Talos Vulnerability Report

TALOS-2021-1427

Blackmagic Design DaVinci Resolve R3D DPDecoder Service frame parsing uninitialized uuid object vulnerability

DECEMBER 20, 2021

CVE NUMBER

CVE-2021-40418

Summary

When parsing a file that is submitted to the DPDecoder service as a job, the R3D SDK will mistakenly skip over the assignment of a property containing an object referring to a UUID that was parsed from a frame within the video container. Upon destruction of the object that owns it, the uninitialized member will be dereferenced and then destroyed using the object's virtual destructor. Due to the object property being uninitialized, this can result in dereferencing an arbitrary pointer for the object's virtual method table, which can result in code execution under the context of the application.

Tested Versions

Blackmagic Design DaVinci Resolve 17.3.1.0005

Product URLs

DaVinci Resolve - http://www.blackmagicdesign.com/products/davinciresolve

CVSSv3 Score

9.8 - CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

CWE

CWE-457 - Use of Uninitialized Variable

Details

DaVinci Resolve is a non-linear video editing application available for a variety of platforms. In order to enable both professionals and amateurs to work with their media, it combines tools for performing editing, color correction, cinematic special effects and motion graphics, and even audio post production within the same application. There is also support for various types of equipment, 3rd-party plugins, and storage.

After the `DPDecoder` service has successfully bound to a port, the service will enter the following loop in order to process packets containing information about decoding jobs that are submitted by the sender. Inside this loop, the server will continuously read a string from the socket at [1]. This string represents the command that is sent by the client. At [2], the length of the string read from the socket is checked in a global. If it matches, then the string will be compared against the command string, "DECODE". After verifying that it matches, the server will call the method at [3] in order to handle the "DECODE" command.

```
0x14001c930: loop_client_1c930:
0x14001c930:        mov [rbp+var_40], r15
0x14001c934:        mov [rbp+var_38], 0
0x14001c93b:        mov [rbp+var_34], 1
0x14001c93f:        mov rax, [r15+BtLock.p_vtable_0]
0x14001c942:        mov dl, 1
0x14001c944:        mov rcx, r15
0x14001c947:        call qword ptr [rax+10h]
...
0x14001c950: loc_14001C950:
0x14001c950:        mov [rbp+lv_readBuffer_30.v_length_10], 0
0x14001c958:        mov [rbp+lv_readBuffer_30.v_size_18], 0Fh
0x14001c960:        mov byte ptr [rbp+lv_readBuffer_30.p_contents_0], 0
0x14001c964:        xor r8d, r8d                                // timeout
0x14001c967:        lea rdx, [rbp+lv_readBuffer_30]             // result string
0x14001c96b:        mov rcx, [rdi+REDDecoder.p_netSocket_40]    // object
0x14001c96f:        call NetSocket::readstring_13ca0            // [1] read a string from the network socket
0x14001c974:        test al, al
0x14001c976:        jz loc_14001CC49
...
0x14001c9c4: try_command(DECODE)_1c9c4:
0x14001c9c4:        lea rdx, str.DECODE_44038                   // string "DECODE"
0x14001c9cb:        cmp cs:gvd_44050, 10h
0x14001c9d3:        cmovnb rdx, cs:str.DECODE_44038             // string2
0x14001c9db:        lea rcx, [rbp+lv_readBuffer_30]             // string1 from packet
0x14001c9df:        cmp r14, 10h
0x14001c9e3:        cmovnb rcx, rsi
0x14001c9e7:        cmp rbx, cs:gv_strlen(DECODE)_44048         // [2] check length of string against length from packet
0x14001c9ee:        jnz short loc_14001CA09
...
0x14001c9f0:        mov r8, rbx                                 // length
0x14001c9f3:        call memcmp                                 // [2] compare string1 and string2
0x14001c9f8:        test eax, eax
0x14001c9fa:        jnz short loc_14001CA09
0x14001c9fc:        mov rcx, rdi
0x14001c9ff:        call REDDecoder::handler(DECODE)_1a400      // [3] call function to handle "DECODE" command
0x14001ca04:        jmp loc_14001CBF9
...
0x14001cd0f:        lea rcx, [rdi+58h]
0x14001cd13:        call BtEvent::method_ee20
0x14001cd18:        cmp qword ptr [rdi+40h], 0
0x14001cd1d:        jnz loop_client_1c930
```

Inside the method that handles the “DECODE” command, the following code will be executed. At [4], the method will first read a string from the socket. Afterwards, a number of integers containing the “header mode”, “frame number”, “decode mode”, “bits per pixel”, etc. are read from the socket and stored to the stack frame. After reading these values from the socket, at [5] an object is allocated and constructed. Afterwards, the function call at [6] will proceed to read data from the socket into the object that was constructed, and then read a string into its parameter before returning. At [7], the method will finally call a constructor for an object using the R3D SDK.

```
0x14001a442:        mov byte ptr [r11+lv_stringBuffer_b0.p_contents_0], r14b
0x14001a449:        lea r8d, [r14+5]                            // av_timeout_3
0x14001a44d:        lea rdx, [r11+lv_stringBuffer_b0]           // ap_resultString_2
0x14001a454:        mov rcx, [rcx+40h]                          // this
0x14001a458:        call NetSocket::readstring_13ca0            // [4] read a string from the socket
0x14001a45d:        test al, al
0x14001a45f:        jz raise_errorReceivingData_1ac8d
...
0x14001a60a:        mov ecx, 188h
0x14001a60f:        call operator new(unsigned __int64)         // [5] allocate an object
0x14001a614:        mov [rsp+278h+lp_stringBuffer_1f8], rax
0x14001a61c:        test rax, rax
0x14001a61f:        jz short loc_14001A62E
0x14001a621:        mov rcx, rax
0x14001a624:        call object(212d0)::constructor_212d0       // [5] construct it
0x14001a629:        mov rbx, rax
0x14001a62c:        jmp short loc_14001A631
...

0x14001a642: read_object_1a642:
0x14001a642:        mov [rsp+278h+lp_object_180], rbx
0x14001a64a:        mov r9d, 5                                  // timeout
0x14001a650:        lea r8, [rsp+278h+lv_stringBuffer_90]       // string buffer
0x14001a658:        mov rdx, rbx                                // destination object
0x14001a65b:        mov rcx, [rsi+REDDecoder.p_netSocket_40]    // this
0x14001a65f:        call sub_140013650                          // [6] read data from socket into object and a string
0x14001a664:        test al, al
0x14001a666:        jz raise_errorReceivingData_1aded
...
0x14001a66c: call_metadataApiObject_1a66c:
0x14001a66c:        lea rcx, [rsp+278h+lv_metadataObject_1d8]   // this
0x14001a674:        call sub_140021050                          // [7] call constructor from R3D SDK
0x14001a679:        nop
0x14001a67a:        lea rdx, [rsp+278h+lv_stringBuffer_d0]
0x14001a682:        cmp [rsi+REDDecoder.vb_38], 0
0x14001a686:        jz decode_frames_1a72b
```

