Talos Vulnerability Report

TALOS-2021-1411

Accusoft ImageGear parse_raster_data out-of-bounds write vulnerability

MARCH 31, 2022

CVE NUMBER

CVE-2021-40398

Summary

An out-of-bounds write vulnerability exists in the parse_raster_data functionality of Accusoft ImageGear 19.10. A specially-crafted malformed file can lead to memory corruption. An attacker can provide a malicious file to trigger this vulnerability.

Tested Versions

Accusoft ImageGear 19.10

Product URLs

ImageGear - https://www.accusoft.com/products/imagegear-collection/

CVSSv3 Score

8.1 - CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H

CWE

CWE-119 - Improper Restriction of Operations within the Bounds of a Memory Buffer

Details

The ImageGear library is a document-imaging developer toolkit that offers image conversion, creation, editing, annotation and more. It supports more than 100 formats such as DICOM, PDF, Microsoft Office and others.

There is a vulnerability in the `parse_raster_data` function, due to a buffer overflow caused by a missing buffer size check.
A specially-crafted PICT file can lead to an out-of-bounds write, which can result in memory corruption.

Trying to load a malformed PICT v1 file, we end up in the following situation:

```
0:000> g
(248c.1394): Access violation - code c0000005 (!!! second chance !!!)
eax=0c4fb013 ebx=0cd21000 ecx=00000001 edx=00000001 esi=0c4fb012 edi=0cd21000
eip=6731dd22 esp=0019f634 ebp=0019f64c iopl=0         nv up ei pl nz ac pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010217
MSVCR110!memcpy+0x2a:
6731dd22 f3a4            rep movs byte ptr es:[edi],byte ptr [esi]
```

When we look at the `edi` memory allocation we can see the buffer allocated is very small, only 1 byte:

```
0:000> !ext.heap -p -a edi
    address 0cd21000 found in
    _DPH_HEAP_ROOT @ 3db1000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                                 cb63034:          cd20ff8                1 -          cd20000             2000
    695ba8b0 verifier!AVrfDebugPageHeapAllocate+0x00000240
    7752f10e ntdll!RtlDebugAllocateHeap+0x00000039
    774970f0 ntdll!RtlpAllocateHeap+0x000000f0
    77496e3c ntdll!RtlpAllocateHeapInternal+0x0000104c
    77495dde ntdll!RtlAllocateHeap+0x0000003e
    6731daff MSVCR110!malloc+0x00000049
    674e64de igCore19d!AF_memm_alloc+0x0000001e
    675c82aa igCore19d!IG_mpi_page_set+0x000dc27a
    675c7751 igCore19d!IG_mpi_page_set+0x000db721
    674c13d9 igCore19d!IG_image_savelist_get+0x00000b29
    675008d7 igCore19d!IG_mpi_page_set+0x000148a7
    67500239 igCore19d!IG_mpi_page_set+0x00014209
    67495757 igCore19d!IG_load_file+0x00000047
    00402219 Fuzzme!fuzzme+0x00000019
    00402524 Fuzzme!fuzzme+0x00000324
    0040668d Fuzzme!fuzzme+0x0000448d
    760ffa29 KERNEL32!BaseThreadInitThunk+0x00000019
    774b7a9e ntdll!__RtlUserThreadStart+0x0000002f
    774b7a6e ntdll!_RtlUserThreadStart+0x0000001b
```

The call stack will indicate where the `memcpy` is happening:

```
0:000> kb
 # ChildEBP RetAddr      Args to Child
00 0019f638 6748f9a6     0cd21000 0c4fb012 00000001 MSVCR110!memcpy+0x2a
WARNING: Stack unwind information not available. Following frames may be wrong.
01 0019f64c 675c9b43     0cd21000 0c4fb012 00000001 igCore19d+0xf9a6
02 0019f66c 675c856f     0cd20ff8 0cd228f8 0c4fb103 igCore19d!IG_mpi_page_set+0xddb13
03 0019f734 675c7751     0019fc3c 1000001e 0e44eff8 igCore19d!IG_mpi_page_set+0xdc53f
04 0019fbb4 674c13d9     0019fc3c 0e44eff8 00000001 igCore19d!IG_mpi_page_set+0xdb721
05 0019fbec 675008d7     00000000 0e44eff8 0019fc3c igCore19d!IG_image_savelist_get+0xb29
06 0019fe68 67500239     00000000 0524dfd0 00000001 igCore19d!IG_mpi_page_set+0x148a7
07 0019fe88 67495757     00000000 0524dfd0 00000001 igCore19d!IG_mpi_page_set+0x14209
08 0019fea8 00402219     0524dfd0 0019febc 00000001 igCore19d!IG_load_file+0x47
09 0019fec0 00402524     0524dfd0 0019fef8 051aff48 Fuzzme!fuzzme+0x19
0a 0019ff28 0040668d     00000005 051a8f68 051aff48 Fuzzme!fuzzme+0x324
0b 0019ff70 760ffa29     003fe000 760ffa10 0019ffdc Fuzzme!fuzzme+0x448d
0c 0019ff80 774b7a9e     003fe000 721cdcd8 00000000 KERNEL32!BaseThreadInitThunk+0x19
0d 0019ffdc 774b7a6e     ffffffff 774d8a44 00000000 ntdll!__RtlUserThreadStart+0x2f
0e 0019ffec 00000000     00406715 003fe000 00000000 ntdll!_RtlUserThreadStart+0x1b
```

The `igCore19d+0xf9a6` is just a wrapper of `memcpy`, so the interesting callback is `igCore19d!IG_mpi_page_set+0xddb13`, which corresponds to the function `write_into_dest_buffer` with the following pseudo code. The `memcpy` is performed at LINE54 as `edi` contains the `dest_buffer` pointer.

```
LINE1  int __stdcall write_into_dest_buffer(void *_dest_buffer, void *source, int size_source, int size_dest)
LINE2  {
         [...]
LINE13
LINE14   dest_buffer = _dest_buffer;
LINE15   _source_buffer = (int)source;
LINE16   max_source_addr = (char *)source + size_source;
LINE17   max_dest_addr = (char *)_dest_buffer + size_dest;
LINE18   one_byte = (char *)source + size_source - 1;
LINE19   size_sourcea = (char *)source + size_source;
LINE20   if ( source >= one_byte )
LINE21     return 0;
LINE22   // loop counter is size_source parameter
LINE23   do
LINE24   {
LINE25     if ( dest_buffer >= max_dest_addr )
LINE26       break;
LINE27     byte_value_from_source = *(_BYTE *)_source_buffer;
LINE28     if ( *(_BYTE *)_source_buffer >= 0x80u )
LINE29     {
LINE30       if ( *(_BYTE *)_source_buffer <= 0x80u )
LINE31       {
LINE32         ++_source_buffer;
LINE33       }
LINE34       else
LINE35       {
LINE36         size_1 = 257 - byte_value_from_source;
LINE37         if ( (char *)dest_buffer + size_1 > max_dest_addr )
LINE38           size_1 = max_dest_addr - (_BYTE *)dest_buffer;
LINE39         OS_memset(dest_buffer, *(_BYTE *)(_source_buffer + 1), size_1);
LINE40         max_dest_addr = (char *)_dest_buffer + size_dest;
LINE41         max_source_addr = size_sourcea;
LINE42         dest_buffer = (char *)dest_buffer + size_1;
LINE43         _source_buffer += 2;
LINE44       }
LINE45     }
LINE46     else
LINE47     {
LINE48       memcpy_size = byte_value_from_source + 1; // read from source buffer
LINE49       if ( (char *)dest_buffer + memcpy_size > max_dest_addr ) // check if destination has enough room
LINE50         memcpy_size = max_dest_addr - (_BYTE *)dest_buffer;
LINE51       if ( memcpy_size + _source_buffer + 1 > (unsigned int)max_source_addr ) // check if source buffer has enough room
LINE52         memcpy_size = (size_t)&max_source_addr[-_source_buffer - 1];
LINE53       source_buffer = (void *)(_source_buffer + 1);
LINE54       OS_memcpy(dest_buffer, source_buffer, memcpy_size);
LINE55       max_dest_addr = (char *)_dest_buffer + size_dest;
LINE56       max_source_addr = size_sourcea;
LINE57       dest_buffer = (char *)dest_buffer + memcpy_size;
LINE58       _source_buffer = (int)source_buffer + memcpy_size;
LINE59     }
LINE60   }
LINE61   while ( _source_buffer < (unsigned int)(max_source_addr - 1) );
LINE62   return 0;
LINE63 }
```

The bound checking for the value of the `memcpy` size (`memcpy_size`) is controlled by `dest_buffer` and `max_dest_addr` at LINE49 and LINE50.
This is to ensure that `memcpy_size`, which is derived from `byte_value_from_source` at LINE48 and coming from the source buffer at LINE27, matches the `dest_buffer`.
The same logic applies to `_source_buffer` at LINE51 and LINE52. The `max_dest_addr` is derived, at LINE17, from the argument `size_dest` passed to the function.

