Talos Vulnerability Report

TALOS-2021-1377

Accusoft ImageGear JPEG-JFIF Scan header parser out-of-bounds write vulnerability

FEBRUARY 23, 2022

CVE NUMBER

CVE-2021-21949

Summary

An improper array index validation vulnerability exists in the JPEG-JFIF Scan header parser functionality of Accusoft ImageGear 19.10. A specially-crafted file can lead to an out-of-bounds write and potential code exectuion. 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

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

CWE

CWE-129 - Improper Validation of Array Index

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.

A specially-crafted JPEG file can lead to a stack-based buffer overflow in the JPEG-JFIF progressive image parser, due to a improper array index validation vulnerability, which leads to a type confusion, combined with a numeric range comparison without minimum check.

Trying to load a malicious JPEG file, we end up in the following situation:

```
(2894.2098): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=ffffac2c ebx=80000000 ecx=6f57e46c edx=0000ff20 esi=ffffed84 edi=0000f896
eip=6f4451e9 esp=0019f924 ebp=0019fa8c iopl=0         nv up ei ng nz na po cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010283
igCore19d!IG_mpi_page_set+0xb91b9:
6f4451e9 668994457cffffff mov     word ptr [ebp+eax*2-84h],dx ss:002b:00195260=????
```

The access violation take place at `[4]` in the `process_jpeg_progressive` function:

```
void process_jpeg_progressive
              (jpeg_dec *jpeg_dec,SOF_object *SOF,short restart_interval,int max_X_sampling,
              int max_Y_sampling)

{

[...]
LOOP_COMPONENT_START:
  if (0 < (int)comp_idx) {
    comp_idx_ = 0;
    cur_SOF = SOF;
    do {
      temp_var__ = comp_idx_ * 0x50;
      component_obj = *cur_SOF->nr_component_buffer_data + comp_idx_;
      y_idx = 0;
      compo_size_times_comp_idx = temp_var__;
      if (0 < *(int *)(&component_obj->field_0x0 + 0xe)) {
        do {
                    /* subsampling_Y ^  */
          x_idx = 0;
          if (0 < (int)(component_obj->component_values).subsampling_X) {
            local_EDI_2224 = component_array[comp_idx_];
            local_fc = local_EDI_2224;
            do {
              OS_memcpy(stack_image_row_temp,
                        (void *)(((local_EDI_2224->Y_times_height_idx_div_Y_MAX + y_idx) *
                                  local_EDI_2224->related_standardized_width + x_idx +
                                local_EDI_2224->probably_counter) * 0x80 +
                                local_EDI_2224->probably_data_ptr),0x80);
              SOS_Ss__ = (short)SOS_Ss;
              if ((ushort)SOS_Ah == 0) {
                [...]

                SOS_Ss_related = (short)SOS_Ss_plus_done;
                zig_zag_plus_SOS_Ss = (word *)(JPEG_ZIGZAG_MAP + SOS_Ss_related);
                if (flag_to_enter_the_crash_branch == 0) {
                  huffman_AC_table_obj = *(huffman_table_struct **)
                            ((int)&(*SOF->nr_component_buffer_data)[0].component_values.
                              huffman_AC_Table + compo_size_times_comp_idx);                          [1]
                  PRE_huffman_code_idx_buff = &huffman_AC_table_obj->PRE_huffman_code_idx_buff;
                  PRE_huffman_code_length_buff = &huffman_AC_table_obj->PRE_huffman_code_length_buff;
                  PRE_huffman_code_next_elem_length = (byte *)&huffman_AC_table_obj->PRE_huffman_code_next_elem_length;
                  raw_AC_table_values = (ushort *)huffman_AC_table_obj->raw_values;
                  parsed_huffman_code = huffman_AC_table_obj->parsed_huffman_code;

                  while ((local_EDI_2224 = local_fc, SOS_Ss_related < 0x40 &&
                         ((short)SOS_Ss_plus_done <= (short)SOS_Se))) {
                    if (((int)bit_read < 0x10) &&
                      (temp_var_ = (byte *)::read_n_bytes((io_buffer *)&io_buff_,8,&read_n_bytes),
                      temp_var_ != (byte *)0x0)) {
                      [... read the scan data and update a variable called dword_parsed_sum ...]
                    }
                    huffman_code = (ushort)parsed_huffman_code[dword_parsed_sum >> 0x15];

                    [... update variables and based on the huffman table the dword_parsed_sum and temp_var__ values ...]

                    if ((dword_parsed_sum & 0xffffff00) == 0) {
                      huffman_code = raw_AC_table_values[dword_parsed_sum];
                    }
                    else {
                      huffman_code = raw_AC_table_values[PRE_huffman_code_length_buff[temp_var__]];
                    }
                    offset_source_shifted = (short)huffman_code >> 4;                                 [2]
                    _offset_buffer_idx = (uint)offset_source_shifted;
                    local_a8 = huffman_code & 0xf;
                    local_a4 = comp_idx;

                    [... calculate calcualted_value and other values ...]

                    if (flag_to_enter_the_crash_branch != 0) goto SKIP_WRITE;
                    zig_zag_accessing_offset = (short)((int)SOS_Ss_plus_done + _offset_buffer_idx);
                    if ((zig_zag_accessing_offset <= (short)SOS_Se) &&
                                  (zig_zag_accessing_offset < 0x40)) {                               [3]
                      stack_image_row_temp[(short)zig_zag_plus_SOS_Ss[(short)_offset_buffer_idx]] =
                          (ushort)calcualted_value << ((byte)SOS_Al & 0x1f);                          [4]
                    }
                    SOS_Ss_plus_done = (byte *)((int)SOS_Ss_plus_done + _offset_buffer_idx + 1);
                    zig_zag_plus_SOS_Ss = zig_zag_plus_SOS_Ss + (short)_offset_buffer_idx + 1;
                    SOS_Ss_related = (short)SOS_Ss_plus_done;
                    dword_parsed_sum = local_a4;
                  }
                }
                else {
                  [...]
                }
              }
              else {
                [...]
              }
              OS_memcpy((void *)(((local_EDI_2224->Y_times_height_idx_div_Y_MAX + y_idx) *
                                  local_EDI_2224->related_standardized_width + x_idx +
                                local_EDI_2224->probably_counter) * 0x80 +
                                local_EDI_2224->probably_data_ptr),stack_image_row_temp,0x80);
              x_idx = x_idx + 1;
              dword_parsed_sum = local_a4;
            } while (x_idx < *(int *)((int)&(*SOF->nr_component_buffer_data)[0].
                                          component_values.subsampling_X +
                                    compo_size_times_comp_idx));
            bit_after_SOS_read_2 = bit_read;
            cur_SOF = SOF;
            temp_var__ = compo_size_times_comp_idx;
          }
          [...]
        } while (y_idx < (int)(component_obj->component_values).subsampling_Y);
      }
      [...]
}
```