After constructing the object using the R3D SDK, the strings that were read from the socket will be passed to the construct for an object at [8]. One of these strings contains the filename that the client is asking the server to decode. A user can specify any path, as this string allows for a user to decode a file that has been served remotely via an SMB share. Afterwards at [9], all of the fields that were read from the socket, as well as the object that was read, will be passed to the method call at [9] to construct a container for holding the clip and to pass the filename to a method from the R3D SDK.

```
0x14001a72b: decode_frames_1a72b:
0x14001a72b:        lea rax, [rsp+278h+lv_stringBuffer_130]
0x14001a733:        mov [rsp+278h+lp_stringBuffer_1f8], rax
0x14001a73b:        lea rcx, [rsp+278h+lv_stringBuffer_130]     // this
0x14001a743:        call sub_140014430                          // [8] copy string into an object
0x14001a748:        mov rdi, rax
...
0x14001a74b:        lea rdx, [rsp+278h+lv_stringBuffer_b0]          // source
0x14001a753:        lea rcx, [rsp+278h+lv_stringBuffer_110]     // this
0x14001a75b:        call sub_140014430                          // [8] copy string into an object
0x14001a760:        mov rdx, rax                                // ap_filename_2
...
0x14001a763:        lea rax, [rsp+278h+lv_metadataObject_1d8]
0x14001a76b:        mov [rsp+60h], rax                          // ap_metadataObject_13
0x14001a770:        lea rax, [rsp+278h+lv_stringBuffer_90]
0x14001a778:        mov [rsp+58h], rax                          // ap_string_12
0x14001a77d:        mov [rsp+50h], rbx                          // ap_readerObject_11
0x14001a782:        mov [rsp+48h], rdi                          // ap_string_10
0x14001a787:        movzx ecx, [rsp+278h+lvb_hasPalette_208]
0x14001a78c:        mov [rsp+40h], cl                           // avb_hasPalette_9
0x14001a790:        movss xmm0, [rsp+278h+lvS_1f0]
0x14001a799:        movss dword ptr [rsp+38h], xmm0             // avS_8
0x14001a79f:        mov ecx, [rsp+278h+lvd_HDRtype_1ec]
0x14001a7a6:        mov [rsp+30h], ecx                          // avd_HDRtype_7
0x14001a7aa:        mov eax, [rsp+278h+lvd_bpp_1e8]
0x14001a7b1:        mov [rsp+28h], eax                          // avd_bpp_6
0x14001a7b5:        mov eax, [rsp+278h+lvd_decodeMode_204]
0x14001a7b9:        mov [rsp+20h], eax                          // av_decodeMode_5
0x14001a7bd:        mov r9d, [rsp+278h+lvd_frameNumber_200]     // avd_frameNumber_4
0x14001a7c2:        mov r8d, [rsp+278h+lvd_hdrMode_1e4]         // avd_3
0x14001a7ca:        mov rcx, rsi                                // ap_this_1
0x14001a7cd:        call REDDecoder::create_1ec40               // [9] call method to open up file
```

First the method will copy the parameters that were passed to it into the frame for the method. Afterwards at [10], the method will pass the filename that was read from the socket and included as a parameter to the constructor at [10]. This constructor is responsible for initializing an object known as the “clip container”. After initializing a number of properties within the “clip container”, at [11] the filename that was passed as a parameter will be used to call into the R3D SDK. This call will result in the R3D SDK opening up the filename, parsing it, and then constructing an object that will be used to fetch attributes of the parsed video container.

```
0x14001ec67:        mov [rsp+160h+lvd_frameNumber_124], r9d
0x14001ec6c:        mov [rsp+160h+lvd_hdrMode?_128], r8d
0x14001ec71:        mov r15, rdx
0x14001ec74:        mov rsi, rcx
0x14001ec77:        mov [rsp+160h+lp_redDecoder_108], rcx
0x14001ec7c:        mov [rbp+60h+lp_string_60], rdx
0x14001ec80:        mov r14, [rbp+60h+ap_string_58]
0x14001ec87:        mov [rbp+60h+lp_string_58], r14
0x14001ec8b:        mov rdi, [rbp+60h+ap_readerObject_60]
0x14001ec92:        mov [rsp+160h+lp_readerObject_66], rdi
0x14001ec97:        mov r12, [rbp+60h+ap_string_68]
0x14001ec9e:        mov rax, [rbp+60h+ap_metadataObject_70]
0x14001eca5:        mov [rsp+160h+lp_metadataObject_68], rax
...
0x14001ecaa:        mov rcx, rdx                                // filename
0x14001ecad:        call pClipContainer::constructor_1f350      // [10] \...\ pass filename to constructor
0x14001ecb2:        mov rbx, rax
0x14001ecb5:        mov [rsp+160h+lp_clipContainer_100], rax
\...\
0x14001f4cd:        mov ecx, size clipContainerList
0x14001f4d2:        call operator new(unsigned __int64)
0x14001f4d7:        mov [rsp+1C0h+lp_clipContainerList?_180], rax
0x14001f4dc:        test rax, rax
0x14001f4df:        jz short loc_14001F4FB
...
0x14001f4e1:        mov rdx, r14
0x14001f4e4:        cmp [r14+stringArray.v_size_18], 10h
0x14001f4e9:        jb short redConstructClipContainer?_1f4ee
0x14001f4eb:        mov rdx, [r14+stringArray.p_contents_0]     // filename
0x14001f4ee: redConstructClipContainer?_1f4ee:
0x14001f4ee:        mov rcx, rax                                // this
0x14001f4f1:        call clipContainerList::newRedObject_21280  // [11] construct an object using the R3D SDK
0x14001f4f6:        mov rsi, rax
0x14001f4f9:        jmp short redGetClipContainerProperty?_1f4fd
```

The object that is constructed by the R3D SDK is described by the following code. First at [12], the object will be allocated and stored into a variable belonging to the function frame. At [13] its constructor will be called. Inside this constructor, another object will be allocated and constructed at [14]. After the object has been initialized, the filename that was passed as a parameter will be used with the method at [15] to open up the provided video container.

```
0x18003e712: loc_18003E712:
0x18003e712:        mov ecx, size R3D_AAC_object
0x18003e717:        call operator new(unsigned __int64)     // [12] allocate object
0x18003e71c:        mov [rsp+38h+p_8], rax
0x18003e721: loc_18003E721:
0x18003e721:        test rax, rax
0x18003e724:        jz short leave_3e732
0x18003e726:        mov rdx, rbx                            // filename
0x18003e729:        mov rcx, rax                            // this
0x18003e72c:        call R3D_AAC_object::method_51970       // [13] \ construct the object
\
0x18005198a:        mov ecx, size object_51a10
0x18005198f:        call operator new(unsigned __int64)     // [14] allocate object
0x180051994:        mov [rsp+38h+p_object_0], rax
0x180051999: loc_180051999:
0x180051999:        test rax, rax
0x18005199c:        jz short loc_1800519A7
0x18005199e:        mov rcx, rax                            // this
0x1800519a1:        call object_51a10::constructor_51a10    // [14] construct the object
0x1800519a6:        nop
0x1800519a7: loc_1800519A7:
0x1800519a7:        mov [rbx+R3D_AAC_object.p_object_0], rax
0x1800519aa:        mov rdx, rdi                            // filename
0x1800519ad:        mov rcx, rbx                            // this
0x1800519b0:        call object_51a10::method_56350         // [15] open up the provided filename using method
0x1800519b5:        mov rax, rbx
```

Eventually after constructing a few more objects, a method will be called which will execute the following code. This code will construct an object within the method's frame at [16], which will contain the result of parsing the filename for the video container that was passed to it. After constructing the object, at [17] the filename and the object that was constructed will be passed to a virtual method at 0x180039f80 in order to process the different parts of the file.

```
0x180033512:        lea rcx, [rbp+870h+lv_object_898]           // this
0x180033516:        call object_359e0::constructor_359e0        // [16] construct object in frame
...
0x180033544: parse_fileHeaderWithStages_33544:
0x180033544:        mov [rbp+870h+lp_object_8a0], rcx           // this
0x180033548:        mov rax, [rcx+object_35d50.p_vtable_0]
0x18003354b:        lea r8, [rbp+870h+lv_object_898]            // result object
0x18003354f:        mov rdx, r13                                // filename
0x180033552:        call qword ptr [rax+8]                      // [17] \ begin parsing of video container using method at 0x180039f80
0x180033555:        mov ebx, eax
0x180033557:        test eax, eax
0x180033559:        jz short collectIntoVector_33576
```

