [31] Add one more
5b6339c7 0faf050461cc5b  imul    eax,dword ptr [npdf!CAPCosObj::smEnumProc+0x18 (5bcc6104)]     ; [32] Multiply number of pages by page size
5b6339ce 50              push    eax
5b6339cf 6a00            push    0
5b6339d1 ff159cf2b25b    call    dword ptr [npdf!CAPContent::Wrap+0x29daac (5bb2f29c)]          ; [33] VirtualAlloc
5b6339d7 8bf8            mov     edi,eax
5b6339d9 89bd74e1ffff    mov     dword ptr [ebp-1E8Ch],edi                                      ; [33] Store into tile
```

After allocating the buffer for decoding the stripes for the tile, the application will then allocate a list in order to store the stride for each color component. This will be used to adjust the buffer when the application needs to move to the next tile for a given sub-sample. At [34], the application allocates enough space for 0x100 elements on the stack. When the list grows past this length, the application will then use dynamic memory to store the list. A similar list is allocated for a number of components that belong to the tile. Each of these lists will be populated later prior to decoding.

```
npdf!CAPCosStream::SetAttributesDict+0x1617:
5b633a67 8d85bcebffff    lea     eax,[ebp-1444h]
5b633a6d c785b4ebffff00000000 mov dword ptr [ebp-144Ch],0       ; Next available slot
5b633a77 8985b0ebffff    mov     dword ptr [ebp-1450h],eax      ; Pointer to elements of list
5b633a7d c785b8ebffff00010000 mov dword ptr [ebp-1448h],100h    ; [34] Length
...
5b633a87 ffb564e1ffff    push    dword ptr [ebp-1E9Ch]
5b633a8d 898578e1ffff    mov     dword ptr [ebp-1E88h],eax
5b633a93 8d8db0ebffff    lea     ecx,[ebp-1450h]                ; Pointer into list
5b633a99 ff7604          push    dword ptr [esi+4]
5b633a9c 8d8578e1ffff    lea     eax,[ebp-1E88h]
5b633aa2 c645fc0c        mov     byte ptr [ebp-4],0Ch
5b633aa6 50              push    eax
5b633aa7 8d8564e1ffff    lea     eax,[ebp-1E9Ch]
5b633aad 50              push    eax
5b633aae e88deaffff      call    npdf!CAPCosStream::SetAttributesDict+0xf0 (5b632540)
```

After allocating a number of lists, the application will finally enter a loop that is responsible for populating these lists with values relevant to the tile. This loop will iterate through the number of components writing any of the values previously calculated within the function for a given component into its given list. At [35], the application will first the number of bytes per component as previously calculated. This will then be multiplied at [36] by the number of components in order to determine the component stride for the application to move to the next sub-sample in a tile. This is then stored at the list base pointer at [36]. Next the tile width is is fetched from the tile's dimensions at [37]. This is also multiplied by the number of bytes per component and stored at the list base pointer at [37]. Similarly at [38], the horizontal stride is fetched, multiplied by the tile stride, and stored to a list. Lastly, a list containing the aggregate value of tile strides is populated at [39]. Each of these lists will be utilised by the stripe decoder during the sub-sample decoding process.

```
npdf!CAPCosStream::SetAttributesDict+0x16ba:
5b633b0a 8b8570e1ffff    mov     eax,dword ptr [ebp-1E90h]      ; [35] Load bytes per component
5b633b10 33ff            xor     edi,edi
...
npdf!CAPCosStream::SetAttributesDict+0x16c2:
5b633b12 8b4e04          mov     ecx,dword ptr [esi+4]          ; Number of components
5b633b15 0fafc8          imul    ecx,eax                        ; [36] Multiply number of components by number of bytes per component
5b633b18 8b85a4e7ffff    mov     eax,dword ptr [ebp-185Ch]
5b633b1e 890c90          mov     dword ptr [eax+edx*4],ecx      ; [36] Store to list for tile decoding
...
5b633b21 8b85e0fbffff    mov     eax,dword ptr [ebp-420h]       ; [37] Tile dimensions
5b633b27 8b480c          mov     ecx,dword ptr [eax+0Ch]        ; [37] kdu_dims::size.X
5b633b2a 0faf4e04        imul    ecx,dword ptr [esi+4]          ; [37] Multiply by number of bytes per component
...