We need to go earlier in the call stack to see where these parameters are coming from. The source of our interesting call to `write_into_dest_buffer` is happening at LINE321 in the function `parse_raster_data`.

```
LINE64  int __stdcall parse_raster_data(
LINE65          mys_table_function *mys_table_func,
LINE66          uint kind_of_heap,
LINE67          undefined4 param_3,
LINE68          pict_header *pict_header,
LINE69          HIGDIBINFO hDib)
LINE70  {
        [...]
LINE136
LINE137   _kind_of_heap = (void *)kind_of_heap;
LINE138   _pict_header = pict_header;
LINE139   buffer2_28 = 0;
LINE140   io_buffer.buffer_size = 0;
LINE141   DIB_width_get(hDib);
LINE142   height = DIB_height_get(hDib);
LINE143   _raster_size = IO_raster_size_get(hDib);
LINE144   table_of_pic_raster_data = pict_header->table_of_pic_raster_data;
LINE145   __raster_size = _raster_size;
LINE146   pixel_size = table_of_pic_raster_data->pixel_size;
LINE147   num_planes = table_of_pic_raster_data->num_planes;
LINE148   size__destination_buffer = table_of_pic_raster_data->bounding_rectangle.lower_right_corner_y
LINE149                             - table_of_pic_raster_data->bounding_rectangle.top_left_corner_Y;
LINE150   _pixel_size = pixel_size;
LINE151   _err_count = IO_DIB_create_ex(mys_table_func, hDib);
LINE152   if ( _err_count )
LINE153     return AF_err_record_set((LPCHAR)"..\\..\\..\\..\\Common\\Formats\\pctwread.c", 759, -2464, 0, 0, 0, 0);
LINE154   rgb_dest = (char *)AF_memm_alloc(
LINE155                       (int)_kind_of_heap,
LINE156                       __raster_size,
LINE157                       (int)"..\\..\\..\\..\\Common\\Formats\\pctwread.c",
LINE158                       761);
LINE159   __destination_buffer = AF_memm_alloc(
LINE160                           (int)_kind_of_heap,
LINE161                           5 * size__destination_buffer,
LINE162                           (int)"..\\..\\..\\..\\Common\\Formats\\pctwread.c",
LINE163                           762);
LINE164   buffer3 = (char *)AF_memm_alloc(
LINE165                      (int)_kind_of_heap,
LINE166                      height * __raster_size,
LINE167                      (int)"..\\..\\..\\..\\Common\\Formats\\pctwread.c",
LINE168                      764);
LINE169   buffer4 = AF_memm_alloc(
LINE170               (int)_kind_of_heap,
LINE171               height * __raster_size,
LINE172               (int)"..\\..\\..\\..\\Common\\Formats\\pctwread.c",
LINE173               766);
LINE174   if ( rgb_dest && __destination_buffer && buffer3 )
LINE175   {
LINE176     if ( pixel_size == 32 && num_planes == 4 )
LINE177       buffer2_28 = 1;
LINE178   }
LINE179   else
LINE180   {
LINE181     _err_count = AF_err_record_set((LPCHAR)"..\\..\\..\\..\\Common\\Formats\\pctwread.c", 769, -1000, 0, 0, 0, 0);
LINE182   }
LINE183   _buffer3 = buffer3;
LINE184   OS_memset(buffer3, 0, height * __raster_size);
LINE185   _kind_of_heap_1 = _kind_of_heap;
LINE186   _err_count_iobinit = IOb_init(mys_table_func, (int)_kind_of_heap, &io_buffer, 0x5000u, 1);
LINE187   _err_count += _err_count_iobinit;
LINE188   if ( !_err_count )
LINE189   {
LINE190     _pict_header_1 = _pict_header;
LINE191     IO_attribute_set(mys_table_func, 4u, &_pict_header->original_horizontal_pixel_per_inch);
LINE192     v14 = 0;
LINE193     offset_mem = 0;
LINE194     v52 = 0;
LINE195     index_raster_data = 0;
LINE196     do
LINE197     {
LINE198       if ( v14 >= _pict_header_1->num_of_pic_raster_data )
LINE199         break;
LINE200       v16 = (pic_raster_data *)((char *)_pict_header_1->table_of_pic_raster_data + offset_mem);
LINE201       _num_planes = v16->num_planes;
LINE202       if ( (unsigned int)_num_planes >= 3 )
LINE203         _num_planes = 3;
LINE204       for ( i = 0; i < _num_planes; ++i )
LINE205         three_bytes[i] = v16->component_size;
LINE206       x_diff = v16->four_short_1.lower_right_corner_x - v16->four_short_1.top_left_corner_x;
LINE207       y_diff = v16->four_short_1.lower_right_corner_y - v16->four_short_1.top_left_corner_Y;
LINE208       _ydiff = y_diff;
LINE209       v50 = x_diff;
LINE210       IO_raster_size_calc(y_diff, _num_planes, three_bytes);
LINE211       if ( pixel_size == 1 )
LINE212         _raster_size_1 = DIB1bit_packed_raster_size_calc(_ydiff);
LINE213       else
LINE214         _raster_size_1 = DIBStd_raster_size_calc_simple(
LINE215                            _ydiff,
LINE216                            _num_planes,
LINE217                            _pict_header->table_of_pic_raster_data[index_raster_data / 0x440].component_size);
LINE218       dest_size = _raster_size_1;
LINE219       raster_data = &_pict_header->table_of_pic_raster_data[index_raster_data / 0x440];
LINE220       _new_offset = raster_data->some_distance_to_move_into;
LINE221       some_length = raster_data->bounding_rectangle.lower_right_corner_y
LINE222                   - raster_data->bounding_rectangle.top_left_corner_Y;
LINE223       IOb_seek(&io_buffer, _new_offset, SEEK_SET);
LINE224       _rgb_dest = rgb_dest;
LINE225       OS_memset(rgb_dest, 0, __raster_size);
LINE226       if ( pixel_size != 24 )
LINE227         _raster_size_1 = _pict_header->table_of_pic_raster_data[index_raster_data / 0x440].next_offset;
LINE228       raster_size = _raster_size_1;
LINE229       packet_type = (unsigned __int16)_pict_header->table_of_pic_raster_data[index_raster_data / 0x440].packet_type;
LINE230       v25 = 0;
LINE231       io_buffer.size_buffer = packet_type;
LINE232       _pict_header_8 = 0;
LINE233       if ( !_err_count )
LINE234       {
LINE235         while ( v25 < v50 )
LINE236         {
LINE237           if ( _raster_size_1 < 8 || packet_type == 1 || packet_type == 2 && pixel_size >= 24 )
LINE238           {
LINE239             v32 = (void *)read_block(&io_buffer, _raster_size_1);
LINE240             v33 = v32;
LINE241             if ( !v32 )
LINE242             {
LINE243               v38 = AF_err_record_set((LPCHAR)"..\\..\\..\\..\\Common\\Formats\\pctwread.c", 855, -2100, 0, 0, 0, 0);
LINE244               goto LABEL_55;
```

```
LINE245             }