Inside the method, at [18] the library will open up the file using the filename that was passed as the method's parameter. After the library opens the file and constructs a number of objects, each of these objects will be passed to the constructor at [19]. This constructor will copy the references that were passed as its parameters directly into the object that is being constructed. Afterwards, the current method will begin to parse the file that was previously opened using the method call at [20].

```
0x180039f80: object_35d50::method_39f80:
0x180039f80:        mov rax, rsp
0x180039f83:        push rsi
0x180039f84:        push rdi
0x180039f85:        push r12
0x180039f87:        push r14
0x180039f89:        push r15
0x180039f8b:        sub rsp, 60h
0x180039f8f:        mov qword ptr [rax-38h], 0FFFFFFFFFFFFFFFEh
0x180039f97:        mov [rax+8], rbx
0x180039f9b:        mov [rax+18h], rbp
...
0x180039fbb:        mov rax, [rdi+object_35d50.v_fileObject_8.p_vtable_0]
0x180039fbf:        mov rdx, rbx                                    // filename
0x180039fc2:        lea rcx, [rdi+object_35d50.v_fileObject_8]      // this
0x180039fc6:        call qword ptr [rax+20h]                        // [18] open up filename using virtual method at 0x18002e4c0
0x180039fc9:        test al, al
...
0x18003a0ac:        mov ecx, size object_21d50
0x18003a0b1:        call operator new(unsigned __int64)
0x18003a0b6:        mov [rsp+88h+arg_8], rax
0x18003a0be: loc_18003A0BE:
0x18003a0be:        test rax, rax
0x18003a0c1:        jz short perform_parsingCodeLoop_3a0fb
...
0x18003a0c3:        mov [rsp+40h], bl                               // bool
0x18003a0c7:        mov [rsp+38h], r14                              // array of objects
0x18003a0cc:        mov rcx, [rdi+object_35d50.p_object_68]
0x18003a0d0:        mov [rsp+30h], rcx                              // object
0x18003a0d5:        mov rcx, [rdi+object_35d50.p_object_60]
0x18003a0d9:        mov [rsp+28h], rcx                              // object
0x18003a0de:        mov rcx, [rdi+object_35d50.p_object_58]
0x18003a0e2:        mov [rsp+20h], rcx                              // object
0x18003a0e7:        mov r9, [rdi+object_35d50.p_tinyObjectWrappee_50]   // object
0x18003a0eb:        xor r8d, r8d                                    // int
0x18003a0ee:        lea rdx, [rdi+object_35d50.v_fileObject_8]      // file object
0x18003a0f2:        mov rcx, rax                                    // this
0x18003a0f5:        call object_21d50::constructor_21d50            // [19] construct object containing references to each object in
parameters
...
0x18003a0fb: perform_parsingCodeLoop_3a0fb:
0x18003a0fb:        mov [rdi+object_35d50.p_object_f0], rax
0x18003a102:        mov rcx, rax                                    // this
0x18003a105:        call object_21d50::parse_cases_22df0            // [20] begin parsing video container
0x18003a10a:        test al, al
0x18003a10c:        jnz short parse_successful_3a121
```

This method call will execute the following code, which will parse the majority of the video container that was opened. This method contains a loop at [21] that will execute the methods at [22], depending on which stage needs to be parsed. After parsing each individual stage using the corresponding method, the value returned from each method will then be used to determine which stage to resume parsing.

```
0x180022df0: object_21d50::parse_cases_22df0:
0x180022df0:        push rbx
0x180022df2:        sub rsp, 20h
0x180022df6:        mov rbx, rcx
0x180022df9:        xor eax, eax
...
0x180022e00: loop_22e00:
0x180022e00:        test eax, eax                           // [21] determine which case to use
0x180022e02:        jz short case(0)_22e3c
0x180022e04:        dec eax
0x180022e06:        jz short case(1)_22e32
0x180022e08:        dec eax
0x180022e0a:        jz short case(2)_22e28
0x180022e0c:        dec eax
0x180022e0e:        jz short case(3)_22e1e
0x180022e10:        dec eax
0x180022e12:        jnz short return(0)_22e51
0x180022e14: case(4)_22e14:
0x180022e14:        mov rcx, rbx                            // this
0x180022e17:        call object_21d50::method_case(4)_21f70     // [22] parse stage 4
0x180022e1c:        jmp short continue_22e44
0x180022e1e: case(3)_22e1e:
0x180022e1e:        mov rcx, rbx                            // this
0x180022e21:        call object_21d50::method_case(3)_224b0     // [22] parse stage 3
0x180022e26:        jmp short continue_22e44
0x180022e28: case(2)_22e28:
0x180022e28:        mov rcx, rbx                            // this
0x180022e2b:        call object_21d50::method_case(2)_21e23     // [22] parse stage 2
0x180022e30:        jmp short continue_22e44
0x180022e32: case(1)_22e32:
0x180022e32:        mov rcx, rbx                            // this
0x180022e35:        call object_21d50::method_case(1)_22830     // [22] parse stage 1
0x180022e3a:        jmp short continue_22e44
0x180022e3c: case(0)_22e3c:
0x180022e3c:        mov rcx, rbx                            // this
0x180022e3f:        call object_21d50::method_case(0)_222c0     // [22] parse stage 0
0x180022e44: continue_22e44:
0x180022e44:        cmp eax, 5
0x180022e47:        jnz short loop_22e00
```

When first opening the file, the stage at [23] will be used in order to read the version information from the header and in order to determine how to parse the rest of the video container. First the header of the file will be read using the method at [24]. Inside the implementation of this method at [25], the header information of the file containing a type and length will be read using the implementation of the virtual method at 0x180028c00. After reading the header type and length, an object will be constructed at [26]. After constructing the object, the length that was previously read from the header will be used to cache the contents from the file into the object that was constructed. This is done using the virtual method at [27].

```
0x180022e3c: case(0)_22e3c:
0x180022e3c:        mov rcx, rbx                                // this
0x180022e3f:        call object_21d50::method_case(0)_222c0     // [23] \ parse stage 0
\
0x1800222cf:        mov rdi, rcx
0x1800222d2:        add rcx, object_21d50.v_headerObject?_38    // this
0x1800222d6:        call object_28560::read_header_23bc0        // [24] \ parse header from file
0x1800222db:        mov rbx, rax
\
0x180023bc0: object_28560::read_header_23bc0:
0x180023bc0:        push rdi
0x180023bc2:        sub rsp, 30h
0x180023bc6:        mov rax, [rcx+object_28560.p_vtable_0]
0x180023bc9:        mov rdi, rcx
0x180023bcc:        call qword ptr [rax+8]                      // [25] read header information using virtual method at 0x180028c00
0x180023bcf:        mov rax, [rdi+object_28560.p_vtable_0]
...
0x180023c31: found_header_23c31:
0x180023c31:        mov ecx, size object_24ca0
0x180023c36: loc_180023C36:
0x180023c36:        mov [rsp+38h+arg_0], rbx
0x180023c3b:        call operator new(unsigned __int64)         // [26] construct object
0x180023c40:        mov rbx, rax
0x180023c43:        test rax, rax
0x180023c46:        jz short loc_180023C54
...
0x180023c48:        lea rax, gvt_object(24ca0)_161780
0x180023c4f:        mov [rbx+object_24ca0.p_vtable_0], rax
0x180023c52:        jmp short loc_180023C56
0x180023c54: loc_180023C54:
0x180023c54:        xor ebx, ebx
0x180023c56: loc_180023C56:
0x180023c56:        mov rax, [rdi+object_28560.p_vtable_0]
0x180023c59:        mov rcx, rdi
0x180023c5c:        call qword ptr [rax+48h]                    // [27] cache length from header into object using virtual method at
0x180028960
```

After the contents of the header have been cached into the object, the following code will be executed in order to read the file information out of the record. At [28], the object responsible for containing fields from the header is first constructed. Afterwards at [29], values representing the version information of the video container are read from the header and stored directly into the object that was just recently constructed. This version information will be used in order to determine the contents of the rest of the header, and how the contents of the entire container is to be parsed.

```
0x180023c5f:        mov rcx, rbx                                            // this
0x180023c62:        mov rdx, rax                                            // file handle object
0x180023c65:        mov rdi, rax                                            // object containing fields read from header
0x180023c68:        call object_24ca0::constructor_24ca0                    // [28] constrct object containing fields read from header
...
0x180024ce6:        mov rcx, rsi                                            // this
0x180024ce9:        call object_2d5d0::read_byte_2d900                      // [29] read byte for Rversion
0x180024cee:        mov [rdi+object_24ca0.vb_RversionHigh_2c], al
...
0x180024cf1:        mov rcx, rsi                                            // this
0x180024cf4:        call object_2d5d0::read_byte_2d900                      // [29] read byte for Rversion
0x180024cf9:        mov [rdi+object_24ca0.vb_RversionLow_2d], al
...
0x180024cfc:        mov rcx, rsi                                            // this
0x180024cff:        call object_2d5d0::read_short_2d7b0                     // [29] read short for the "header type"
0x180024d04:        mov [rdi+object_24ca0.vw_chars_26], ax
0x180024d08:        xor ebp, ebp
```