This access violation is originated in the `parse_SOS_SOF` function:

```
AT_ERRCOUNT
parse_SOS_SOF(jpeg_dec *jpeg_dec,SOS_From_FILE *SOS_From_FILE,SOF_object *param_3,int param_4,
            SOS_object *output,jpeg_component_table_SOS **param_6,SOS_From_FILE **param_7,
            dword *param_8)

{
  [...]

  SOS_data = SOS_From_FILE->data_marker;

  [...]

  nr_comp = (uint)SOS_data->nr_comp;
  output->nr_comp = nr_comp;

  [...]

  jpeg_component_table = (jpeg_component_table_SOS *)AF_memm_alloc(uVar1,nr_comp * 0x50);
  if (jpeg_component_table == (jpeg_component_table_SOS *)0x0) {
    AVar4 = AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\jpeg_dec.c",0x536,-1000,0,0,0,
                              (LPCHAR)0x0);
    return AVar4;
  }
  OS_memset(jpeg_component_table,0,output->nr_comp * 0x50);
  compnents_data = (SOS_parsed_comp *)AF_memm_alloc(uVar1,output->nr_comp * 0xc);
  output->parsed_comp = compnents_data;
  if (compnents_data == (SOS_parsed_comp *)0x0) {
    local_1c = AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\jpeg_dec.c",0x53d,-1000,0,0,0,
                                (LPCHAR)0x0);
    pSVar10 = (SOS_From_FILE *)0x0;
  }
  else {
    OS_memset(compnents_data,0,output->nr_comp * 0xc);
    if (0 < (int)output->nr_comp) {
      parsed_comp_idx = 0;
      iVar11 = 0;
      current_SOS_data = (SOS_entry *)SOS_data;
      do {
        SOS_entry_shifted = &current_SOS_data->SOS_ENTRY+1;
        iVar11 = iVar11 + 1;
        *(uint *)(&output->parsed_comp->component_id + parsed_comp_idx) = (uint)current_SOS_data->component_id;
        *(uint *)(&output->parsed_comp->DC_table_idx + parsed_comp_idx) =
                                (uint)(SOS_entry_shifted->DC|AC >> 4);                               [5]
        *(uint *)(&output->parsed_comp->AC_table_idx + parsed_comp_idx) =
                                       SOS_entry_shifted->DC|AC & 0xf;                               [6]

        parsed_comp_idx = parsed_comp_idx + 0xc;
        current_SOS_data = (SOS_entry *)SOS_entry_shifted;
      } while (iVar11 < (int)output->nr_comp);
    }
    iVar8 = 0;
    SOS_comp_num = output->nr_comp;
    SOF_comp_num = (param_3->SOF_header).size_ImageTableComponent;
    [...]
    comp_idx = 0;
    if (0 < (int)SOS_comp_num) {
      jpeg_component = &jpeg_component_table->component_values;
      SOS_comp_idx = 0;
      do {
        [...]
        jpeg_component[-1].huffman_DC_Table =
            (dword)jpeg_dec->HuffmanDC_TableSymbols_
                    [*(int *)(&output->parsed_comp->DC_table_idx + SOS_comp_idx)];                   [7]
        jpeg_component->huffman_AC_Table =
            (dword)jpeg_dec->HuffmanAC_TableSymbols
                    [*(int *)(&output->parsed_comp->AC_table_idx + SOS_comp_idx)];                   [8]
        jpeg_component->subsampling_X =
            (*(param_3->SOF_header).ImageTableComponent)[dVar5].horizontalSamplingFactor;
        jpeg_component->subsampling_Y =
            (*(param_3->SOF_header).ImageTableComponent)[dVar5].verticalSamplingFactor;
        [...]
        SOS_comp_idx = SOS_comp_idx + 0xc;
        jpeg_component = jpeg_component + 4;
        comp_idx = comp_idx + 1;
      } while (comp_idx < (int)output->nr_comp);
    }
    [...]
}
```