5b633b2e 8b85b0ebffff    mov     eax,dword ptr [ebp-1450h]      ; [38] Horizontal stride
5b633b34 0faf8d70e1ffff  imul    ecx,dword ptr [ebp-1E90h]      ; [38] Multiply Horizontal stride by tile stride
5b633b3b 890c90          mov     dword ptr [eax+edx*4],ecx      ; [38] Store to another list for tile decoding
...
5b633b3e 8b85bcefffff    mov     eax,dword ptr [ebp-1044h]      ; [39] Aggregate list of tile stride
5b633b44 893c90          mov     dword ptr [eax+edx*4],edi      ; [39] Store to another list for tile decoding
...
5b633b47 42              inc     edx
5b633b48 8b8570e1ffff    mov     eax,dword ptr [ebp-1E90h]      ; Tile stride
5b633b4e 03f8            add     edi,eax                        ; [39] Adjust %edi by tile stride
5b633b50 3b5604          cmp     edx,dword ptr [esi+4]          ; Number of components
5b633b53 72bd            jb      npdf!CAPCosStream::SetAttributesDict+0x16c2 (5b633b12)
```

Once each of the lists have been initialized with all of the values necessary to perform stripe decoding, the application will load the allocation returned by `VirtualAlloc` into `%edi` at [4]. The recommended stripe heights will be requested again at [41], and then at [42] each of the lists will be passed to the function call at [42]. The function call at [42] is simply a wrapper and will pass all of its arguments onto the function call at [43].

```
npdf!CAPCosStream::SetAttributesDict+0x1705:
5b633b55 8bbd74e1ffff    mov     edi,dword ptr [ebp-1E8Ch]                  ; [40] Load allocation made from VirtualAlloc
5b633b5b b201            mov     dl,1
5b633b5d 0f1f00          nop     dword ptr [eax]
5b633b60 84d2            test    dl,dl
5b633b62 0f84ed040000    je      npdf!CAPCosStream::SetAttributesDict+0x1c05 (5b634055)
...
5b633b68 6a00            push    0
5b633b6a ffb5d4f7ffff    push    dword ptr [ebp-82Ch]
5b633b70 8d8d2ce0ffff    lea     ecx,[ebp-1FD4h]
5b633b76 6800040000      push    400h
5b633b7b 6a08            push    8
5b633b7d e8aeaa3000      call    npdf!CAPContent::Wrap+0xace40 (5b93e630)   ; [41] kdu_stripe_decompressor::get_recommended_stripe_heights
...
5b633b82 6a00            push    0
5b633b84 6a00            push    0
5b633b86 ff760c          push    dword ptr [esi+0Ch]                        ; array of bits per component from jpx_source
5b633b89 8d852ce0ffff    lea     eax,[ebp-1FD4h]                            ; stripe decompressor object
5b633b8f ffb5b0ebffff    push    dword ptr [ebp-1450h]                      ; horizontal stride
5b633b95 ffb5a4e7ffff    push    dword ptr [ebp-185Ch]                      ;
5b633b9b ffb5bcefffff    push    dword ptr [ebp-1044h]                      ; aggregate tile stride
5b633ba1 ffb5d4f7ffff    push    dword ptr [ebp-82Ch]                       ; recommended stripe heights
5b633ba7 57              push    edi                                        ; VirtualAlloc buffer
5b633ba8 50              push    eax
5b633ba9 e852180000      call    npdf!FDFOpenFromEmbedded+0x380 (5b635400)  ; [42] \
\
npdf!FDFOpenFromEmbedded+0x3ba:
5b63543a ff7528          push    dword ptr [ebp+28h]
5b63543d ff7524          push    dword ptr [ebp+24h]
5b635440 ff7520          push    dword ptr [ebp+20h]                        ; bits per component array
5b635443 ff751c          push    dword ptr [ebp+1Ch]                        ; horizontal strides
5b635446 ff7518          push    dword ptr [ebp+18h]
5b635449 ff7514          push    dword ptr [ebp+14h]                        ; aggregate tile stride
5b63544c ff7510          push    dword ptr [ebp+10h]                        ; recommended stripe heights
5b63544f ff750c          push    dword ptr [ebp+0Ch]                        ; VirtualAlloc buffer
5b635452 8b4d08          mov     ecx,dword ptr [ebp+8]                      ; stripe decompresssor object
5b635455 e856a03000      call    npdf!CAPContent::Wrap+0xadcc0 (5b93f4b0)   ; [43]
```

The beginning of the following function is responsible for copying data from each array by component into each tile. After checking the number of components at [44], the application will proceed to copy some of the arrays that were stored in the parameters into other variables that are located on the stack. These assignments are shown at [45]. After preparing all the variables, the application will begin to enter the loop at [46] where the `%esi` register is used an index into each of these arrays that were passed as parameters. This index represents the current color component that is being processed. Once inside the loop at [47], the `%ecx` register will be used to point to the current tile or sub-sample that is being populated. At [48], the first list item that is fetched is from the aggregated stride array. It's important to note that this stride is used to seek into the buffer that was allocated. During the decoding process, this is where stripes for a given tile will start being decoded at. After fetching it, it is added to the `VirtualAlloc` buffer and then written into the current tile. The projected stripe from the current component in the array is also copied into the tile at [49], as well as the rest of the fields at [50]. At [51], specifically, is the horizontal stride that is fetched. This element that is fetched from its array and copied into the tile was not used during the calculation of the buffer size. Due to this oversight combined how it is later used, we will show how it can be used to write outside the bounds of the prior mentioned allocation. After populating all of the tiles representing the stripe data for the current sub-sample, the stripe decompressor object will be passed to the method call at [52].