With the provided proof of concept, the “R1” header type is used, which results in the following code being used to parse out the header. At [30], a number of fields are read from the header and stored directly into the object that was allocated. Specifically with regards to the presentlydescribed vulnerability, at [31] the dimensions of the container as well as the number of frames and their rate are read from the header before being stored into the object.

```
0x180024db2:        mov rcx, rsi                                            // this
0x180024db5:        call object_2d5d0::read_int_2d800                       // [30] read int
0x180024dba:        mov [rdi+object_24ca0.field_8], eax
...
0x180024dbd:        mov rcx, rsi                                            // this
0x180024dc0:        call object_2d5d0::read_int_2d800                       // [30] read int
0x180024dc5:        mov [rdi+object_24ca0.field_C], eax
...
0x180024dc8:        lea rdx, [rdi+object_24ca0.field_30]                    // buffer
0x180024dcc:        lea r8d, [rbp+10h]                                      // length
0x180024dd0:        mov rcx, rsi                                            // this
0x180024dd3:        call object_2d5d0::read_data_2d6e0                      // [30] read 0x10 bytes
...
0x180024dd8:        lea rdx, [rdi+object_24ca0.field_40]                    // buffer
0x180024ddc:        lea r8d, [rbp+10h]                                      // length
0x180024de0:        mov rcx, rsi                                            // this
0x180024de3:        call object_2d5d0::read_data_2d6e0                      // [30] read 0x10 bytes
...
0x180024de8:        mov rcx, rsi                                            // this
0x180024deb:        call object_2d5d0::read_int_2d800                       // [31] read int (width)
0x180024df0:        mov [rdi+object_24ca0.vd_imageWidth_10], eax
...
0x180024df3:        mov rcx, rsi                                            // this
0x180024df6:        call object_2d5d0::read_int_2d800                       // [31] read int (height)
0x180024dfb:        mov [rdi+object_24ca0.vd_imageHeight_14], eax
...
0x180024dfe:        mov rcx, rsi                                            // this
0x180024e01:        call object_2d5d0::read_int_2d800                       // [31] read int (frame count)
0x180024e06:        mov [rdi+object_24ca0.vd_framerateNumerator_18], eax
...
0x180024e09:        mov rcx, rsi                                            // this
0x180024e0c:        call object_2d5d0::read_short_2d7b0                     // [31] read int (framerate)
0x180024e11:        mov [rdi+object_24ca0.vw_framerateDenominator_24], ax
...
0x180024e15:        mov rcx, rsi                                            // this
0x180024e18:        call object_2d5d0::read_byte_2d900                      // [30] read byte
0x180024e1d:        mov [rdi+object_24ca0.vb_2e], al
...
0x180024e20:        lea rdx, [rdi+object_24ca0.vb_originalFilename(ff)_50]      // buffer
0x180024e24:        mov r8d, 0FFh                                           // length
0x180024e2a:        mov rcx, rsi                                            // this
0x180024e2d:        call object_2d5d0::read_data_2d6e0                      // [30] read 0xff byte string (original filename)
...
0x180024e32:        mov [rdi+object_24ca0.field_14F], bpl
0x180024e39:        mov rcx, rsi                                            // this
0x180024e3c:        call object_2d5d0::read_short_2d7b0                     // [30] read short (sector size)
0x180024e41:        mov [rdi+object_24ca0.vb_sectorShift?_2f], bpl
```

After the reading of the header is complete and the method returns, the following code will be executed. This will execute a virtual method at [32] using the object containing the header information as one of its parameters. This virtual method will swap its parameters so that the object containing the fields that were read from the header will instead be used as a parameter to the method at 0x180025ae0.

```
0x1800222f2:        mov rax, [rbx+object_24ca0.p_vtable_0]
0x1800222f5:        lea rdx, [rdi+object_21d50.v_tinyObject_98]              // destination object
0x1800222fc:        mov rcx, rbx                                            // this (fields from header)
0x1800222ff:        call qword ptr [rax+8]                                      // [32] call virtual method at 0x180023700
\
0x180023700: sub_180023700:
0x180023700:        mov rax, [rdx+object_21d50::tinyObject_98.p_vtable_0]
0x180023703:        mov r8, rdx                                             // destination object
0x180023706:        mov rdx, rcx                                            // object containing fields from header
0x180023709:        mov rcx, r8                                             // this
0x18002370c:        jmp qword ptr [rax+8]                                   // [32] branch to 0x180025ae0
```

Inside the virtual method at 0x180025ae0, each of the fields that were read from the header will be copied directly into the new object. At [33], the version information that was read from the header will be copied directly into another object that will be later saved into an STL container. Other fields that were parsed are copied into the same object at [34], whereas the dimensions of the video container are copied at [35] prior to the method returning back to the caller.

```
0x180025b03:        mov rax, [rcx+object_21d50::tinyObject_98.p_object_8]
0x180025b07:        mov [rax+object_14830.vw_RversionFull_1b4], r8w                                          // [33] copy version
information
...
0x180025b0f:        mov r8, [rcx+object_21d50::tinyObject_98.p_object_8]
0x180025b13:        movzx eax, [rdx+object_24ca0.vw_chars_26]
0x180025b17:        mov [r8+object_14830.vw_chars_82], ax                                                   // [33] copy "Rx" identifier
...
0x180025b1f:        mov rdx, [rcx+object_21d50::tinyObject_98.p_object_8]
0x180025b23:        mov eax, [r9+object_24ca0.vd_subrecord(e)_1c]
0x180025b27:        mov [rdx+object_14830.vd_subrecord(e)_1c], eax                                          // [34] copy field
...
0x180025b2d:        mov rcx, [rcx+object_21d50::tinyObject_98.p_object_8]
0x180025b31:        mov eax, [r9+object_24ca0.field_8]
0x180025b35:        mov [rcx+object_14830.v_object_2e0.field_3c], eax                                       // [34] copy field
...
0x180025b3b:        mov rax, [rbx+object_21d50::tinyObject_98.p_object_8]
0x180025b3f:        mov edx, 2
0x180025b44:        movups xmm0, xmmword ptr [r9+object_24ca0.field_30]
0x180025b49:        movups xmmword ptr [rax+object_14830.v_object_2e0.vx_someObject?_40.v_recordType_0], xmm0   // [34] copy field
...
0x180025b50:        mov eax, [r9+object_24ca0.field_C]
0x180025b54:        mov rcx, [rbx+object_21d50::tinyObject_98.p_object_8]
0x180025b58:        mov [rcx+object_14830.field_1b0], eax                                                   // [34] copy field
...
0x180025b5e:        movzx eax, [r9+object_24ca0.vb_2e]
0x180025b63:        mov rcx, [rbx+object_21d50::tinyObject_98.p_object_8]
0x180025b67:        mov [rcx+object_14830.vb_recordNeedsSummationOrSomething_1b6], al                       // [34] copy field
...
0x180025b6d:        mov eax, [r9+object_24ca0.vd_imageWidth_10]
0x180025b71:        mov rcx, [rbx+object_21d50::tinyObject_98.p_object_8]
0x180025b75:        mov [rcx+object_14830.v_object_2e0.vd_imageWidth?_50], eax                              // [35] copy width
...
0x180025b7b:        mov eax, [r9+object_24ca0.vd_imageHeight_14]
0x180025b7f:        mov rcx, [rbx+object_21d50::tinyObject_98.p_object_8]
0x180025b83:        mov [rcx+object_14830.v_object_2e0.vd_imageHeight?_54], eax                             // [35] copy height
...
0x180025b89:        mov eax, [r9+object_24ca0.vd_framerateNumerator_18]
0x180025b8d:        mov rcx, [rbx+object_21d50::tinyObject_98.p_object_8]
0x180025b91:        mov [rcx+object_14830.v_object_2e0.vd_framerateNumerator?_58], eax                      // [35] copy frame count
...
0x180025b97:        movzx eax, [r9+object_24ca0.vw_framerateDenominator_24]
0x180025b9c:        mov rcx, [rbx+object_21d50::tinyObject_98.p_object_8]
0x180025ba0:        mov [rcx+object_14830.v_object_2e0.vw_framerateDenominator?_5c], ax                     // [35] copy frames per
second
...
0x180025ba7:        mov eax, [r9+object_24ca0.vS_frameRate_28]
0x180025bab:        mov rcx, [rbx+object_21d50::tinyObject_98.p_object_8]
0x180025baf:        mov [rcx+object_14830.v_object_2e0.vS_frameRate?_60], eax                               // [34] copy field
...
0x180025bb5:        mov rcx, [rbx+object_21d50::tinyObject_98.p_object_8]
0x180025bb9:        movzx eax, [r9+object_24ca0.vb_sectorShift?_2f]
0x180025bbe:        mov [rcx+object_14830.vb_sectorShift?_1b7], al                                          // [34] copy field
```