LINE246             switch ( pixel_size )
LINE247             {
LINE248               case 1:
LINE249               case 4:
LINE250               case 8:
LINE251                 OS_memcpy(rgb_dest, v32, _raster_size_1);
LINE252                 break;
LINE253               case 16:
LINE254                 sub_10148800(rgb_dest, (int)v32, y_diff);
LINE255                 break;
LINE256               case 24:
LINE257               case 32:
LINE258                 v34 = y_diff;
LINE259                 copy_rgb_data(rgb_dest, (char *)v32, y_diff, some_length, buffer2_28);
LINE260                 if ( buffer2_28 )
LINE261                   OS_memcpy(__destination_buffer, v33, v34);
LINE262                 break;
LINE263               default:
LINE264                 break;
LINE265             }
LINE266           }
LINE267           else
LINE268           {
LINE269             if ( _raster_size_1 <= 250 )
LINE270             {
LINE271               if ( !IOb_byte_read(&io_buffer, &_length) )
LINE272               {
LINE273                 v38 = AF_err_record_set((LPCHAR)"..\\..\\..\\..\\Common\\Formats\\pctwread.c", 893, -2100, 0, 0, 0, 0);
LINE274                 goto LABEL_55;
LINE275               }
LINE276               _len_to_read = _length;
LINE277             }
LINE278             else
LINE279             {
LINE280               if ( !IOb_short_read(&io_buffer, &tbd) )
LINE281               {
LINE282                 v38 = AF_err_record_set((LPCHAR)"..\\..\\..\\..\\Common\\Formats\\pctwread.c", 884, -2100, 0, 0, 0, 0);
LINE283 LABEL_55:
LINE284                 _err_count = v38;
LINE285                 break;
LINE286               }
LINE287               _len_to_read = (unsigned __int16)tbd;
LINE288             }
LINE289             __source_buffer = (void *)read_block(&io_buffer, _len_to_read);
LINE290             _source_data = (int)__source_buffer;
LINE291             if ( !__source_buffer )
LINE292             {
LINE293               v38 = AF_err_record_set((LPCHAR)"..\\..\\..\\..\\Common\\Formats\\pctwread.c", 901, -2100, 0, 0, 0, 0);
LINE294               goto LABEL_55;
LINE295             }
LINE296             // biBitCount = 0x1f
LINE297             switch ( pixel_size )
LINE298             {
LINE299               case 1:
LINE300               case 8:
LINE301                 write_into_dest_buffer(_rgb_dest, __source_buffer, _len_to_read, dest_size);
LINE302                 break;
LINE303               case 4:
LINE304                 v28 = __raster_size / 2;
LINE305                 v29 = &rgb_dest[__raster_size / 2];
LINE306                 write_into_dest_buffer(v29, (void *)_source_data, _len_to_read, dest_size);
LINE307                 v42 = v29;
LINE308                 pixel_size = _pixel_size;
LINE309                 iIG_IP_raster_unpack(1, _pixel_size, size__destination_buffer, v42, rgb_dest, v28);
LINE310                 break;
LINE311               case 16:
LINE312                 v44 = _len_to_read;
LINE313                 v30 = (int)__destination_buffer;
LINE314                 sub_10149A10(__destination_buffer, (int)__source_buffer, v44, raster_size);
LINE315                 sub_10148800(_rgb_dest, v30, y_diff);
LINE316                 break;
LINE317               case 24:
LINE318               case 32:
LINE319                 size_source = _len_to_read;
LINE320                 rgb_source = (char *)__destination_buffer;
LINE321                 write_into_dest_buffer(__destination_buffer, __source_buffer, size_source, raster_size);
LINE322                 copy_rgb_data(_rgb_dest, rgb_source, y_diff, some_length, buffer2_28);
LINE323                 break;
LINE324               default:
LINE325                 break;
LINE326             }
LINE327           }
LINE328           v35 = _pict_header->table_of_pic_raster_data;
LINE329           top_left_corner_Y = v35[index_raster_data / 0x440].field_2A.top_left_corner_Y;
LINE330           v37 = &buffer3[__raster_size * (_pict_header_8 + v35[index_raster_data / 0x440].field_2A.top_left_corner_x)];
LINE331           if ( pixel_size < 8 )
LINE332           {
LINE333             OS_memcpy(&v37[top_left_corner_Y / (8 / pixel_size)], rgb_dest, dest_size);
LINE334           }
LINE335           else if ( pixel_size == 8 )
LINE336           {
LINE337             OS_memcpy(&v37[top_left_corner_Y], rgb_dest, dest_size);
LINE338           }
LINE339           else
LINE340           {
LINE341             OS_memcpy(&v37[top_left_corner_Y * ((pixel_size > 16) + 2)], rgb_dest, dest_size);
LINE342           }
LINE343           _raster_size_1 = raster_size;
LINE344           _rgb_dest = rgb_dest;
LINE345           packet_type = io_buffer.size_buffer;
LINE346           v25 = ++_pict_header_8;
LINE347         }
LINE348       }
LINE349       _pict_header_1 = _pict_header;
LINE350       v14 = v52 + 1;
LINE351       offset_mem = index_raster_data + 0x440;
LINE352       ++v52;
LINE353       index_raster_data += 0x440;
LINE354     }
LINE355     while ( !_err_count );
LINE356     v39 = 0;
LINE357     if ( height > 0 )
LINE358     {
LINE359       v40 = buffer3;
LINE360       v41 = __raster_size;
LINE361       do
LINE362       {
```

```
LINE363         IO_raster_set(mys_table_func, (int)v40, v39++, v41);
LINE364         v40 += v41;
LINE365       }
LINE366       while ( v39 < height );
LINE367     }
LINE368     _kind_of_heap_1 = _kind_of_heap;
LINE369     _buffer3 = buffer3;
LINE370   }
LINE371   IOb_done(&io_buffer);
LINE372   if ( __destination_buffer )
LINE373     AF_memm_free(_kind_of_heap_1, __destination_buffer, (int)"..\\..\\..\\..\\Common\\Formats\\pctwread.c", 980);
LINE374   if ( rgb_dest )
LINE375     AF_memm_free(_kind_of_heap_1, rgb_dest, (int)"..\\..\\..\\..\\Common\\Formats\\pctwread.c", 983);
LINE376   if ( _buffer3 )
LINE377     AF_memm_free(_kind_of_heap_1, _buffer3, (int)"..\\..\\..\\..\\Common\\Formats\\pctwread.c", 986);
LINE378   if ( buffer2_28 && buffer4 )
LINE379     AF_memm_free(_kind_of_heap_1, buffer4, (int)"..\\..\\..\\..\\Common\\Formats\\pctwread.c", 989);
LINE380   return AF_error_check();
LINE381 }
```