```
npdf!CAPContent::Wrap+0xadccf:
5b93f4bf 395f0c          cmp     dword ptr [edi+0Ch],ebx        ; [44] Number of components
5b93f4c2 0f8ef9000000    jle     npdf!CAPContent::Wrap+0xaddd1 (5b93f5c1)
...
5b93f4e6 8b5518          mov     edx,dword ptr [ebp+18h]        ; [45] horizontal stride array
5b93f4e9 89550c          mov     dword ptr [ebp+0Ch],edx        ; [45] store over a parameter
5b93f4ec 29450c          sub     dword ptr [ebp+0Ch],eax        ; always subtracts zero when decoding stripes
5b93f4ef 8955ec          mov     dword ptr [ebp-14h],edx
5b93f4f2 8b551c          mov     edx,dword ptr [ebp+1Ch]        ; [45] bits per component array
5b93f4f5 895518          mov     dword ptr [ebp+18h],edx        ; [45] store over a parameter
5b93f4f8 294518          sub     dword ptr [ebp+18h],eax
...
npdf!CAPContent::Wrap+0xadd0f:
5b93f4ff 8bf0            mov     esi,eax                        ; [46] uses as index when fetching from lists
5b93f501 8b5510          mov     edx,dword ptr [ebp+10h]
..
npdf!CAPContent::Wrap+0xadd14:
5b93f504 034f10          add     ecx,dword ptr [edi+10h]        ; [47] adjust %ecx to current tile in array
5b93f507 c7412800000000  mov     dword ptr [ecx+28h],0
5b93f50e 85d2            test    edx,edx
5b93f510 7504            jne     npdf!CAPContent::Wrap+0xadd26 (5b93f516)
...
5b93f516 8b45fc          mov     eax,dword ptr [ebp-4]          ; [48] fetch aggregated stride array
5b93f519 8b0430          mov     eax,dword ptr [eax+esi]        ; [48] fetch the actual aggregated stride
5b93f51c 034508          add     eax,dword ptr [ebp+8]          ; [48] position aggregated stride as an offst into allocated buffer
5b93f51f 837d2000        cmp     dword ptr [ebp+20h],0
5b93f523 89412c          mov     dword ptr [ecx+2Ch],eax        ; [48] store to tile object
5b93f526 7504            jne     npdf!CAPContent::Wrap+0xadd3c (5b93f52c)
...
5b93f52e 837d1400        cmp     dword ptr [ebp+14h],0
5b93f532 8b5510          mov     edx,dword ptr [ebp+10h]
5b93f535 894130          mov     dword ptr [ecx+30h],eax
5b93f538 8b45f4          mov     eax,dword ptr [ebp-0Ch]        ; [49] projected stripe height
5b93f53b 8b0430          mov     eax,dword ptr [eax+esi]        ; [49] fetch stripe height from array
5b93f53e 894134          mov     dword ptr [ecx+34h],eax        ; [49] write to current tile object
5b93f541 7510            jne     npdf!CAPContent::Wrap+0xadd63 (5b93f553)
...
5b93f553 8b45f0          mov     eax,dword ptr [ebp-10h]        ; [50] grab the list for the aggregated number of components
5b93f556 8b1430          mov     edx,dword ptr [eax+esi]        ; [50] fetch aggregated value from list
...