After copying the fields from the header into another object, the method will return back to the caller and then enter a conditional (as demonstrated in the following code) that will execute different code depending on the version that was detected. This will assign other fields that were parsed during the reading of the header before returning a code to the caller. This code will then be used to determine the next component that will need to be parsed.

```
0x180022328: loc_180022328:
0x180022328:        mov eax, [rbx+object_24ca0.vd_loopIndex_154]
0x18002232e:        mov [rdi+object_21d50.vd_loopIndex_e0], eax
0x180022334:        movzx eax, [rbx+object_24ca0.vb_one_158]
0x18002233b:        mov [rdi+object_21d50.vb_one_db], al
0x180022341:        cmp [rbx+object_24ca0.vb_RversionHigh_2c], 5
0x180022345:        setnb al
0x180022348:        mov [rdi+object_21d50.vb_RversionAboveOrEqualTo5_da], al
0x18002234e:        test al, al
0x180022350:        mov rax, [rbx+object_24ca0.v_leftoverSize_1e0]
0x180022357:        jz short RversionFull(05xx)_2237b
\
0x180022e14: case(4)_22e14:
0x180022e14:        mov rcx, rbx                                // this
0x180022e17:        call object_21d50::method_case(4)_21f70
0x180022e1c:        jmp short continue_22e44
0x180022e1e: case(3)_22e1e:
0x180022e1e:        mov rcx, rbx                                // this
0x180022e21:        call object_21d50::method_case(3)_224b0
0x180022e26:        jmp short continue_22e44
0x180022e28: case(2)_22e28:
0x180022e28:        mov rcx, rbx                                // this
0x180022e2b:        call object_21d50::method_case(2)_21e20
0x180022e30:        jmp short continue_22e44
0x180022e32: case(1)_22e32:
0x180022e32:        mov rcx, rbx                                // this
0x180022e35:        call object_21d50::method_case(1)_22830
0x180022e3a:        jmp short continue_22e44
```

After successfully parsing everything and returning to the caller, at [36] the number of successfully parsed frames and records are written to the object that owns the calling method. Later in the function, the version information is also copied at [36], followed by the dimensions of the video container as shown at [37].

```
0x18003a121: parse_successful_3a121:
0x18003a121:        mov rcx, [rdi+object_35d50.p_tinyObjectWrappee_50]
0x18003a125:        mov eax, [rcx+object_14830.v_object_2e0.vd_numberofRecords(REDV)_34]
0x18003a12b:        mov [rdi+object_35d50.v_numberOfRecords(REDV)_f8], rax                  // [36] write number of records
...
0x18003a132:        mov eax, [rcx+object_14830.v_object_2e0.vd_numberOfTotalRecords_38]
0x18003a138:        mov [rdi+object_35d50.v_totalNumberOfContentRecords?_100], rax          // [36] write total number of records
...
0x18003a13f:        movzx eax, [rcx+object_14830.v_object_2e0.vb_count_488]
0x18003a146:        mov [rdi+object_35d50.v_count_218], rax                                 // [36] copy field
...
0x18003a297:        mov rax, [rdi+object_35d50.p_tinyObjectWrappee_50]
0x18003a29b:        movzx ecx, [rax+object_14830.vw_chars_82]
0x18003a2a2:        mov [rsi+object_359e0.v_chars_10], rcx                                  // [36] copy "Rx" version
...
0x18003a2a6:        mov rax, [rdi+object_35d50.p_tinyObjectWrappee_50]
0x18003a2aa:        movzx ecx, [rax+object_14830.vb_recordNeedsSummationOrSomething_1b6]
0x18003a2b1:        mov [rsi+object_359e0.v_recordNeedsSummationOrSomething_48], rcx        // [36] copy field
...
0x18003a2b5:        mov rax, [rdi+object_35d50.p_tinyObjectWrappee_50]
0x18003a2b9:        mov ecx, [rax+object_14830.v_object_2e0.vd_framerateNumerator?_58]
0x18003a2bf:        mov [rsi+object_359e0.field_30], rcx                                    // [37] copy frame count
...
0x18003a2c3:        mov rax, [rdi+object_35d50.p_tinyObjectWrappee_50]
0x18003a2c7:        movzx ecx, [rax+object_14830.v_object_2e0.vw_framerateDenominator?_5c]
0x18003a2ce:        mov [rsi+object_359e0.field_38], rcx                                    // [37] copy frames per second
...
0x18003a2d2:        mov rax, [rdi+object_35d50.p_tinyObjectWrappee_50]
0x18003a2d6:        mov ecx, [rax+object_14830.v_object_2e0.vd_imageWidth?_50]
0x18003a2dc:        mov [rsi+object_359e0.field_20], rcx                                    // [37] copy width
...
0x18003a2e0:        mov rax, [rdi+object_35d50.p_tinyObjectWrappee_50]
0x18003a2e4:        mov ecx, [rax+object_14830.v_object_2e0.vd_imageHeight?_54]
0x18003a2ea:        mov [rsi+object_359e0.field_28], rcx                                    // [37] copy height
```

After copying the necessary fields and returning to the caller, the calling method will use this object to iterate through all of the frame records that were encoded within the container and copy them into a vector of intermediary objects. At [38], the objects will get assigned into an object containing a vector using the standard template library. This is then followed by the attributes that were parsed out of the video container being copied into the same object at [39].

```
0x180033576: collectIntoVector_33576:
0x180033576:        mov r14, r12
0x180033579:        lea rdi, [rsi+object_5b600.v_someVector?_3e8]
0x180033580:        lea rax, [rbp+870h+lp_object_8a0]
0x180033584:        mov [rsp+20h], rax                          // source object
0x180033589:        mov r9d, 1                                  // index
0x18003358f:        mov r8, [rdi+std::vector.p_first_0]         // std::vector
0x180033592:        lea rdx, [rbp+870h+p_object_8]              // object parsed by stages
0x180033599:        mov rcx, rdi                                // this
0x18003359c:        call sub_180035120                          // [38] copy elements into object containing std::vector
...
0x1800335a1:        lea rdx, [rbp+870h+lv_object_898]           // source object
0x1800335a5:        mov rcx, rsi                                // this
0x1800335a8:        call sub_180034DA0                          // [39] copy fields into object containing std::vector
```

Prior to exiting the method, the following code is executed to copy the attributes from the object containing a vector of all of the parsed records into its 3rd parameter, which contains the direct attributes that were parsed out of the video container. This is shown at [40] where all of the relevant fields are written into the object found in the method's parameters.

```
0x180033a43: loc_180033A43:
0x180033a43:        mov [rsi+object_5b600.p_object_408], rax
0x180033a4a:        mov r8, [rdi+(object_5b600.v_someVector?_3e8.p_first_0-3E8h)]
0x180033a4d:        mov [rbp+870h+p_object_8], r8
...
0x180033a54:        movzx eax, word ptr [r8+18h]
0x180033a59:        mov [r15+object_14830.vw_chars_82], ax                              // [40] store the "Rx" version
...
0x180033a61:        movzx eax, byte ptr [r8+50h]
0x180033a66:        mov [r15+object_14830.vb_recordNeedsSummationOrSomething_1b6], al   // [40] store a field
...
0x180033a6d:        mov eax, [r8+38h]
0x180033a71:        mov [r15+object_14830.v_object_2e0.vd_framerateNumerator?_58], eax  // [40] store the frame count
...
0x180033a78:        movzx eax, word ptr [r8+40h]
0x180033a7d:        mov [r15+object_14830.v_object_2e0.vw_framerateDenominator?_5c], ax // [40] store the fps
0x180033a85:        mov rcx, [r8+40h]
...
0x180033acf: loc_180033ACF:
0x180033acf:        movss [r15+object_14830.v_object_2e0.vS_frameRate?_60], xmm0        // [40] store the frame rate
0x180033ad8:        mov eax, [r8+28h]
0x180033adc:        mov [r15+object_14830.v_object_2e0.vd_imageWidth?_50], eax          // [40] store with
0x180033ae3:        mov eax, [r8+30h]
0x180033ae7:        mov [r15+object_14830.v_object_2e0.vd_imageHeight?_54], eax         // [40] store height
0x180033aee:        cmp [rsi+object_5b600.p_string_3e0], 0
0x180033af6:        jz short loc_180033B30
0x180033af8:        mov rcx, [r8+70h]
...
0x180033bb1:        mov eax, dword ptr [rsi+object_5b600.v_numberOfRecords(REDV)_2c8]
0x180033bb7:        mov [r15+object_14830.v_object_2e0.vd_numberofRecords(REDV)_34], eax // [40] store number of records
0x180033bbe:        mov eax, dword ptr [rsi+object_5b600.v_numberOfTotalRecords_2d0]
0x180033bc4:        mov [r15+object_14830.v_object_2e0.vd_numberOfTotalRecords_38], eax  // [40] store total number of records
0x180033bcb:        movzx eax, byte ptr [r8+16Ch]
```

