Talos Vulnerability Report

TALOS-2021-1426

Blackmagic Design DaVinci Resolve R3D DPDecoder Service frame decoding heap-based buffer overflow vulnerability

DECEMBER 20, 2021

CVE NUMBER

CVE-2021-40417

Summary

When parsing a file that is submitted to the DPDecoder service as a job, the service will use the combination of decoding parameters that were submitted with the job along with fields that were parsed for the submitted video by the R3D SDK to calculate the size of a heap buffer. Due to an integer overflow with regards to this calculation, this can result in an undersized heap buffer being allocated. When this heap buffer is written to, a heap-based buffer overflow will occur. This 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-680 - Integer Overflow to Buffer Overflow

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, 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, allowing 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 and parsing it, 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, and then at [13] it's 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 opening up the file and constructing 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 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 has 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 presently described 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]. 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. This 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
```

Once it copyies the necessary fields and returns them 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 hash table 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. This will then be returned to the caller so that a consumer of the library may access the contents of the video container that was parsed.

```
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 that is returned
0x180033e35:        mov rbx, rax                                        // store result object in %rbx prior to returning
0x180033e38:        test rax, rax
0x180033e3b:        jz short return(3)_33de6
```

Once the library returns back to the decoder and the “clipContainer” object has been constructed, the decoder executes the function call at [44]. This will use the decode mode that was read from the socket in order to determine the pixel format and its size, which are passed as parameters. At [45], the decoder will call back into the library in order to fetch the width and height that were parsed directly out of the container. These values are then divided by the pixel size that was returned from the decode mode.

```
0x14001ecec:        lea rcx, [rbp+60h+lv_object_d0]
0x14001ecf0:        call object_215b0::constructor_215b0
...
0x14001ecf5:        mov [rbp+60h+lv_object_d0.p_readerObject_30], rdi
0x14001ecf9:        mov dword ptr [rsp+160h+lv_resultModeDivisor_64], 1
0x14001ed01:        lea r9, [rsp+160h+lv_resultModeDivisor_64]          // pixel size
0x14001ed06:        lea r8, [rbp+60h+lv_object_d0.v_resultFourCC_c0]     // pixel format
0x14001ed0a:        mov edx, dword ptr [rbp+60h+av_decodeMode_30]       // decode mode
0x14001ed10:        mov rcx, rsi                                        // this
0x14001ed13:        call REDDecoder::getPixelFormatAndDivisor_1fad0     // [44] determine pixel format and size in order to write to
parameters
0x14001ed18:        test al, al
0x14001ed1a:        jnz short fetched_pixelformat_1ed90
...
0x14001ed90: fetched_pixelformat_1ed90:
0x14001ed90:        movsxd rdi, dword ptr [rsp+160h+lv_resultModeDivisor_64]
0x14001ed95:        mov rcx, r13                                        // this (clipContainerList)
0x14001ed98:        call clipContainerList::property(imageWidth)_21940  // [45] image width
0x14001ed9d:        xor edx, edx
...
0x14001ed9f:        div rdi
0x14001eda2:        mov rsi, rax                                        // [45] store width divided by mode
0x14001eda5:        mov rcx, r13                                        // this (clipContainerList)
0x14001eda8:        call clipContainerList::property(imageHeight)_21720 // [45] image height
0x14001edad:        xor edx, edx
0x14001edaf:        div rdi
0x14001edb2:        mov rcx, rax                                        // [45] store height divided by mode
0x14001edb5:        mov [rsp+160h+lv_resultModeDivisor_64], rax
```

After calculating the dimensions, the decoder will then use the bits-per-pixel that was sent over the socket and passed as a decoding parameter to this method in order to determine the pitch of the decoded video container. At [46], the bits-per-pixel and a boolean describing whether the video to decode contains a palette will be read from the method’s parameter and then used to determine how to calculate the pitch. At [47], the bits-per-pixel will be checked and used to multiply the width by the amount of data that is associated with each pixel.