5b93f559 837dec00        cmp     dword ptr [ebp-14h],0
5b93f55d 89511c          mov     dword ptr [ecx+1Ch],edx        ; [50] update field in tile object
5b93f560 7508            jne     npdf!CAPContent::Wrap+0xadd7a (5b93f56a)
...
5b93f562 8b4108          mov     eax,dword ptr [ecx+8]          ; [50] grab tile object width
5b93f565 0fafc2          imul    eax,edx                        ; [50] multiply tile width by aggregated number of components
5b93f568 eb06            jmp     npdf!CAPContent::Wrap+0xadd80 (5b93f570)
...
5b93f56a 8b450c          mov     eax,dword ptr [ebp+0Ch]        ; [51] grab horizontal stride array
5b93f56d 8b0430          mov     eax,dword ptr [eax+esi]        ; [51] fetch horizontal stride element
...
5b93f570 837d1c00        cmp     dword ptr [ebp+1Ch],0
5b93f574 894118          mov     dword ptr [ecx+18h],eax        ; [51] update tile object with horizontal stride
5b93f577 7507            jne     npdf!CAPContent::Wrap+0xadd90 (5b93f580)
...
5b93f580 8b4518          mov     eax,dword ptr [ebp+18h]        ; [50] bits per component array
5b93f583 8b0430          mov     eax,dword ptr [eax+esi]        ; [50] fetch element from bits per component array
5b93f586 894120          mov     dword ptr [ecx+20h],eax        ; [50] update tile object with bits per component
5b93f589 c6412400        mov     byte ptr [ecx+24h],0
...
5b93f5a7 8b4df8          mov     ecx,dword ptr [ebp-8]
5b93f5aa 43              inc     ebx                            ; current tile index
5b93f5ab 8b5510          mov     edx,dword ptr [ebp+10h]
5b93f5ae 83c150          add     ecx,50h                        ; next tile to be read from
5b93f5b1 83c604          add     esi,4                          ; next pointer
5b93f5b4 894df8          mov     dword ptr [ebp-8],ecx
5b93f5b7 3b5f0c          cmp     ebx,dword ptr [edi+0Ch]        ; number of components
5b93f5ba 0f8c44ffffff    jl      npdf!CAPContent::Wrap+0xadd14 (5b93f504)
...
npdf!CAPContent::Wrap+0xaddd1:
5b93f5c1 ff7524          push    dword ptr [ebp+24h]
5b93f5c4 8bcf            mov     ecx,edi                        ; [52] stripe decompressor object
5b93f5c6 e845faffff      call    npdf!CAPContent::Wrap+0xad820 (5b93f010)
```

The following function is the final stage before the application begins to decode stripes related to the current sub-sample or component. To prepare for the decoding, a number of fields are copied from the current object into the stack at [53]. The last thing the application must do is to copy fields from the tiles initialized by the parent function, into a list of the decoding tile object which is used to actually decode the sub-samples. At [54], the application sets the `%edx` register to point to the array of decoding tiles which will be the target of this copying. Each field will then be copied at [55] into the decoding object. After copying each field, at [56], the offset into the buffer that was allocated with `VirtualAlloc` will be read from the tile for the current component. This will then immediately get written into the tile decoding object. For each iteration of this loop, at [57] the offsets into both the source tile object and destination tile object is adjusted. This is done for the number of tiles that were in the array. When the copying is done, the application can finally use the stripe decompressor object to begin decoding stripes for each sub-sample. This is done at [58].

```
npdf!CAPContent::Wrap+0xad870:
5b93f060 8b4620          mov     eax,dword ptr [esi+20h]        ; [53] stripe width
5b93f063 8b5614          mov     edx,dword ptr [esi+14h]        ; stripe X
5b93f066 8b7e18          mov     edi,dword ptr [esi+18h]        ; stripe Y
5b93f069 8b5e24          mov     ebx,dword ptr [esi+24h]        ; tile decoding object
5b93f06c c645f200        mov     byte ptr [ebp-0Eh],0
5b93f070 8955e4          mov     dword ptr [ebp-1Ch],edx        ; store X
5b93f073 897ddc          mov     dword ptr [ebp-24h],edi        ; store Y
5b93f076 8945e8          mov     dword ptr [ebp-18h],eax        ; store width
5b93f079 85c0            test    eax,eax
5b93f07b 0f8ed6020000    jle     npdf!CAPContent::Wrap+0xadb67 (5b93f357)
...