The same fields are also added into a hashtable that can be used by a consumer of the library to query the attributes directly by name. At [41], all of the integer attributes that were parsed are stored using a related key, followed by the "framerate" as a float at [42] and the "original_filename" as a string at [43].

```
0x180033cf6:        mov r8d, [r15+object_14830.v_object_2e0.vd_imageWidth?_50]             // value
0x180033cfd:        lea rdx, str.imagewidth                                                // key
0x180033d04:        mov rbx, [rbp+870h+p_maybeLinkedList_136]
0x180033d0b:        mov rcx, rbx                                                           // this
0x180033d0e:        call sub_180015DA0                                                     // [41] store integer as "image_width"
...
0x180033d13:        mov r8d, [r15+object_14830.v_object_2e0.vd_imageHeight?_54]            // value
0x180033d1a:        lea rdx, str.imageheight                                               // key
0x180033d21:        mov rcx, rbx                                                           // this
0x180033d24:        call sub_180015DA0                                                     // [41] store integer as "image_height"
...
0x180033d29:        movss xmm2, [r15+object_14830.v_object_2e0.vS_frameRate?_60]           // value
0x180033d32:        lea rdx, str.framerate                                                 // key
0x180033d39:        mov rcx, rbx                                                           // this
0x180033d3c:        call sub_180015AE0                                                     // [42] store float as "framerate"
...
0x180033d41:        mov r8d, [r15+object_14830.v_object_2e0.vd_framerateNumerator?_58]     // value
0x180033d48:        lea rdx, str.frameratenumerator                                        // key
0x180033d4f:        mov rcx, rbx                                                           // this
0x180033d52:        call sub_180015DA0                                                     // [41] store integer as "framerate_numerator"
...
0x180033d57:        movzx r8d, [r15+object_14830.v_object_2e0.vw_framerateDenominator?_5c] // value
0x180033d5f:        lea rdx, str.frameratedenominator                                      // key
0x180033d66:        mov rcx, rbx                                                           // this
0x180033d69:        call sub_180015DA0                                                     // [41] store integer as "framerate_denominator"
...
0x180033d6e:        lea r13, [r15+object_14830.v_object_2e0.vb_originalFilename(100)?_348]
0x180033d75:        mov r8, r13                                                            // value
0x180033d78:        lea rdx, str.originalfilename                                          // key
0x180033d7f:        mov rcx, rbx                                                           // this
0x180033d82:        call sub_180016040                                                     // [43] store string as "original_filename"
```

After storing each of the necessary attributes, the following code will be executed in order to check the version. With the provided proof-of-concept, the version is set to “R1”, which will result in the branch at [44] being taken. Within this branch, a method will be called at [45] in order to construct an object that will be used to read information about the records that were parsed within the video container. At [46], the method will allocate and construct an object, which will then be used to parse an index of all of the records within the video container. Each iteration of the records in the container will result in storing a record type at [47]. After indexing each record, this object is returned to the caller and stored in the `%rbx` register.

```
0x180033da6: check_version(R1)_33da6:
0x180033da6:        mov eax, 'R1'
0x180033dab:        cmp [r15+object_14830.v_object_1d0.vb_one_f1], 0
0x180033db3:        jz short version(R1)_33df8
0x180033db5:        cmp [r15+object_14830.vw_chars_82], ax
0x180033dbd:        jz short version(R1)_33df8                          // [44] take branch for version
...
0x180033e1d: constructObjectUsedForUuid_33e1d:
0x180033e1d:        mov rcx, [rsi+object_5b600.p_object_400]            // this
0x180033e24:        call object_2d5d0::method_2d950
0x180033e29:        mov rcx, [rsi+object_5b600.p_object_400]            // this
0x180033e30:        call object_2d5d0::constructObject_23e30            // [45] \ construct object
0x180033e35:        mov rbx, rax                                        // store result object in %rbx
0x180033e38:        test rax, rax
0x180033e3b:        jz short return(3)_33de6
\
0x180023e46: loc_180023E46:
0x180023e46:        mov ecx, size object_25650
0x180023e4b: loc_180023E4B:
0x180023e4b:        mov [rsp+28h+arg_0], rbx
0x180023e50:        call operator new(unsigned __int64)                 // [46] allocate object
0x180023e55:        mov rbx, rax
0x180023e58:        test rax, rax
0x180023e5b:        jz short loc_180023E9A
...
0x180023e9d: construct_23e9d:
0x180023e9d:        mov rdx, rdi                                        // object
0x180023ea0:        mov rcx, rbx                                        // this
0x180023ea3:        call object_25650::constructor_25650                // [46] \...\ construct object that will index each of the records
within the container
0x180023ea8:        test al, al
0x180023eaa:        jz short return(0)_23eba
\...\
0x1800256cd: loop_256cd:
0x1800256cd:        mov eax, ebp
0x1800256cf:        shr eax, 18h
0x1800256d2:        mov [rdi+object_25650.vw_R1_12], ax
0x1800256d6:        mov eax, ebx
0x1800256d8:        shr ebx, 10h
0x1800256db:        shr eax, 18h
0x1800256de:        mov [rdi+object_25650.field_11], bl
0x1800256e1:        mov [rdi+object_25650.vb_case_10], al               // [47] store each case from beginning of indexed record
0x1800256e4:        sub al, 5
0x1800256e6:        cmp al, 1
0x1800256e8:        ja return(0)_258d4
```

Once the records from the video container have been indexed and the object is stored in the `%rbx` register, a virtual method will be called to parse the actual contents of the record. This is shown in the following code at [48]. At [49], the implementation of the virtual method will simply branch to the real method at 0x180025f60 in order to parse the contents of each record that were indexed when the object was constructed.

```
0x180033e84:        mov rax, [rbx+object_25650.p_vtable_0]
0x180033e87:        lea rdx, [rbp+870h+lv_object_8d0]           // list of records
0x180033e8b:        mov rcx, rbx                                // this
0x180033e8e:        call qword ptr [rax+8]                      // [48] \ call virtual method implemented at 0x180108580
0x180033e91:        test al, al
0x180033e93:        jz return(3)_349a7
\
0x180108580: sub_180108580:
0x180108580:        mov rax, [rdx+object_23370.p_vtable_0]
0x180108583:        mov r8, rdx
0x180108586:        mov rdx, rcx
0x180108589:        mov rcx, r8
0x18010858c:        jmp qword ptr [rax+28h]                     // [49] call virtual method at 0x180025f60
```