The allocation of the destination buffer `__destination_buffer` is performed at LINE159, and the size of the allocation buffer `size__destination_buffer` is computed earlier at LINE148.
At LINE159 there is a potential integer overflow as the size allocated is the result of the multiplication by `5 * size__destination_buffer`.
The size of this allocation however is different from the size used by the `write_into_dest_buffer` function that eventually writes to the buffer. This is where the crux of the matter lies: the function writing to the buffer is not respecting the bounds of the allocation.

In fact, the `size_dest` argument passed to `write_into_dest_buffer` corresponds to `raster_size` at LINE321. The `raster_size` is derived from `_raster_size_1` at LINE228.
The value for `_raster_size_1` can be computed from a different place at LINE227 or LINE212 depending on the values contained in `pixel_size`.

When `_raster_size_1` is computed in LINE227, it's computed from an offset structure field named `next_offset` in an earlier process that parses the opcode from the PICT file in the function `get_data_from_opcode`:

```
LINE383 AT_ERRCOUNT __stdcall get_data_from_opcode(
LINE384         lp_iobuffer lp_iobuffer,
LINE385         unsigned __int16 opcode,
LINE386         pic_raster_data *pic_raster_data)
LINE387 {
        [...]
LINE400
LINE401   current_pos = IOb_tell(lp_iobuffer);
LINE402   OS_memset(pic_raster_data, 0, 0x440u);
LINE403   pic_raster_data->opcode = opcode;
LINE404   pic_raster_data->offset_in_file = current_pos;
LINE405   if ( opcode > 0x9Bu )
LINE406   {
LINE407     if ( (unsigned int)opcode - 0x8200 <= 1 )
LINE408     {
LINE409       v4 = lp_iobuffer;
LINE410       pic_raster_data->field_0x0 = 4 * (opcode != 0x8200) + 4;
LINE411       if ( !fill_in_pict2_buffer(lp_iobuffer, opcode, &_pict_2_buffer) )
LINE412       {
LINE413         pic_raster_data->next_offset = v11;
LINE414         pic_raster_data->packet_type = v12;
LINE415         v7 = _pict_2_buffer.field_0x9e;
LINE416         pic_raster_data->pixel_size = _pict_2_buffer.field_0x9e;
LINE417         pic_raster_data->component_size = v7;
LINE418         pic_raster_data->original_horizontal_pixel_per_inch = _pict_2_buffer.field_0x70;
LINE419         pic_raster_data->original_vertical_pixel_per_inch = _pict_2_buffer.field_0x74;
LINE420         *(_DWORD *)&pic_raster_data->bounding_rectangle.top_left_corner_x = 0;
LINE421         pic_raster_data->bounding_rectangle.lower_right_corner_x = _pict_2_buffer.field_0x6e;
LINE422         pic_raster_data->bounding_rectangle.lower_right_corner_y = _pict_2_buffer.field_0x6c;
LINE423         pic_raster_data->num_planes = 1;
LINE424         if ( opcode == 0x8200 )
LINE425         {
LINE426           *(_DWORD *)&pic_raster_data->four_short.top_left_corner_x = *(_DWORD *)&_pict_2_buffer.four_short;
LINE427           v8 = *(dword *)((char *)&_pict_2_buffer.current_offset + 2);
LINE428         }
LINE429         else
LINE430         {
LINE431           *(_DWORD *)&pic_raster_data->four_short.top_left_corner_x = v13;
LINE432           v8 = v14;
LINE433         }
LINE434         *(_DWORD *)&pic_raster_data->four_short.lower_right_corner_x = v8;
LINE435         pic_raster_data->some_shift_offset = _pict_2_buffer.opcode_8200_or_8201;
LINE436         goto LABEL_12;
LINE437       }
LINE438     }
LINE439   }
LINE440   else if ( opcode >= 0x98u || (unsigned int)opcode - 0x90 <= 1 )
LINE441   {
LINE442     v4 = lp_iobuffer;
LINE443     if ( !possible_parse_pixmap(lp_iobuffer, opcode, &pict_1_buffer, (int)&pic_raster_data->dib_palette) )
LINE444     {
LINE445       rowbytes = pict_1_buffer.rowbytes;
LINE446       pic_raster_data->field_0x0 = ((pict_1_buffer.rowbytes & 0x8000) != 0) + 1;
LINE447       pic_raster_data->packet_type = pict_1_buffer.packet_type;
LINE448       pic_raster_data->pixel_size = pict_1_buffer.pixel_size;
LINE449       pic_raster_data->num_planes = pict_1_buffer.num_planes;
LINE450       pic_raster_data->component_size = pict_1_buffer.component_size;
LINE451       pic_raster_data->original_horizontal_pixel_per_inch = pict_1_buffer.original_horizontal_pixel_per_inch;
LINE452       pic_raster_data->original_vertical_pixel_per_inch = pict_1_buffer.original_vertical_pixel_per_inch;
LINE453       pic_raster_data->bounding_rectangle = pict_1_buffer.bounding_rectangle;
LINE454       pic_raster_data->four_short = pict_1_buffer.four_short;
LINE455       *(_DWORD *)&pic_raster_data->four_short_1.top_left_corner_x = *(_DWORD *)&pict_1_buffer.four_short_1.top_left_corner_x;
LINE456       v6 = *(_DWORD *)&pict_1_buffer.four_short_1.lower_right_corner_x;
LINE457       pic_raster_data->next_offset = rowbytes & 0x7FFF;
LINE458       *(_DWORD *)&pic_raster_data->four_short_1.lower_right_corner_x = v6;
LINE459 LABEL_12:
LINE460       pic_raster_data->some_distance_to_move_into = IOb_tell(v4);
LINE461     }
LINE462   }
LINE463   return AF_error_check();
LINE464 }
```

At LINE457 we can see the variable `next_offset` corresponding to our `_raster_size_1` seen previously, obtained from `rowbytes`, which is coming from `pict_1_buffer.rowbytes` (LINE445).
As the function `possible_parse_pixmap` is quite large, you can see part of it below and observe the `rowbytes` corresponds to the short (LINE500) after the PICT opcode inside the file:

```
LINE465 BOOL __stdcall possible_parse_pixmap(
LINE466         lp_iobuffer lp_iobuffer,
LINE467         __int16 opcode,
LINE468         pict_1_buffer *pict_1_buffer,
LINE469         int four_short)
LINE470 {
        [...]
LINE493
LINE494   _pict1_buffer = pict_1_buffer;
LINE495   OS_memset(pict_1_buffer, 0, 0x44u);
LINE496   _lp_iobuffer = lp_iobuffer;
LINE497   if ( opcode == 0x9A || opcode == 0x9B )
LINE498     IOb_seek(lp_iobuffer, 4, SEEK_CUR);
LINE499   _current_pos = IOb_tell(lp_iobuffer);
LINE500   IOb_short_read(lp_iobuffer, &pict_1_buffer->rowbytes);
LINE501   pict_read_4_short(lp_iobuffer, &pict_1_buffer->bounding_rectangle.top_left_corner_x);
LINE502   if ( (unsigned int)IOb_tell(lp_iobuffer) - _current_pos == 10 )
LINE503   {
LINE504     if ( pict_1_buffer->rowbytes >= 0 )
LINE505     {
LINE506       if ( (opcode == 0x91 || opcode == 0x99) && seek_toward_from_reading_short(lp_iobuffer) )// if (opcode == 145 or opcode == 143)
LINE507                                                 // read_short_value
LINE508                                                 // if value is positive then seek from current -minus2
LINE509                                                 // otherwise error
LINE510         return 1;
LINE511       pict_1_buffer->component_size = 1;
LINE512       pict_1_buffer->original_horizontal_pixel_per_inch = 0x48;
LINE513       pict_1_buffer->original_vertical_pixel_per_inch = 0x48;
LINE514       *(_DWORD *)&pict_1_buffer->pixel_size = 65537;
LINE515       if ( four_short )
LINE516       {
LINE517         *(_DWORD *)four_short = 0xFFFFFF;
LINE518         *(_DWORD *)(four_short + 4) = 0;
LINE519       }
LINE520     }
LINE521     else
LINE522     {
LINE523       _current_pos_1 = IOb_tell(lp_iobuffer);
LINE524       IOb_short_read(lp_iobuffer, &pict_1_buffer->version);
LINE525       IOb_short_read(lp_iobuffer, &pict_1_buffer->packet_type);
LINE526       IOb_dword_read(lp_iobuffer, &pict_1_buffer->packed_size);
LINE527       IOb_dword_read(lp_iobuffer, &pict_1_buffer->original_horizontal_pixel_per_inch);
LINE528       IOb_dword_read(lp_iobuffer, &pict_1_buffer->original_vertical_pixel_per_inch);
LINE529       IOb_short_read(lp_iobuffer, &pict_1_buffer->pixel_type);
LINE530       IOb_short_read(lp_iobuffer, &pict_1_buffer->pixel_size);
LINE531       IOb_short_read(lp_iobuffer, &pict_1_buffer->num_planes);
LINE532       IOb_short_read(lp_iobuffer, &pict_1_buffer->component_size);
LINE533       IOb_dword_read(lp_iobuffer, &pict_1_buffer->offset_to_next_color_plane);
LINE534       IOb_dword_read(lp_iobuffer, &pict_1_buffer->reserved);
LINE535       IOb_dword_read(lp_iobuffer, &pict_1_buffer->id_color_table);
         [...]
LINE626   _4_short = pict_read_4_short(_lp_iobuffer, &_pict1_buffer->four_short.top_left_corner_x);
LINE627   v23 = pict_read_4_short(_lp_iobuffer, &_pict1_buffer->four_short_1.top_left_corner_x) + _4_short;
LINE628   IOb_seek(_lp_iobuffer, 2, SEEK_CUR);
LINE629   if ( opcode == 0x91 || opcode == 0x99 || opcode == 0x9B )
LINE630     v23 += seek_toward_from_reading_short(_lp_iobuffer);
LINE631   return v23 != 0;
LINE632 }
```

So in the case of `pixel_size` equal to 32 (condition at LINE257), the `raster_size` derived from `_raster_size_1` is computed from `rowbytes` where the value is bound to 0x7FFF, as you can see at LINE457.
When `pixel_size` is equal to 16 (condition at LINE311), the logic stays the same. `pixel_size` is stored at LINE448.