npdf!CAPContent::Wrap+0xad924:
5b93f114 83bb8400000000  cmp     dword ptr [ebx+84h],0
5b93f11b 8b7e10          mov     edi,dword ptr [esi+10h]
5b93f11e 0f8e8d000000    jle     npdf!CAPContent::Wrap+0xad9c1 (5b93f1b1)
...
npdf!CAPContent::Wrap+0xad940:
5b93f130 8b9388000000    mov     edx,dword ptr [ebx+88h]        ; [54] tile object array
5b93f136 8b4f18          mov     ecx,dword ptr [edi+18h]        ; projected stripe height
5b93f139 03d0            add     edx,eax
5b93f13b 3b0a            cmp     ecx,dword ptr [edx]            ; maximum stripe height
5b93f13d 0f4f0a          cmovg   ecx,dword ptr [edx]
5b93f140 894a14          mov     dword ptr [edx+14h],ecx        ; [55] copy into tile decoding object
5b93f143 8b07            mov     eax,dword ptr [edi]            ; read from tile object
5b93f145 894218          mov     dword ptr [edx+18h],eax        ; [55] copy into tile decoding object
5b93f148 8945d4          mov     dword ptr [ebp-2Ch],eax
5b93f14b 8b47fc          mov     eax,dword ptr [edi-4]          ; read recommended stripe height
5b93f14e 89421c          mov     dword ptr [edx+1Ch],eax        ; [55] copy into tile decoding object
5b93f151 8b4704          mov     eax,dword ptr [edi+4]          ; read horizontal stride
5b93f154 894220          mov     dword ptr [edx+20h],eax        ; [55] copy into tile decoding object
5b93f157 8a4708          mov     al,byte ptr [edi+8]            ; read flag
5b93f15a 884224          mov     byte ptr [edx+24h],al          ; [55] copy into tile decoding object
5b93f15d 8b470c          mov     eax,dword ptr [edi+0Ch]        ; read style
5b93f160 894228          mov     dword ptr [edx+28h],eax        ; [55] copy into tile decoding object
5b93f163 8945d0          mov     dword ptr [ebp-30h],eax
5b93f166 8b4710          mov     eax,dword ptr [edi+10h]        ; [56] read current VirtualAlloc buffer
5b93f169 89422c          mov     dword ptr [edx+2Ch],eax        ; [56] copy into tile decoding object
5b93f16c 8b4f14          mov     ecx,dword ptr [edi+14h]
5b93f16f 8945cc          mov     dword ptr [ebp-34h],eax
...
5b93f184 8b420c          mov     eax,dword ptr [edx+0Ch]
5b93f187 46              inc     esi
5b93f188 0faf45d4        imul    eax,dword ptr [ebp-2Ch]
5b93f18c 83c750          add     edi,50h                        ; [57] adjust offset into next tile object
5b93f18f 8b4dd0          mov     ecx,dword ptr [ebp-30h]
5b93f192 83e103          and     ecx,3
5b93f195 d3e0            shl     eax,cl
5b93f197 0345cc          add     eax,dword ptr [ebp-34h]
5b93f19a 89422c          mov     dword ptr [edx+2Ch],eax
5b93f19d 8b45e0          mov     eax,dword ptr [ebp-20h]
5b93f1a0 83c040          add     eax,40h                        ; [57] adjust offset into next tile decoding object
5b93f1a3 8945e0          mov     dword ptr [ebp-20h],eax
5b93f1a6 3bb384000000    cmp     esi,dword ptr [ebx+84h]        ; [57] check against number of tiles
5b93f1ac 7c82            jl      npdf!CAPContent::Wrap+0xad940 (5b93f130)
...