```
0x14001edba:        mov r13d, dword ptr [rbp+60h+avd_bpp_38]                // [46] read bits per pixel from parameter
0x14001edc1:        cmp byte ptr [rbp+60h+avb_hasPalette_50], 0             // [46] check parameter if a palette was included
0x14001edc8:        jz short check_bpp_1ee01
0x14001edca:        cmp r13d, 10h
0x14001edce:        jz short depth(20)_1edef
...
0x14001edef: depth(20)_1edef:
0x14001edef:        mov [rbp+60h+lv_object_d0.vd_pixelType_bc], 'RBHA'      // [47] if bits per pixel is 32
0x14001edf6:        lea eax, [rsi+rsi*2]                                    // width * 3
0x14001edf9:        lea edi, [rax+rax]                                      // width * 6
0x14001edfc:        jmp allocate_sourceBuffer_1eeda
...
0x14001ee01: check_bpp_1ee01:
0x14001ee01:        cmp r13d, 8                                             // [47] check bits per pixel is 8
0x14001ee05:        jz depth(8)_1eed0
0x14001ee0b:        cmp r13d, 0Ah                                           // [47] check bits per pixel is 10
0x14001ee0f:        jz depth(a)_1eec0
0x14001ee15:        cmp r13d, 10h                                           // [47] check bits per pixel is 16
0x14001ee19:        jz depth(10)_1eeb1
...
0x14001eeb1: depth(10)_1eeb1:
0x14001eeb1:        mov [rbp+60h+lv_object_d0.vd_pixelType_bc], 'RB6I'      // [47] if bits per pixel is 16
0x14001eeb8:        lea eax, [rsi+rsi*2]                                    // width * 3
0x14001eebb:        lea edi, [rax+rax]                                      // width * 6
0x14001eebe:        jmp short allocate_sourceBuffer_1eeda
...
0x14001eec0: depth(a)_1eec0:
0x14001eec0:        mov [rbp+60h+lv_object_d0.vd_pixelType_bc], 'DP0B'      // [47] if bits per pixel is 10
0x14001eec7:        lea edi, [rsi*4+0]                                      // width * 4
0x14001eece:        jmp short allocate_sourceBuffer_1eeda
...
0x14001eed0: depth(8)_1eed0:
0x14001eed0:        mov [rbp+60h+lv_object_d0.vd_pixelType_bc], 'BGR8'      // [47] if bits per pixel is 8
0x14001eed7:        lea edi, [rsi+rsi*2]                                    // width * 3
```

The resulting pitch will then be stored into the object that was allocated on the stack. At [48], the pitch and the height will be multiplied as a signed value in order to calculate the amount of memory required to contain a frame that is decoded from a video. The result of this multiplication will then be passed to the method at [49]. Inside the method, the value 0xfff will be added to the signed size, which will then be passed to the call to `malloc` at [50]. If the addition of the value 0xfff to the signed product of the pitch (composed of the width and the bytes per pixel) and the height is larger than 64-bits, this can result in the allocation being of a smaller size than expected. Using this undersized buffer can cause a heap-based buffer overflow which can result in code execution under the context of the application.