In the other case, we can observe `_raster_size_1` is computed at LINE212 with the call to `DIB1bit_packed_raster_size_calc`, below the pseudo code.

```
unsigned int __stdcall DIB1bit_packed_raster_size_calc(int a1)
{
  return ((a1 + 31) >> 3) & 0xFFFFFFFC;
}
```

Now the argument of this function, represented by the variable `_ydiff`, is derived from the subtraction of frame fields `v16->field_2A.lower_right_corner_y` and `v16->field_2A.top_left_corner_Y` at LINE207.
These fields are also read from the file through the `possible_parse_pixmap` first at LINE627, then potentially modified after in the process of the entire function `pctwread`:

```
LINE633  AT_ERRCOUNT __stdcall pctwread(
LINE634          mys_table_function *mys_table_function_obj,
LINE635          uint kind_of_heap,
LINE636          pict_header *pict_header,
LINE637          void *dest_bytes,
LINE638          int a5)
LINE639  {
        [...]
LINE730    _pict_header = pict_header;
LINE731    pict_header->original_horizontal_pixel_per_inch.nUnits = 0;
LINE732    io_buffer.buffer_size = 0;
LINE733    IO_byte_order_set(mys_table_function_obj, 1);
LINE734    // prepare some io_buffer
LINE735    error_iobinit = (__int16 *)IOb_init(mys_table_function_obj, kind_of_heap, &io_buffer, 0x5000u, 1);
LINE736    _error_status = error_iobinit;
LINE737    if ( error_iobinit )
LINE738      return AF_error_check();
LINE739    // get current offset in file
LINE740    start_header = IOb_tell(&io_buffer);
LINE741    // move from file to 512 offset
LINE742    IOb_seek(&io_buffer, 512, SEEK_CUR);
LINE743    // init pict_header struct
LINE744    OS_memset(pict_header, 0, 0x48u);
LINE745    // read first short
LINE746    IOb_short_read(&io_buffer, &pict_header->size_of_file);
LINE747    // read frame data
LINE748    pict_read_4_short(&io_buffer, &pict_header->pict_frame_72_dpi.top_left_corner_x);
LINE749    first_non_null_byte = 0;
LINE750    *(_DWORD *)pict_version = 0;
LINE751    do
LINE752    {
LINE753      if ( first_non_null_byte >= 20 )
LINE754        break;
LINE755      IOb_short_read(&io_buffer, pict_version);
LINE756      ++first_non_null_byte;
LINE757    }
LINE758    while ( !pict_version[0] );
LINE759    // Not Processed
LINE760    // ------------------------------
LINE761    if ( IO_tag_set_exists(mys_table_function_obj) )
LINE762      IO_tag_word_set(mys_table_function_obj, 0x1036, (int)pict_version);
LINE763    // ------------------------------
LINE764    // Validating we don't have extra bytes
LINE765    current_pos_1 = IOb_tell(&io_buffer);
LINE766    if ( current_pos_1 - start_header == 2 * first_non_null_byte + 0x20A )
LINE767    {
LINE768      if ( pict_version[0] == 0x1101 || pict_version[0] == 0x11 )
LINE769      {
LINE770        pict_header->pict_version = pict_version[0];
LINE771        goto pict_valid_header;
LINE772      }
LINE773      // ------------------------------------------------------------
LINE774      // Invalid pict header
LINE775      error_current_offset_in_file = IOb_tell(&io_buffer);
LINE776      error_iobinit = (__int16 *)AF_err_record_set(
LINE777                                   (LPCHAR)"..\\..\\..\\..\\Common\\Formats\\pctwread.c",
LINE778                                   481,
LINE779                                   -2060,
LINE780                                   0,
LINE781                                   error_current_offset_in_file,
LINE782                                   0,
LINE783                                   0);
LINE784      _error_status = error_iobinit;
LINE785      // ----------------------------------------
LINE786    }
LINE787    else
LINE788    {
LINE789      // ------------------------------------------------------------
LINE790      // Invalid pict header
LINE791      v9 = IOb_tell(&io_buffer);
LINE792      error_iobinit = (__int16 *)AF_err_record_set(
LINE793                                   (LPCHAR)"..\\..\\..\\..\\Common\\Formats\\pctwread.c",
LINE794                                   479,
LINE795                                   -2100,
LINE796                                   0,
LINE797                                   v9,
LINE798                                   0,
LINE799                                   0);
LINE800      _error_status = error_iobinit;
LINE801      // ------------------------------------
LINE802    }
LINE803    if ( error_iobinit )
LINE804      goto pict_v1;
LINE805  pict_valid_header:
LINE806    if ( pict_header->pict_version != 0x11 )
LINE807      goto pict_v1;
LINE808    current_pos_2 = IOb_tell(&io_buffer);
LINE809    IOb_short_read(&io_buffer, pict_version);
LINE810    IOb_short_read(&io_buffer, (__int16 *)&io_buffer.size_buffer);
LINE811    IOb_dword_read(&io_buffer, &dword_value);
LINE812    if ( (unsigned int)IOb_tell(&io_buffer) - current_pos_2 == 8 )
LINE813    {
LINE814      if ( pict_version[0] == 0x2FF && LOWORD(io_buffer.size_buffer) == 0xC00 )
LINE815      {
LINE816        v13 = dword_value & 0xFFFF0000;
LINE817        pict_header->header_value_FFFF = dword_value & 0xFFFF0000;
LINE818        if ( v13 != 0xFFFE0000 )
LINE819        {
LINE820          IOb_seek(&io_buffer, 20, SEEK_CUR);
LINE821          goto pict_v1;
LINE822        }
LINE823        current_pos_3 = IOb_tell(&io_buffer);
LINE824        pict_header->original_horizontal_pixel_per_inch.nUnits = 3;
LINE825        HIDWORD(pict_header->original_horizontal_pixel_per_inch.xResNumerator) = 1;
LINE826        HIDWORD(pict_header->original_horizontal_pixel_per_inch.yResNumerator) = 1;
LINE827        IOb_dword_read(&io_buffer, &dword_value);
LINE828        LODWORD(pict_header->original_horizontal_pixel_per_inch.xResNumerator) = HIWORD(dword_value);
LINE829        IOb_dword_read(&io_buffer, &dword_value);
LINE830        LODWORD(pict_header->original_horizontal_pixel_per_inch.yResNumerator) = HIWORD(dword_value);
LINE831        pict_read_4_short(&io_buffer, &pict_header->pict_frame_original.top_left_corner_x);
LINE832        IOb_seek(&io_buffer, 4, SEEK_CUR);
LINE833        if ( (unsigned int)IOb_tell(&io_buffer) - current_pos_3 == 20 )
LINE834          goto pict_v1;
LINE835        current_pos_4 = IOb_tell(&io_buffer);
LINE836        v12 = (__int16 *)AF_err_record_set(
LINE837                           (LPCHAR)"..\\..\\..\\..\\Common\\Formats\\pctwread.c",
LINE838                           512,
```

```
LINE839                         -2453,
LINE840                         0,
LINE841                         current_pos_4,
LINE842                         0,
LINE843                         0);
LINE844     }
LINE845     else
LINE846     {
LINE847       _err_current_pos = IOb_tell(&io_buffer);
LINE848       v12 = (__int16 *)AF_err_record_set(
LINE849                         (LPCHAR)"..\\..\\..\\..\\Common\\Formats\\pctwread.c",
LINE850                         495,
LINE851                         -2060,
LINE852                         0,
LINE853                         _err_current_pos,
LINE854                         0,
LINE855                         0);
LINE856     }
LINE857   }
LINE858   else
LINE859   {
LINE860     _current_pos = IOb_tell(&io_buffer);
LINE861     v12 = (__int16 *)AF_err_record_set(
LINE862                       (LPCHAR)"..\\..\\..\\..\\Common\\Formats\\pctwread.c",
LINE863                       493,
LINE864                       -2100,
LINE865                       0,
LINE866                       _current_pos,
LINE867                       0,
LINE868                       0);
LINE869   }
LINE870   error_iobinit = v12;
LINE871   _error_status = v12;
LINE872 pict_v1:
LINE873   // /------------------------------
LINE874   // PICT V1
LINE875   _src = 0;
LINE876   v16 = 0;
LINE877   Src = 0;
LINE878   index_1 = 0;
LINE879   if ( !error_iobinit )
LINE880   {
LINE881     _offset_in_bloc = 0;
LINE882     offset_in_bloc = 0;
LINE883     while ( parse_opcode_and_seek(&io_buffer, pict_header) )
LINE884     {
LINE885       v18 = Src;
LINE886       // ----------------------------------
LINE887       // Allocate some Memory
LINE888       // If null is passed as size then some default bloc size is used
LINE889       mem_bloc = (char *)AF_memm_realloc(
LINE890                           kind_of_heap,
LINE891                           Src,