npdf!CAPContent::Wrap+0xad9c1:
5b93f1b1 ff7638          push    dword ptr [esi+38h]
5b93f1b4 8bcb            mov     ecx,ebx                        ; [58] stripe decompressor object
5b93f1b6 e8a5f7ffff      call    npdf!CAPContent::Wrap+0xad170 (5b93e960)
5b93f1bb 84c0            test    al,al
5b93f1bd 0f8414010000    je      npdf!CAPContent::Wrap+0xadae7 (5b93f2d7)
```

Once the application is inside the function, the application will begin decoding the data that the parent functions have setup for the strip decoder object. This starts out by grabbing the tile array length at [59]. Once this has been performed, the application will enter two loops which will iterate through all of the stripes that are to be decoded into the `VirtualAlloc` buffer that was prior mentioned. The innermost loop will use the tile index that is stored in the `%esi` register. This is done at [60], by taking the current index and multiplying it by the size of the tile. When positioned relative to the tile decoding array, this will allow the application to start acting on the given sub-sample tile. At [70], the application will grab the maximum and projected stripe heights in order to shift them due to the sizes being inclusive. At [71], the value of “1” is subtracted from then and then they are written back into the tile decoding object. The next thing the application can do is to fetch an object from one of the properties of the tile decoding object. It is suspected by the author that this is the `kd_core_local::kd_multi_synthesis` object from analyzing the headers of the Kakadu library. This object is used to call the `kd_core_local::kd_multi_synthesis::get_line` method at [72]. This particular method returns a `kdu_core::kdu_line_buf` object which contains the data that is to be decoded. Once the line buffer object has been fetched, the application will then load a number of tile decoding attributes into registers at [73]. Most importantly at [74] is the stride that will be used to adjust the pointer used for writing at each iteration of this loop.

```
npdf!CAPContent::Wrap+0xad17e:
5b93e96e 8b8784000000    mov     eax,dword ptr [edi+84h]        ; [59]
5b93e974 33f6            xor     esi,esi
5b93e976 c645fc01        mov     byte ptr [ebp-4],1
5b93e97a 8975e4          mov     dword ptr [ebp-1Ch],esi
5b93e97d b101            mov     cl,1
5b93e97f 884dfe          mov     byte ptr [ebp-2],cl
5b93e982 85c0            test    eax,eax
5b93e984 0f8e6f060000    jle     npdf!CAPContent::Wrap+0xad809 (5b93eff9)
...
npdf!CAPContent::Wrap+0xad1a0:
5b93e990 0fb645fe        movzx   eax,byte ptr [ebp-2]
5b93e994 8bde            mov     ebx,esi                        ; [60] current decoding tile index
5b93e996 c1e306          shl     ebx,6                          ; [60] multiply by the decoding tile size
5b93e999 33c9            xor     ecx,ecx
5b93e99b 039f88000000    add     ebx,dword ptr [edi+88h]        ; [60] add to decoding tile array
5b93e9a1 895df8          mov     dword ptr [ebp-8],ebx
5b93e9a4 8b13            mov     edx,dword ptr [ebx]            ; [70] maximum stripe height
5b93e9a6 85d2            test    edx,edx
5b93e9a8 0f4ec8          cmovle  ecx,eax
5b93e9ab 884dfe          mov     byte ptr [ebp-2],cl
5b93e9ae 8b4b14          mov     ecx,dword ptr [ebx+14h]        ; [70] projected stripe height
5b93e9b1 85c9            test    ecx,ecx
5b93e9b3 0f8417060000    je      npdf!CAPContent::Wrap+0xad7e0 (5b93efd0)
...
5b93e9c9 ff7508          push    dword ptr [ebp+8]
5b93e9cc 8d42ff          lea     eax,[edx-1]                    ; [71] maximum stripe height - 1
5b93e9cf 8903            mov     dword ptr [ebx],eax            ; [71] write back into decoding tile object
5b93e9d1 8d41ff          lea     eax,[ecx-1]                    ; [71] projected stripe height - 1
5b93e9d4 894314          mov     dword ptr [ebx+14h],eax        ; [71] write back into decoding tile object
...
5b93e9d7 8b4338          mov     eax,dword ptr [ebx+38h]
5b93e9da 014310          add     dword ptr [ebx+10h],eax
5b93e9dd 8b4328          mov     eax,dword ptr [ebx+28h]
5b93e9e0 8b4f04          mov     ecx,dword ptr [edi+4]          ; kd_core_local::kd_multi_synthesis object
5b93e9e3 83e003          and     eax,3
5b93e9e6 8945dc          mov     dword ptr [ebp-24h],eax
5b93e9e9 56              push    esi
5b93e9ea 8b01            mov     eax,dword ptr [ecx]
5b93e9ec ff5010          call    dword ptr [eax+10h]            ; [72] kd_core_local::kd_multi_synthesis::get_line
5b93e9ef 8bf8            mov     edi,eax
...
5b93e9f1 8b4330          mov     eax,dword ptr [ebx+30h]
5b93e9f4 83e003          and     eax,3
5b93e9f7 897de8          mov     dword ptr [ebp-18h],edi        ; [72] store kdu_core::kdu_line_buf
...