```
0x14001eeda: allocate_sourceBuffer_1eeda:
0x14001eeda:        mov [rbp+60h+lv_object_d0.v_pitch_20], rdi
0x14001eede:        imul rdi, rcx                                           // [48] multiply pitch by height from video container
0x14001eee2:        mov rdx, rdi                                            // decode size
0x14001eee5:        mov rcx, [rsp+160h+lp_redDecoder_108]                   // this
0x14001eeea:        call REDDecoder::allocateSourceBuffer_1e930             // [49] \ call function to allocate memory
0x14001eeef:        mov [rsp+160h+lp_srcBuffer_118], rax                    // store allocation
0x14001eef4:        test rax, rax
0x14001eef7:        jnz short allocate_success_1ef2f
\
0x14001e930: REDDecoder::allocateSourceBuffer_1e930:
0x14001e930:        mov [rsp+arg_0], rbx
0x14001e935:        push rdi
0x14001e936:        sub rsp, 20h
0x14001e93a:        mov rdi, rdx
0x14001e93d:        mov rbx, rcx
...
0x14001e966: alloc_1e966:
0x14001e966:        lea rcx, [rdi+0FFFh]                                    // [50] add 0xfff to size
0x14001e96d:        mov [rbx+REDDecoder.field_18], 0
0x14001e975:        mov [rbx+REDDecoder.v_decodeBufferSize_10], rdi
0x14001e979:        call cs:__imp_malloc                                    // [50] call malloc to allocate memory
0x14001e97f:        mov [rbx+REDDecoder.p_decodeBuffer_8], rax
0x14001e983:        mov rcx, rax
0x14001e986:        test rax, rax
0x14001e989:        jnz short return_success_1e9b0
```

Later at [51], the application will call into the R3D SDK to decode the specific frame from the video container. Afterwards, this frame will be copied into the undersized buffer using the signed value without the addition of 0xfff, resulting in heap corruption.

```
0x14001f0aa: loc_14001F0AA:
0x14001f0aa:        movsxd r8, [rsp+160h+lvd_frameNumber_124]           // frame number
0x14001f0af:        movsxd rdx, [rsp+160h+lvd_hdrMode?_128]             // header mode
0x14001f0b4:        lea r9, [rbp+60h+lv_object_d0]                      // object containing parameters
0x14001f0b8:        mov rcx, rdi                                        // this
0x14001f0bb:        call clipContainerList::parseFileContents_218d0     // [51] call R3D SDK to parse contents of file
0x14001f0c0: loc_14001F0C0:
0x14001f0c0:        mov ebx, eax
...
0x14001f1d3: memcpy_1f1d3:
0x14001f1d3:        imul esi, dword ptr [rsp+160h+lv_resultModeDivisor_64]
0x14001f1d8:        lea eax, [rsi+rsi*2]
0x14001f1db:        add eax, eax
0x14001f1dd: memcpy_1f1dd:
0x14001f1dd:        movsxd r8, eax                                      // size that is sign-extended
0x14001f1e0: memcpy_1f1e0:
0x14001f1e0:        mov rdx, [rsp+160h+lp_srcBuffer_118]                // source
0x14001f1e5:        mov rcx, rdi                                        // destination
0x14001f1e8:        call memcpy                                         // [52] copy into undersized buffer
```

Crash Information

Upon running the provided proof-of-concept against the port running the `DPDecoder` service and executing until the call to `memcpy` described in the advisory, the debugger will interrupt execution right before executing the call to `memcpy`. A number of exceptions are raised prior to this due to the provided proof-of-concept not containing correctly formatted frames. These exceptions are raised by the libkakadu library, handled by the R3D SDK, and will still result in the service queueing the video container in its list of files to decode.

```
0:034> g dpdecoder+1f1e8
(1bfc.608): C++ EH exception - code e06d7363 (first chance)
(1bfc.608): C++ EH exception - code e06d7363 (first chance)
(1bfc.608): C++ EH exception - code e06d7363 (first chance)
DPDecoder+0x1f1e8:
00007ff6`d054f1e8 e837bc0000      call    DPDecoder+0x2ae24 (00007ff6`d055ae24)
```

The parameters show the source, destination, and the length that are passed to memcpy.

```
0:003> r rcx, rdx, r8
rcx=000001ba726c0ea0 rdx=000001b9dc671000 r8=ffffffff9404815b
```

According to the !heap command, the allocation size is smaller than the length due to missing the sign-extension.