LINE892                           (size_t)&_offset_in_bloc[2].buffer4,
LINE893                           (int)"..\\..\\..\\..\\Common\\Formats\\pctwread.c",
LINE894                           523);
LINE895       if ( mem_bloc )
LINE896       {
LINE897         pic_raster_data = (pic_raster_data *)((char *)_offset_in_bloc + (_DWORD)mem_bloc);
LINE898         Src = mem_bloc;
LINE899         error_boolean = get_data_from_opcode(&io_buffer, pict_header->opcode, pic_raster_data);
LINE900         error_boolean_1 = move_in_the_file(&io_buffer, pic_raster_data);
LINE901         _offset_in_bloc = offset_in_bloc;
LINE902         error_iobinit = (__int16 *)(error_boolean_1 + error_boolean);
LINE903       }
LINE904       else
LINE905       {
LINE906         AF_memm_free((void *)kind_of_heap, v18, (int)"..\\..\\..\\..\\Common\\Formats\\pctwread.c", 526);
LINE907         error_iobinit = (__int16 *)AF_err_record_set(
LINE908                                     (LPCHAR)"..\\..\\..\\..\\Common\\Formats\\pctwread.c",
LINE909                                     527,
LINE910                                     0xFFFFF65F,
LINE911                                     0,
LINE912                                     0,
LINE913                                     0,
LINE914                                     0);
LINE915       }
LINE916       v16 = (char *)index_1 + 1;
LINE917       _offset_in_bloc = (mys_table_function *)((char *)_offset_in_bloc + 0x440);
LINE918       index_1 = (char *)index_1 + 1;
LINE919       offset_in_bloc = _offset_in_bloc;
LINE920       if ( error_iobinit )
LINE921         goto LABEL_35;
LINE922     }
LINE923     v16 = (char *)index_1;
LINE924 LABEL_35:
LINE925     _src = (int *)Src;
LINE926     _error_status = error_iobinit;
LINE927   }
LINE928   if ( v16 == (char *)1 )
LINE929   {
LINE930     *(int *)((char *)_src + 42) = *(int *)((char *)_src + 34);
LINE931     *(int *)((char *)_src + 46) = *(int *)((char *)_src + 38);
LINE932   }
LINE933   v23 = 0;
LINE934   mys_table_function_objb = 0;
LINE935   if ( !error_iobinit )
LINE936   {
LINE937     num_planes = 0;
LINE938     start_header = 0;
LINE939     if ( (int)v16 > 0 )
LINE940     {
LINE941       v24 = v16;
LINE942       v25 = 0;
LINE943       do
LINE944       {
LINE945         v26 = *_src;
LINE946         v27 = *_src == 1;
LINE947         _src += 0x110;
LINE948         start_header += v27;
LINE949         num_planes = (__int16 *)((char *)num_planes + (v26 == 2));
LINE950         v23 += v26 == 4;
LINE951         v25 = (__int16 *)((char *)v25 + (v26 == 8));
LINE952         --v24;
LINE953       }
LINE954       while ( v24 );
LINE955       error_iobinit = _error_status;
LINE956       v82 = v25;
```

```
LINE957        _pict_header = pict_header;
LINE958        if ( v23 )
LINE959        {
LINE960          v23 = 4;
LINE961          mys_table_function_objb = 4;
LINE962  LABEL_51:
LINE963          v16 = (char *)index_1;
LINE964          goto LABEL_52;
LINE965        }
LINE966        if ( v82 )
LINE967        {
LINE968          v23 = 8;
LINE969          mys_table_function_objb = 8;
LINE970          goto LABEL_51;
LINE971        }
LINE972        if ( num_planes )
LINE973        {
LINE974          v23 = 2;
LINE975          mys_table_function_objb = 2;
LINE976          goto LABEL_51;
LINE977        }
LINE978        if ( start_header )
LINE979        {
LINE980          v23 = 1;
LINE981          mys_table_function_objb = 1;
LINE982          goto LABEL_51;
LINE983        }
LINE984      }
LINE985      error_iobinit = (__int16 *)AF_err_record_set(
LINE986                                 (LPCHAR)"..\\..\\..\\..\\Common\\Formats\\pctwread.c",
LINE987                                 565,
LINE988                                 -2394,
LINE989                                 0,
LINE990                                 0,
LINE991                                 0,
LINE992                                 0);
LINE993      v23 = 0;
LINE994      goto LABEL_51;
LINE995    }
LINE996  LABEL_52:
LINE997    _pict_header->table_of_pic_raster_data = 0;
LINE998    _pict_header->num_of_pic_raster_data = 0;
LINE999    if ( !error_iobinit )
LINE1000   {
LINE1001     _bytes = (char *)Src;
LINE1002     loop_index = (int)(v16 - 1);
LINE1003     v30 = loop_index;
LINE1004     _temp_loop_index = loop_index;
LINE1005     if ( loop_index >= 0 )
LINE1006     {
LINE1007       v31 = (char *)Src + 0x440 * loop_index;
LINE1008       do
LINE1009       {
LINE1010         if ( v23 == *v31 )
LINE1011           break;
LINE1012         v31 -= 272;
LINE1013         --v30;
LINE1014       }
LINE1015       while ( v30 >= 0 );
LINE1016     }
LINE1017     v32 = 0x440 * v30;
LINE1018     v33 = *(unsigned __int16 *)((char *)Src + v32 + 0x1C);
LINE1019     v34 = (pic_raster_data *)((char *)Src + v32);
LINE1020     start_header = v33;
LINE1021     num_planes = (__int16 *)v34->num_planes;
LINE1022     _pict_header->pict_frame_72_dpi.top_left_corner_x = v34->four_short_1.top_left_corner_x;
LINE1023     _pict_header->pict_frame_72_dpi.lower_right_corner_x = v34->four_short_1.lower_right_corner_x;
LINE1024     _pict_header->pict_frame_72_dpi.top_left_corner_Y = v34->four_short_1.top_left_corner_Y;
LINE1025     v35 = loop_index;
LINE1026     _pict_header->pict_frame_72_dpi.lower_right_corner_y = v34->four_short_1.lower_right_corner_y;
LINE1027     if ( loop_index >= 0 )
LINE1028     {
LINE1029       v36 = start_header;
LINE1030       v37 = mys_table_function_objb;
LINE1031       v38 = (__int16 *)&_bytes[1088 * v35 + 42];
LINE1032       do
LINE1033       {
LINE1034         if ( *(_DWORD *)(v38 - 21) == v37 && *(v38 - 7) == v36 )
LINE1035         {
LINE1036           if ( *(v38 - 6) == (_WORD)num_planes )
LINE1037           {
LINE1038             ++_pict_header->num_of_pic_raster_data;
LINE1039             top_left_corner_x = *v38;
LINE1040             if ( _pict_header->pict_frame_72_dpi.top_left_corner_x < *v38 )
LINE1041               top_left_corner_x = _pict_header->pict_frame_72_dpi.top_left_corner_x;
LINE1042             top_left_corner_Y = _pict_header->pict_frame_72_dpi.top_left_corner_Y;
LINE1043             _pict_header->pict_frame_72_dpi.top_left_corner_x = top_left_corner_x;
LINE1044             v41 = v38[1];
LINE1045             if ( top_left_corner_Y < v41 )
LINE1046               v41 = top_left_corner_Y;
LINE1047             lower_right_corner_x = _pict_header->pict_frame_72_dpi.lower_right_corner_x;
LINE1048             _pict_header->pict_frame_72_dpi.top_left_corner_Y = v41;
LINE1049             v43 = v38[2];
LINE1050             if ( lower_right_corner_x > v43 )
LINE1051               v43 = lower_right_corner_x;
LINE1052             lower_right_corner_y = _pict_header->pict_frame_72_dpi.lower_right_corner_y;
LINE1053             _pict_header->pict_frame_72_dpi.lower_right_corner_x = v43;
LINE1054             v45 = v38[3];
LINE1055             if ( lower_right_corner_y > v45 )
LINE1056               v45 = lower_right_corner_y;
LINE1057             _pict_header->pict_frame_72_dpi.lower_right_corner_y = v45;
LINE1058           }
LINE1059           v37 = mys_table_function_objb;
LINE1060         }
LINE1061         v38 -= 544;
LINE1062         --v35;
LINE1063       }
LINE1064       while ( v35 >= 0 );
LINE1065       loop_index = _temp_loop_index;
LINE1066     }
LINE1067     table_pic_data = (pic_raster_data *)AF_memm_alloc(
LINE1068                                           kind_of_heap,
LINE1069                                           0x440 * _pict_header->num_of_pic_raster_data,
LINE1070                                           (int)"..\\..\\..\\..\\Common\\Formats\\pctwread.c",
LINE1071                                           619);
LINE1072     _pict_header->table_of_pic_raster_data = table_pic_data;
LINE1073     if ( table_pic_data )
LINE1074     {
```

```
LINE1075       v47 = _pict_header->num_of_pic_raster_data - 1;
LINE1076       if ( loop_index >= 0 )
LINE1077       {
LINE1078         index = 0x440 * v47;
LINE1079         src_buffer = (pic_raster_data *)((char *)Src + 0x440 * loop_index);
LINE1080         index_1 = (void *)(0x440 * v47);
LINE1081         do
LINE1082         {
LINE1083           if ( v47 < 0 )
LINE1084             break;
LINE1085           if ( src_buffer->field_0x0 == mys_table_function_objb