5b93ea08 8b5318          mov     edx,dword ptr [ebx+18h]        ; [73] horizontal stride
5b93ea0b 8b4320          mov     eax,dword ptr [ebx+20h]        ; [73] projected stride
5b93ea0e 8b4b34          mov     ecx,dword ptr [ebx+34h]        ; [73] tile array bit depth
5b93ea11 8b732c          mov     esi,dword ptr [ebx+2Ch]        ; [73] VirtualAlloc buffer
5b93ea14 8955f4          mov     dword ptr [ebp-0Ch],edx        ; [74] store stride into variable used as iterator
```

When decoding a line buffer with the provided proof-of-concept, the following loop will be executed. This loop is responsible for consuming a stripe and decoding it at the offset of the `VirtualAlloc` buffer that was stored into the tile decoding object earlier. At [75], the current `kdu_core::kdu_line_buf` will be fetched. The loop will continue for up to the number of bytes specified as the horizontal stride. At [76], the byte for the sub-sample will be decoded into the buffer, and for every iteration the adjustment of the write pointer at [77] will occur.

```
npdf!CAPContent::Wrap+0xad3d0:
5b93ebc0 f30f1007        movss   xmm0,dword ptr [edi]
5b93ebc4 8b5de8          mov     ebx,dword ptr [ebp-18h]            ; [75] kdu_core::kdu_line_buf
5b93ebc7 f30f59c1        mulss   xmm0,xmm1
5b93ebcb f30f2cc0        cvttss2si eax,xmm0
5b93ebcf 2bd8            sub     ebx,eax
5b93ebd1 f7c3000000ff    test    ebx,0FF000000h
5b93ebd7 7414            je      npdf!CAPContent::Wrap+0xad3fd (5b93ebed)
5b93ebd9 85db            test    ebx,ebx
5b93ebdb c745ec00000000  mov     dword ptr [ebp-14h],0
5b93ebe2 b8ffffff00      mov     eax,0FFFFFFh
5b93ebe7 0f4845ec        cmovs   eax,dword ptr [ebp-14h]
5b93ebeb 8bd8            mov     ebx,eax
5b93ebed d3fb            sar     ebx,cl
5b93ebef 4a              dec     edx
5b93ebf0 881e            mov     byte ptr [esi],bl                  ; [76] write decoded byte
5b93ebf2 83c704          add     edi,4
5b93ebf5 0375f4          add     esi,dword ptr [ebp-0Ch]            ; [77] iterator
5b93ebf8 85d2            test    edx,edx
5b93ebfa 7fc4            jg      npdf!CAPContent::Wrap+0xad3d0 (5b93ebc0)
```

When the horizontal stride for the `kdu_core::kdu_line_buf` has been decoded, the inner loop will continue with the following code. It is this continuation block that that will adjust the iterator outside the bounds of the `VirtualAlloc` buffer for the current decoding tile. At [78] the application will load a number of values so that when it resumes the loop at the top of the function, the registers will be of the current tile that is being processed. At [79], the application will load the pointer into the `VirtualAlloc` buffer that is to be decoded into. Once these values are loaded, the application will continue the loop at [80] by loading the tile decoding object back into the `%edx` register. At [81], the application will adjust the pointer into the `VirtualAlloc` buffer that is being decoded into by the recommended stripe height, and then continue execution. The loop will only terminate once the current index is greater or equal to the tile array count that is highlighted at [82].

```
npdf!CAPContent::Wrap+0xad54c:
5b93ed3c 837de000        cmp     dword ptr [ebp-20h],0
5b93ed40 0f8476020000    je      npdf!CAPContent::Wrap+0xad7cc (5b93efbc)
5b93ed46 8b4328          mov     eax,dword ptr [ebx+28h]
5b93ed49 85c0            test    eax,eax
5b93ed4b 0f859d000000    jne     npdf!CAPContent::Wrap+0xad5fe (5b93edee)
5b93ed51 8b4320          mov     eax,dword ptr [ebx+20h]       ; [78] horizontal stride
5b93ed54 8b7318          mov     esi,dword ptr [ebx+18h]       ; [78] projetcted tile stride
5b93ed57 8b7b04          mov     edi,dword ptr [ebx+4]         ; [78] maximum stripe width
5b93ed5a 8b532c          mov     edx,dword ptr [ebx+2Ch]       ; [79] virtual alloc buffer
5b93ed5d 8945e0          mov     dword ptr [ebp-20h],eax
5b93ed60 8b4330          mov     eax,dword ptr [ebx+30h]
5b93ed63 83fe02          cmp     esi,2
5b93ed66 0f8c50020000    jl      npdf!CAPContent::Wrap+0xad7cc (5b93efbc)
...