Using the case that was read from the beginning of each record, parsed from the video container, the implementation of the virtual method at address 0x180025f60 that is shown in the following code will branch to the handler responsible for each record type. At [50], the record type is read and then checked. When processing a record of type 3, the branch at [51] will be taken. At [52], the handler for record type 3 will loop over all of the objects within the record's contents while looking for header types. When iterating over each record, the method will first call the method at [53]. This method will try and parse any records of the type "uuid". If this method fails, then at [54] the loop will try to parse a record of type "jp2c" before breaking from the loop.

```
0x180025f81:        movzx eax, [rbx+object_25650.vb_case_10]               // [50] read record type from index
0x180025f85:        cmp al, 3
0x180025f87:        jnz short case(4)_25f9e
0x180025f89: case(3)_25f89:
0x180025f89:        mov rdx, rbx                                           // record object
0x180025f8c:        mov rcx, rdi                                           // this
0x180025f8f:        mov rbx, [rsp+28h+arg_0]
0x180025f94:        add rsp, 20h
0x180025f98:        pop rdi
0x180025f99:        jmp object_23370::method_case(3)_25ff0                 // [51] branch to handler for record type
\
0x18002604a: loop_record(uuid,jp2c)_2604a:
0x18002604a:        lea rcx, [rsp+0E8h+lv_object_98.v_object_30]           // this
0x180026052:        call object_28250::field_30::readRecord?_28b30         // [52] read contents of current record in loop
0x180026057:        test al, al
0x180026059:        jz return(0)_2626b
...
0x18002605f:        mov rdx, rbp                                           // object
0x180026062:        lea rcx, [rsp+0E8h+lv_object_98.v_object_30]           // this
0x18002606a:        call object_28250::field_30::method_23a20              // [53] try and parse any uuid records
0x18002606f:        mov rbx, rax
0x180026072:        test rax, rax
0x180026075:        jz short continue_record(jp2c,a5)_2609e
0x180026077:        cmp [rdi+object_23370.p_object_10], 0
0x18002607c:        jz short continue_record(uuid,10)_2608f
...
0x18002607e:        mov r8, [rax+object_uuid.p_vtable_0]
0x180026081:        mov rdx, rdi
0x180026084:        mov rcx, rax
0x180026087:        call qword ptr [r8+8]                                      // [54] parse the jp2c record type using method at
0x1800236e0
0x18002608b:        test al, al
0x18002608d:        jnz short return(0)_260d9
...
0x18002608f: continue_record(uuid,10)_2608f:
0x18002608f:        mov rax, [rbx]                                         // object virtual method table at 0x1801617b0
0x180026092:        mov edx, 1
0x180026097:        mov rcx, rbx
0x18002609a:        call qword ptr [rax]                                   // destructor for uuid record type
0x18002609c:        jmp short loop_record(uuid,jp2c)_2604a
0x18002609e: continue_record(jp2c,a5)_2609e:
0x18002609e:        mov edx, 'jp2c'
0x1800260a3:        mov r8d, 0A5h
0x1800260a9:        lea rcx, [rsp+0E8h+lv_recordHeader_a8]                 // this
0x1800260ae:        call recordHeader::assign_28620
0x1800260b3:        movaps xmm0, xmmword ptr [rsp+0E8h+lv_recordHeader_a8.v_recordType_0]
0x1800260b8:        movdqa xmmword ptr [rsp+0E8h+lv_recordHeader_c8.v_recordType_0], xmm0
0x1800260be:        lea rdx, [rsp+0E8h+lv_recordHeader_c8]                 // record header result
0x1800260c3:        lea rcx, [rsp+0E8h+lv_object_98.v_object_30]           // this
0x1800260cb:        call object_28560::compare_recordLength_minimum_28e60  // [52] check record length to continue loop
0x1800260d0:        test al, al
0x1800260d2:        jnz short break_record(uuid,jp2c)_260eb
0x1800260d4:        jmp loop_record(uuid,jp2c)_2604a
```

The method at address 0x180023a2a is implemented by the following code. This code will first read the header of the record from the file at [53] and check that its type is "uuid" and that its size is at least 0x1d bytes. If so, the loop will allocate and initialize an object at [54] in order to retain the contents of the "uuid" element. This element will then get stored in the `%rdi` register. Afterwards at [55], the loop will construct an object to read the 0x10 bytes into a buffer located on the stack.

```
0x180023a2a:        movups xmm0, xmmword ptr cs:gv_recordHeader(uuid,1d)_31e930.v_recordType_0
0x180023a31:        mov rax, [rcx+object_28250::field_30.p_vtable_0]
0x180023a34:        mov rbp, rdx
0x180023a37:        lea rdx, [rsp+38h+lv_recordHeader(uuid,1d)_18]
0x180023a3c:        mov rsi, rcx
0x180023a3f:        movaps xmmword ptr [rsp+38h+lv_recordHeader(uuid,1d)_18.v_recordType_0], xmm0
0x180023a44:        call qword ptr [rax+28h]                           // [53] check record length using virtual method at 0x180028e60
0x180023a47:        test al, al
0x180023a49:        jnz short loc_180023A58
...
0x180023a58: loc_180023A58:
0x180023a58:        mov ecx, size object_uuid
0x180023a5d:        mov [rsp+38h+arg_0], rbx
0x180023a62:        mov [rsp+38h+arg_8], rdi
0x180023a67:        call operator new(unsigned __int64)                // [54] allocate object for uuid
0x180023a6c:        xor ebx, ebx
0x180023a6e:        mov rdi, rax                                       // [54] store it in %rdi
0x180023a71:        test rax, rax
0x180023a74:        jz short loc_180023A82
0x180023a76:        lea rax, gvt_1617b0
0x180023a7d:        mov [rdi+object_uuid.p_vtable_0], rax              // [54] write virtual method table
0x180023a80:        jmp short loc_180023A85
0x180023a82: loc_180023A82:
0x180023a82:        mov rdi, rbx
0x180023a85: loc_180023A85:
0x180023a85:        mov rax, [rsi+object_28250::field_30.p_vtable_0]   // virtual method table at 0x180161a88
0x180023a88:        mov rcx, rsi
0x180023a8b:        call qword ptr [rax+48h]                           // [55] call method at 0x180028960 to construct object for reading
contents
...
0x180023a8e:        lea rdx, [rsp+38h+lv_recordHeader(uuid,1d)_18]     // buffer (reused)
0x180023a93:        mov r8d, 10h                                       // length
0x180023a99:        mov rcx, rax                                       // this
0x180023a9c:        mov rsi, rax                                       // [55] read 0x10 bytes from contents
0x180023a9f:        call object_2d5d0::read_data_2d6e0
```

After reading the contents of the “uuid” record, the following code will be executed to check the 0x10 bytes that were read from the file. First at [56], the expected 0x10 bytes is loaded into the `%rax` register representing the UUID, `{0A0DD7535-694A-DB11-BD870002A5D5C51B}`. This value is then compared against the bytes read from the file using the loop at [57], which reads each byte and compares it until all 0x10 bytes are read. If all 0x10 bytes from the file match, only then at [58] will the object that was previously allocated be written to the object that was constructed for the uuid. If the bytes do not match, then the loop will exit at [59] before writing the reference to the object that was allocated to the uuid object. The vulnerability described by this document is specifically due to this condition, where the reference is not written to the object, which leaves the property uninitialized. Upon breaking from the loop, the implementation will destroy any references to the uuid object. At [60], if the uuid object was allocated, its virtual destructor will be called. The implementation of this destructor will then attempt to destroy the property that should’ve been assigned an object. Due to the loop terminating before the property was written, the destruction of this property will dereference a pointer to an object that was uninitialized. At [61], the property will be fetched from the uuid object before having its virtual method table dereferenced in order to call its destructor. This dereference of the virtual method table can lead to code execution under the context of the library.

```
0x180023aa4:        lea rax, gv_guid_1614b0                           // [56] bytes
'\x35\x75\xdd\xa0\x4a\x69\x11\xdb\xbd\x87\x00\x02\xa5\xd5\xc5\x1b'
0x180023aab:        nop dword ptr [rax+rax+00h]
...
0x180023ab0: loop_check(uuid)_23ab0:
0x180023ab0:        movzx ecx, byte ptr [rbx+rax]                     // [57] read next byte
0x180023ab4:        cmp byte ptr [rsp+rbx+38h+lv_recordHeader(uuid,1d)_18.v_recordType_0], cl
0x180023ab8:        jnz short break_uuid_23ad0                        // [59] exit loop to destroy uuid object if bytes do not match
0x180023aba:        inc rbx
0x180023abd:        cmp rbx, 10h                                      // [57] continue to read 0x10 bytes
0x180023ac1:        jb short loop_check(uuid)_23ab0
...
0x180023ac3:        mov [rdi+object_uuid.p_recordObject_8], rsi       // [58] if bytes match, then write pointer of object that was
allocated to uuid object property
0x180023ac7:        mov [rdi+object_uuid.p_object_10], rbp
0x180023acb:        mov rax, rdi
0x180023ace:        jmp short leave_23ae4
...
0x180023ad0: break_uuid_23ad0:
0x180023ad0:        test rdi, rdi
0x180023ad3:        jz short return(0)_23ae2
0x180023ad5:        mov rax, [rdi+object_uuid.p_vtable_0]             // dereference virtual method table for uuid object
0x180023ad8:        mov edx, 1
0x180023add:        mov rcx, rdi
0x180023ae0:        call qword ptr [rax]                              // [60] call virtual destructor for uuid object at 0x180023590
\
0x1800235b7: loc_1800235B7:
0x1800235b7:        mov rcx, [rcx+object_uuid.p_recordObject_8]       // [61] dereference uninitialized property
0x1800235bb:        test rcx, rcx
0x1800235be:        jz short loc_1800235CB
0x1800235c0:        mov rax, [rcx]                                    // [61] dereference virtual method table
0x1800235c3:        mov edx, 1
0x1800235c8:        call qword ptr [rax]                              // [61] call dereferenced pointer
```