LINE1086             && src_buffer->pixel_size == (_WORD)start_header
LINE1087             && src_buffer->num_planes == (_WORD)num_planes )
LINE1088           {
LINE1089             qmemcpy((char *)_pict_header->table_of_pic_raster_data + index, src_buffer, sizeof(pic_raster_data));
LINE1090             loop_index = _temp_loop_index;
LINE1091             --v47;
LINE1092             index = (int)index_1 - 0x440;
LINE1093             index_1 = (char *)index_1 - 0x440;
LINE1094           }
LINE1095           --loop_index;
LINE1096           --src_buffer;
LINE1097           _temp_loop_index = loop_index;
LINE1098         }
LINE1099         while ( loop_index >= 0 );
LINE1100       }
LINE1101       if ( mys_table_function_objb == 1 || mys_table_function_objb == 2 )
LINE1102       {
LINE1103         if ( _pict_header->num_of_pic_raster_data > 0 )
LINE1104         {
LINE1105           v61 = 0;
LINE1106           v62 = 1;
LINE1107           pict_headerc = 0;
LINE1108           start_header = 1;
LINE1109           do
LINE1110           {
LINE1111             v63 = v62;
LINE1112             if ( v62 < _pict_header->num_of_pic_raster_data )
LINE1113             {
LINE1114               p_lower_right_corner_y = (int)&v61[15].pict_frame_72_dpi.lower_right_corner_y;
LINE1115               num_planes = &v61[15].pict_frame_72_dpi.lower_right_corner_y;
LINE1116               do
LINE1117               {
LINE1118                 table_of_pic_raster_data = _pict_header->table_of_pic_raster_data;
LINE1119                 v66 = (__int16 *)((char *)&v61->size_of_file + (_DWORD)table_of_pic_raster_data);
LINE1120                 v67 = (char *)table_of_pic_raster_data + p_lower_right_corner_y;
LINE1121                 v82 = (__int16 *)((char *)table_of_pic_raster_data + p_lower_right_corner_y);
LINE1122                 LOWORD(table_of_pic_raster_data) = *(__int16 *)((char *)&table_of_pic_raster_data-
>four_short_1.lower_right_corner_x
LINE1123                                                                   + p_lower_right_corner_y);
LINE1124                 v81 = v66;
LINE1125                 if ( (__int16)table_of_pic_raster_data <= v66[21] )
LINE1126                 {
LINE1127                   qmemcpy(v79, v67, sizeof(v79));
LINE1128                   qmemcpy(v82, v81, 0x440u);
LINE1129                   qmemcpy((char *)pict_headerc + (unsigned int)_pict_header->table_of_pic_raster_data, v79, 0x440u);
LINE1130                   p_lower_right_corner_y = (int)num_planes;
LINE1131                 }
LINE1132                 v61 = pict_headerc;
LINE1133                 ++v63;
LINE1134                 p_lower_right_corner_y += 0x440;
LINE1135                 num_planes = (__int16 *)p_lower_right_corner_y;
LINE1136               }
LINE1137               while ( v63 < _pict_header->num_of_pic_raster_data );
LINE1138               v62 = start_header;
LINE1139             }
LINE1140             start_header = v62 + 1;
LINE1141             v61 = (pict_header *)((char *)v61 + 0x440);
LINE1142             v68 = v62++ < _pict_header->num_of_pic_raster_data;
LINE1143             pict_headerc = v61;
LINE1144           }
LINE1145           while ( v68 );
LINE1146         }
LINE1147       }
LINE1148       else
LINE1149       {
LINE1150         v50 = (char *)Src;
LINE1151         v51 = 1;
LINE1152         *(_DWORD *)((char *)Src + 42) = *(_DWORD *)((char *)Src + 34);
LINE1153         *(_DWORD *)(v50 + 46) = *(_DWORD *)(v50 + 38);
LINE1154         num_planes = (__int16 *)1;
LINE1155         if ( _pict_header->num_of_pic_raster_data > 1 )
LINE1156         {
LINE1157           v52 = v50 + 1130;
LINE1158           v53 = -42 - (_DWORD)v50;
LINE1159           index_1 = v50 + 1130;
LINE1160           v81 = (__int16 *)(-42 - (_DWORD)v50);
LINE1161           do
LINE1162           {
LINE1163             *v52 = *(v52 - 272);
LINE1164             v52[1] = *(v52 - 271);
LINE1165             v54 = *((_WORD *)v52 - 2);
LINE1166             *(_WORD *)v52 += v54;
LINE1167             *((_WORD *)v52 + 2) += v54;
LINE1168             v55 = v51;
LINE1169             if ( v51 < _pict_header->num_of_pic_raster_data )
LINE1170             {
LINE1171               v56 = (int)v52 + v53;
LINE1172               start_header = v56;
LINE1173               v57 = (pict_header *)v56;
LINE1174               pict_headerb = (pict_header *)v56;
LINE1175               do
LINE1176               {
LINE1177                 v58 = _pict_header->table_of_pic_raster_data;
LINE1178                 v59 = (__int16 *)((char *)v58 + v56);
LINE1179                 v60 = (char *)v58 + (_DWORD)v57;
LINE1180                 _error_status = (__int16 *)((char *)v58 + (_DWORD)v57);
LINE1181                 LOWORD(v58) = *(__int16 *)((char *)&v58->four_short_1.top_left_corner_x + (_DWORD)v57);
LINE1182                 v82 = v59;
LINE1183                 if ( (__int16)v58 < v59[21] )
LINE1184                 {
LINE1185                   qmemcpy(v79, v60, sizeof(v79));
LINE1186                   qmemcpy(_error_status, v82, 0x440u);
LINE1187                   qmemcpy((char *)_pict_header->table_of_pic_raster_data + start_header, v79, sizeof(pic_raster_data));
LINE1188                   v57 = pict_headerb;
LINE1189                 }
LINE1190                 v56 = start_header;
LINE1191                 ++v55;
```

```
LINE1192                v57 = (pict_header *)((char *)v57 + 1088);
LINE1193                pict_headerb = v57;
LINE1194              }
LINE1195              while ( v55 < _pict_header->num_of_pic_raster_data );
LINE1196              v52 = index_1;
LINE1197              v51 = (int)num_planes;
LINE1198              v53 = (int)v81;
LINE1199            }
LINE1200            ++v51;
LINE1201            v52 += 272;
LINE1202            num_planes = (__int16 *)v51;
LINE1203            index_1 = v52;
LINE1204          }
LINE1205          while ( v51 < _pict_header->num_of_pic_raster_data );
LINE1206        }
LINE1207      }
LINE1208      num_of_pic_raster_data = 0;
LINE1209      if ( _pict_header->num_of_pic_raster_data > 0 )
LINE1210      {
LINE1211        v70 = 0;
LINE1212        do
LINE1213        {
LINE1214          ++num_of_pic_raster_data;
LINE1215          _pict_header->table_of_pic_raster_data[v70++].four_short_1.top_left_corner_x -= _pict_header-
>pict_frame_72_dpi.top_left_corner_x;
LINE1216          _pict_header->table_of_pic_raster_data[v70 - 1].four_short_1.lower_right_corner_x -= _pict_header-
>pict_frame_72_dpi.top_left_corner_x;
LINE1217          _pict_header->table_of_pic_raster_data[v70 - 1].four_short_1.top_left_corner_Y -= _pict_header-
>pict_frame_72_dpi.top_left_corner_Y;
LINE1218          _pict_header->table_of_pic_raster_data[v70 - 1].four_short_1.lower_right_corner_y -= _pict_header-
>pict_frame_72_dpi.top_left_corner_Y;
LINE1219        }
LINE1220        while ( num_of_pic_raster_data < _pict_header->num_of_pic_raster_data );
LINE1221      }
LINE1222      goto LABEL_109;
LINE1223    }
LINE1224    if ( !AF_err_record_set((LPCHAR)"..\\..\\..\\..\\Common\\Formats\\pctwread.c", 621, -3020, 0, 0, 0, 0) )
LINE1225    {
LINE1226 LABEL_109:
LINE1227      v71 = _pict_header->table_of_pic_raster_data;
LINE1228      pixel_size = v71->pixel_size;
LINE1229      if ( pixel_size == 1 || pixel_size == 4 || pixel_size == 8 )
LINE1230        OS_memcpy(dest_bytes, &v71->dib_palette, 4 * (1 << pixel_size));
LINE1231      if ( mys_table_function_objb == 4 )
LINE1232      {
LINE1233        *(_DWORD *)a5 = 6;
LINE1234      }
LINE1235      else
LINE1236      {
LINE1237        v73 = _pict_header->table_of_pic_raster_data;
LINE1238        if ( v73->next_offset < 8u
LINE1239          || (packet_type = (unsigned __int16)v73->packet_type, packet_type == 1)
LINE1240          || packet_type == 2 && v73->pixel_size >= 0x18u )
LINE1241        {
LINE1242          *(_DWORD *)a5 = 0;
LINE1243        }
LINE1244        else
LINE1245        {
LINE1246          *(_DWORD *)a5 = 7;
LINE1247        }
LINE1248      }
LINE1249    }
LINE1250  }
LINE1251  if ( Src )
LINE1252    AF_memm_free((void *)kind_of_heap, Src, (int)"..\\..\\..\\..\\Common\\Formats\\pctwread.c", 703);
LINE1253  IOb_done(&io_buffer);
LINE1254  return AF_error_check();
LINE1255 }
```