```
0:003> !heap -p -a @rdx
    address 000001b9dc671000 found in
    _DPH_HEAP_ROOT @ 1b9d59b1000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                            1b9d5a06f70:     1b9dc670ea0         9404915a -      1b9dc670000         9404b000
    00007ff977f8867b ntdll!RtlDebugAllocateHeap+0x000000000000003b
    00007ff977ebd255 ntdll!RtlpAllocateHeap+0x00000000000000f5
    00007ff977ebb44d ntdll!RtlpAllocateHeapInternal+0x0000000000000a2d
    00007ff97568fde6 ucrtbase!_malloc_base+0x0000000000000036
    00007ff6d054e97f DPDecoder+0x000000000001e97f
    00007ff6d054eeef DPDecoder+0x000000000001eeef
    00007ff6d054a7d2 DPDecoder+0x000000000001a7d2
    00007ff6d054ca04 DPDecoder+0x000000000001ca04
    00007ff6d054dbf9 DPDecoder+0x000000000001dbf9
    00007ff95ff8b63e pthreadVC2!pthread_rwlockattr_init+0x000000000000a152
    00007ff9756a1bb2 ucrtbase!thread_start<unsigned int (__cdecl*)(void *),1>+0x0000000000000042
    00007ff977577034 KERNEL32!BaseThreadInitThunk+0x0000000000000014
    00007ff977ee2651 ntdll!RtlUserThreadStart+0x0000000000000021
```

As the video frame did not decode correctly, the source buffer is left uninitialized.

```
0:003> dq @rcx
000001ba`726c0ea0  c0c0c0c0`c0c0c0c0 c0c0c0c0`c0c0c0c0
000001ba`726c0eb0  c0c0c0c0`c0c0c0c0 c0c0c0c0`c0c0c0c0
000001ba`726c0ec0  c0c0c0c0`c0c0c0c0 c0c0c0c0`c0c0c0c0
000001ba`726c0ed0  c0c0c0c0`c0c0c0c0 c0c0c0c0`c0c0c0c0
000001ba`726c0ee0  c0c0c0c0`c0c0c0c0 c0c0c0c0`c0c0c0c0
000001ba`726c0ef0  c0c0c0c0`c0c0c0c0 c0c0c0c0`c0c0c0c0
000001ba`726c0f00  c0c0c0c0`c0c0c0c0 c0c0c0c0`c0c0c0c0
000001ba`726c0f10  c0c0c0c0`c0c0c0c0 c0c0c0c0`c0c0c0c0
```

The base address of the decoder is located at 0x7ff6d0530000.

```
0:003> lm a dpdecoder
Browse full module list
start             end                 module name
00007ff6`d0530000 00007ff6`d0586000   DPDecoder   (no symbols)
```

Resuming execution crashes while reading off of a page during copy.

```
0:003> g
(1bfc.608): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
VCRUNTIME140!memcpy+0x180:
00007ff9`673b1470 c4a17e6f6c02e0  vmovdqu ymm5,ymmword ptr [rdx+r8-20h] ds:000001b9`706b913b=??
```

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 32-bit width and 32-bit height that is used in the overflowed calculation can be found at offset 0x34 and 0x38 of the file. This particular variation relies on the version information that is found at offset 0x8 of the file.

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 this 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.

The decoding parameters are submitted in the second packet by the client. This packet begins with a uint32_t, representing the length of a string containing the path to the filename to decode. After the filename is a uint32_t containing the header mode, frame number, decode mode, and bits-per-pixel. If the sum of the value 0xfff with the product of the dimensions of the image and the bytes calculated for the bits-per-pixel is larger than 64-bits, then this vulnerability is being triggered.

Credit

Discovered by a member of Cisco Talos.
https://talosintelligence.com/vulnerability_reports/

Vendor Response

There is a button to download DaVinci Resolve Free and this will give you download options for the most recent versions for Mac, Windows and Linux for both DaVinci Resolve 17 and DaVinci Resolve Studio 17.

https://www.blackmagicdesign.com/products/davinciresolve/

Timeline

2021-12-09 - Vendor Disclosure

2021-12-20 - Public Release

CREDIT

Discovered by a member of Cisco Talos.