npdf!CAPContent::Wrap+0xad7cc:
5b93efbc 8b55f8          mov     edx,dword ptr [ebp-8]         ; [80] tile decoding object
5b93efbf 8b4ddc          mov     ecx,dword ptr [ebp-24h]
5b93efc2 8b75e4          mov     esi,dword ptr [ebp-1Ch]       ; [80] tile decoding array index
5b93efc5 8b7dd8          mov     edi,dword ptr [ebp-28h]       ; stripe decoding object
5b93efc8 8b421c          mov     eax,dword ptr [edx+1Ch]       ; recommended stripe height
5b93efcb d3e0            shl     eax,cl
5b93efcd 01422c          add     dword ptr [edx+2Ch],eax       ; [81] adjust buffer by recommended stripe height
5b93efd0 8b8784000000    mov     eax,dword ptr [edi+84h]       ; tile array count
5b93efd6 46              inc     esi                           ; increase tile array index
5b93efd7 8975e4          mov     dword ptr [ebp-1Ch],esi       ; [82] current tile array index
5b93efda 3bf0            cmp     esi,eax                       ; [82] check index is less than tile array count
5b93efdc 0f8caef9ffff    jl      npdf!CAPContent::Wrap+0xad1a0 (5b93e990)
```

Due to the application trusting the horizontal stride while decoding sample data from the `kdu_core::kdu_line_buf` object, each iteration of the decoding loop has a change of pushing the pointer that is being written past the end of the `VirtualAlloc` buffer that was allocated for decoding sub-sample tile. Due to this, an out-of-bounds write may occur which can cause memory corruption. Controlled memory corruptions can lead to code execution under the context of the application.

Crash Information

When opening up the provided proof-of-concept in the application, the following crash will occur.

```
(27b8.1a30): Access violation - code c0000005 (first/second chance not available)
eax=00000000 ebx=00000080 ecx=00000010 edx=0000000f esi=20f8c007 edi=304b2d80
eip=5b93ebf0 esp=0118bd94 ebp=0118bdc8 iopl=0         nv up ei pl nz ac pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00210217
npdf!CAPContent::Wrap+0xad400:
5b93ebf0 881e            mov     byte ptr [esi],bl          ds:0023:20f8c007=??
```

As described in the proof-of-concept, the size of the stripe being decoded is being set to 0x10.

```
0:000> r edx
edx=0000000f
```

Each iteration of the outer loop in the decoder function is a multiple of 4-pages in order to skip past the guard page.

```
0:000> dc @ebp-8 L4
0118bdc0  2fb0efc0 00000400 0118be3c 5b93f1bb  .../....<......[
0:000> dc @$p+1c L8
2fb0efdc  00004002 00000008 00000000 00000000  .@..............
2fb0efec  20f8c007 00000000 00000001 00000001  ... ............
```

At the current address, there is currently nothing mapped as this was debugged with gflags set to +hpa (full page heap).

```
0:000> dc @esi L10
20f8c007  ???????? ???????? ???????? ????????  ????????????????
20f8c017  ???????? ???????? ???????? ????????  ????????????????
20f8c027  ???????? ???????? ???????? ????????  ????????????????
20f8c037  ???????? ???????? ???????? ????????  ????????????????
```

If we seek back the size of the iterator, the previous write was within the bounds of the `VirtualAlloc` buffer.

```
0:000> dc @esi-4002 L10
20f88005  00800080 00800080 00800080 00800080  ................
20f88015  00800080 00800080 00800080 00800080  ................
20f88025  00000000 00000000 00000000 00000000  ................
20f88035  00000000 00000000 00000000 00000000  ................
```

The base addresses of the libraries in this report.

```
0:000> lm m npdf
Browse full module list
start    end        module name
5b560000 5bfa7000   npdf       (export symbols)       npdf.dll
011c0000 01a41000   NitroPDF   (deferred)
```

Timeline

2020-05-06 - Vendor Disclosure

2020-09-01 - Vendor Patched

2020-09-15 - Public Release

CREDIT

Discovered a member of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT

NEXT REPORT

TALOS-2020-1036

TALOS-2020-1063