Crash Information

Upon running the provided-proof-of-concept against the port running the `DPDecoder` service, the debugger will immediately break while trying to dereference an uninitialized pointer.

```
0:002> g
ModLoad: 00007ff9`73540000 00007ff9`73552000   C:\Windows\SYSTEM32\kernel.appcore.dll
(137c.128): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
REDR3D_x64+0x235c0:
00007ff9`444335c0 488b01          mov     rax,qword ptr [rcx] ds:c0c0c0c0`c0c0c0c0=????????????????
```

The code that follows will attempt to execute the dereferenced pointer.

```
0:002> u .
REDR3D_x64+0x235c0:
00007ff9`444335c0 488b01          mov     rax,qword ptr [rcx]
00007ff9`444335c3 ba01000000      mov     edx,1
00007ff9`444335c8 ff10            call    qword ptr [rax]
00007ff9`444335ca 90              nop
00007ff9`444335cb 488d057eeb1300  lea     rax,[REDR3D_x64!RED_LIB_PATCH_VERSION+0x1213e0 (00007ff9`44572150)]
00007ff9`444335d2 488903          mov     qword ptr [rbx],rax
00007ff9`444335d5 40f6c701        test    dil,1
00007ff9`444335d9 7408            je      REDR3D_x64+0x235e3 (00007ff9`444335e3)
```

The backtrace corresponds to the call stack that is described in the advisory.

```
0:002> kv
 # Child-SP          RetAddr           : Args to Child                                                   : Call Site
00 00000001`004fe7e0 00007ff9`44433ae2 : 0000013a`5edf9fe0 00000000`00000000 00000000`000001f8 00000001`004fe908 : REDR3D_x64+0x235c0
01 00000001`004fe820 00007ff9`4443606f : 0000013a`5edf4f90 00000001`004fe9f0 0000013a`5edf4f90 00000000`00000000 : REDR3D_x64+0x23ae2
02 00000001`004fe860 00007ff9`44443e91 : 0000013a`5edf4f90 0000013a`5edf4f90 00000001`004fea50 0000013a`5eaaa650 : REDR3D_x64+0x2606f
03 00000001`004fe950 00007ff9`4446b155 : 0000013a`5ea9cfe0 0000013a`5ed39d30 0000013a`5eaaaa68 0000013a`5eaa6ff0 : REDR3D_x64+0x33e91
04 00000001`004ff2d0 00007ff9`44470511 : 0000013a`5eaa4ff0 0000013a`5ea9cfe0 0000013a`00000004 0000013a`5eaaa650 :
REDR3D_x64!RED_LIB_PATCH_VERSION+0x1a3e5
05 00000001`004ff370 00007ff9`4446639a : 00000000`00000000 00000001`004ff3c4 0000013a`5eaa4ff0 00007ff9`75b83ed0 :
REDR3D_x64!RED_LIB_PATCH_VERSION+0x1f7a1
06 00000001`004ff3b0 00007ff9`444619b5 : 0000013a`5eaa2ca0 0000013a`5eaa0ff0 0000013a`5d187fb0 00000001`004ffa38 :
REDR3D_x64!RED_LIB_PATCH_VERSION+0x1562a
07 00000001`004ff4e0 00007ff9`4444e731 : 0000013a`5eaa2ca0 0000013a`5ea9cfe0 00000000`00000000 00000000`00000008 :
REDR3D_x64!RED_LIB_PATCH_VERSION+0x10c45
08 00000001`004ff520 00007ff6`d055129d : 00000000`00000008 0000013a`5eaa0ff0 000063f1`b97c93d3 0000013a`5e8b2f70 : REDR3D_x64!R3D_AAC+0x31
09 00000001`004ff560 00007ff6`d054f4f6 : 00000004`00000000 0000013a`5e8b2f60 0000013a`5e898870 00007ff9`00000004 : DPDecoder+0x2129d
0a 00000001`004ff590 00007ff6`d054ecb2 : 00000004`00000048 0000013a`5ea90e70 0000013a`5e860f80 0000013a`5ea90e70 : DPDecoder+0x1f4f6
0b 00000001`004ff760 00007ff6`d054a7d2 : 0000013a`5e860f80 0000013a`5ea90e70 00000001`004ffbb0 00000000`00000006 : DPDecoder+0x1ecb2
0c 00000001`004ff8d0 00007ff6`d054ca04 : 00000000`00000001 00000000`00000006 00004544`4f434544 0000013a`5e860f80 : DPDecoder+0x1a7d2
0d 00000001`004ffb50 00007ff6`d054dbf9 : 0000013a`5e878fd0 0000013a`5e878fd0 0000013a`5e876ff0 00000001`004ffc78 : DPDecoder+0x1ca04
0e 00000001`004ffbe0 00007ff9`6967b63e : 0000013a`5e878fd0 00000000`00000000 0000013a`5e878fd0 0000013a`5e876ff0 : DPDecoder+0x1dbf9
0f 00000001`004ffc10 00007ff9`756a1bb2 : 0000013a`5e876fe0 00000000`00000000 00000000`00000000 00000000`00000000 :
pthreadVC2!pthread_rwlockattr_init+0xa152
10 00000001`004ffcc0 00007ff9`77577034 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 :
ucrtbase!thread_start<unsigned int (__cdecl*)(void *),1>+0x42
11 00000001`004ffcf0 00007ff9`77ee2651 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 :
KERNEL32!BaseThreadInitThunk+0x14
12 00000001`004ffd20 00000000`00000000 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 :
ntdll!RtlUserThreadStart+0x21
```

The base address of the R3D SDK helper library is located at 0x7ff944410000.

```
0:002> lm a redr3d_x64
Browse full module list
start             end                 module name
00007ff9`44410000 00007ff9`4475f000   REDR3D_x64 C (export symbols)       REDR3D-x64.dll

Unloaded modules:
00007ff9`44410000 00007ff9`4475f000   REDR3D-x64.dll
```

Exploit Proof of Concept

The proof-of-concept provided with this vulnerability is in two parts. The first part is specifically to generate the file that will be submitted to the `DPDecoder` process. There are multiple variations of this file format. However, as this file format is completely without documentation, only one variation (“R1” v0400) was implemented.

To generate the file, one may run the proof-of-concept as follows:

```
$ python poc.file.zip /path/to/save/file 1
```

Similarly, to examine the file that is generated, one may pass the `-i` parameter to Python and explore the format in the Python interpreter using the `self` variable.

```
$ python -i poc.file.zip /path/to/save/file 1
>>> print(self)
...
```

In the file that is generated, the version information that is found at offset 0x8 represents the particular variation that is triggered by this proof-of-concept. The frames that are decoded begin at offset 0x144 of the file. At offset +0x18 of each frame is a byte representing the case number, which should be 0x03. If this is true, then at offset 0x1c of each frame should be a list of blocks that should normally follow the JPEG2000 image specification, and will terminate when encountering the “jp2c” block. If any of the blocks prior to the “jp2c” block are using the type “uuid” and are followed by 0x10 bytes that do not match the string “\x35\x75\xdd\xa0\x4a\x69\x11\xdb\xbd\x87\x00\x02\xa5\xd5\xc5\x1b” exactly, then this vulnerability is being triggered.

Once the file has been generated, it must be placed on a file share that is accessible by the host.

To submit the file into the `DPDecoder` service, one may use the second part of the proof-of-concept in order to enqueue the generated into the service.

This can be done by running the proof-of-concept with the hostname and port number of the `DPDecoder` service, and the path to the share containing the file that was generated in part 1. If the port is unknown, this port may be fingerprinted by connecting to a target port and sending the string “\x00\x00\x00\x07VERSION”, at which point if an instance of the `DPDecoder` service is listening, the service will respond with the version information of the service.

```
$ python poc.client.zip hostname:portnumber \\client\sharename\path\to\file
```

If the user does not wish to enqueue the file via an SMB share, they may simply copy the generated file to a location that is accessible by the `DPDecoder` service. They may then enqueue the file by passing an absolute path to the client.

```
$ python poc.client.zip hostname:portnumber drive:\path\to\file
```

After the client has submitted the generated proof-of-concept to the `DPDecoder.exe` process as a job, an attached debugger should crash at the described location within this advisory.

Timeline

2021-12-09 - Vendor Disclosure

2021-12-20 - Public Release

CREDIT

Discovered by a member of Cisco Talos.