Effectively, values can be subtracted from the original frame in 72 dpi contained in the header of the PICT file at LINE1218. And this is all controlled by the number `num_of_pic_raster_data` in the do-while loop from LINE1212 to LINE1220.

The original frame 72 dpi values are read at LINE748 into `pict_header->pict_frame_72_dpi` through the call to the `pict_read_4_short` function.

So we see that we can have control over the size of the destination buffer that we want to overwrite in multiple ways. The main issue is that the size of the allocated buffer is not considered inside the `write_into_dest_buffer` function, which eventually writes out-of-bounds.

This makes the code run under wrong size assumptions, and the do-while loop will run past the end of the buffer, triggering the out-of-bounds write condition in the heap, which can lead to arbitrary code execution.

Crash Information

```
0:000> !analyze -v
*******************************************************************************
*                                                                             *
*                        Exception Analysis                                   *
*                                                                             *
*******************************************************************************


KEY_VALUES_STRING: 1

    Key  : AV.Fault
    Value: Write

    Key  : Analysis.CPU.mSec
    Value: 3765

    Key  : Analysis.DebugAnalysisManager
    Value: Create

    Key  : Analysis.Elapsed.mSec
    Value: 11741

    Key  : Analysis.Init.CPU.mSec
    Value: 8937

    Key  : Analysis.Init.Elapsed.mSec
    Value: 64720149

    Key  : Analysis.Memory.CommitPeak.Mb
    Value: 162

    Key  : Timeline.OS.Boot.DeltaSec
    Value: 451097

    Key  : Timeline.Process.Start.DeltaSec
    Value: 64719

    Key  : WER.OS.Branch
    Value: vb_release

    Key  : WER.OS.Timestamp
    Value: 2019-12-06T14:06:00Z

    Key  : WER.OS.Version
    Value: 10.0.19041.1

    Key  : WER.Process.Version
    Value: 1.0.1.1


NTGLOBALFLAG:  2000000

APPLICATION_VERIFIER_FLAGS:  0

APPLICATION_VERIFIER_LOADED: 1

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 6731dd22 (MSVCR110!memcpy+0x0000002a)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 00000001
   Parameter[1]: 0cd21000
Attempt to write to address 0cd21000

FAULTING_THREAD:  00001394

PROCESS_NAME:  Fuzzme.exe

WRITE_ADDRESS:  0cd21000

ERROR_CODE: (NTSTATUS) 0xc0000005 - The instruction at 0x%p referenced memory at 0x%p. The memory could not be %s.

EXCEPTION_CODE_STR:  c0000005

EXCEPTION_PARAMETER1:  00000001

EXCEPTION_PARAMETER2:  0cd21000

STACK_TEXT:
0019f638 6748f9a6     0cd21000 0c4fb012 00000001 MSVCR110!memcpy+0x2a
WARNING: Stack unwind information not available. Following frames may be wrong.
0019f64c 675c9b43     0cd21000 0c4fb012 00000001 igCore19d+0xf9a6
0019f66c 675c856f     0cd20ff8 0cd228f8 0c4fb103 igCore19d!IG_mpi_page_set+0xddb13
0019f734 675c7751     0019fc3c 1000001e 0e44eff8 igCore19d!IG_mpi_page_set+0xdc53f
0019fbb4 674c13d9     0019fc3c 0e44eff8 00000001 igCore19d!IG_mpi_page_set+0xdb721
0019fbec 675008d7     00000000 0e44eff8 0019fc3c igCore19d!IG_image_savelist_get+0xb29
0019fe68 67500239     00000000 0524dfd0 00000001 igCore19d!IG_mpi_page_set+0x148a7
0019fe88 67495757     00000000 0524dfd0 00000001 igCore19d!IG_mpi_page_set+0x14209
0019fea8 00402219     0524dfd0 0019febc 00000001 igCore19d!IG_load_file+0x47
0019fec0 00402524     0524dfd0 0019fef8 051aff48 Fuzzme!fuzzme+0x19
0019ff28 0040668d     00000005 051a8f68 051aff48 Fuzzme!fuzzme+0x324
0019ff70 760ffa29     003fe000 760ffa10 0019ffdc Fuzzme!fuzzme+0x448d
0019ff80 774b7a9e     003fe000 721cdcd8 00000000 KERNEL32!BaseThreadInitThunk+0x19
0019ffdc 774b7a6e     ffffffff 774d8a44 00000000 ntdll!__RtlUserThreadStart+0x2f
0019ffec 00000000     00406715 003fe000 00000000 ntdll!_RtlUserThreadStart+0x1b


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  MSVCR110!memcpy+2a

MODULE_NAME: MSVCR110

IMAGE_NAME:  MSVCR110.dll

FAILURE_BUCKET_ID:  INVALID_POINTER_WRITE_STRING_DEREFERENCE_AVRF_c0000005_MSVCR110.dll!memcpy

OS_VERSION:  10.0.19041.1

BUILDLAB_STR:  vb_release

OSPLATFORM_TYPE:  x86
```

```
OSNAME:  Windows 10

IMAGE_VERSION:  11.0.50727.1

FAILURE_ID_HASH:  {77975e19-9d4d-daf1-6c0e-6a3a4c334a80}

Followup:     MachineOwner
---------
```

Timeline

2021-11-16 - Vendor disclosure
2021-11-17 - Vendor acknowledged and created case number
2021-12-01 - Vendor advised Q1 2022 plans for fix
2021-12-07 - 30 day disclosure extension granted
2022-01-06 - Follow up w/ vendor re: disclosure release
2022-03-31 - Public Release

CREDIT

Discovered by Emmanuel Tacheau of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2021-1377

NEXT REPORT TALOS-2022-1512

© 2022 Cisco Systems, Inc. and/or its affiliates. All rights reserved. View our Privacy Policy.