This function, among other things, is responsible for associating the image components with the correct `AC` and `DC` parsed huffman tables, called from now on, respectively, `AC_t` and `DC_t`. The function that parses the huffman tables, from this point called `parse_huffman`, parses the `AC_t` and `DC_t` checking that the several specification constraints are respected. For instance, in the `parse_huffman` function is ensured that, per each table type, the identifier can only range from `0` to `3`. After the tables are parsed, if any, their pointer are placed sequentially into a structure. Following a schematization of the structure's memory layout:

```
0x00 XXXX  YYYY  XXXX  YYYY
...
0x28 DC_t0 DC_t1 DC_t2 DC_t3
0x38 AC_t0 AC_t1 AC_t2 AC_t3
...
```

The number after `DC_t` and `AC_t` is the table's specified identifier. This struct, that also contains data and pointers not related to the huffman tables, is used at `[7]` and `[8]` to associate the parsed component element with the correct parsed `DC_t` and `AC_t`. This association is perfomed using the parsed `SOS`'s `DC` and `AC` component values, taken respectively at `[5]` and `[6]`. These values specify the element indexes of the huffman tables to be used. The problem is that these variables are four bits long, which means they can range from `0` to `15`. These values are used as an array index to get the specified element, but because no check is perfomed on the value of `AC` or `DC` it is possible to select elements that are not parsed huffman tables. This will break the assumption ensured by the various checks in `parse_huffman`.

The instruction used at `[8]` is `mov eax,dword ptr [esi+eax*4+38h]`:

```
0:000> dd esi+0x38
0a300f98  00000000 00000000 00000000 00000000
0a300fa8  0a960720 6f453880 00000000 00000000
0a300fb8  00000000 00000000 00000002 00000000
0a300fc8  00000000 00000f0f 00000000 00000000
0a300fd8  00000002 0ae30fd0 00000001 00000000
0a300fe8  00000000 00000000 00000000 00000015
0a300ff8  00000000 d0d0d0d0 ???????? ????????
0a301008  ???????? ???????? ???????? ????????
```

At `esi+0x38` the first of the four possible parsed `AC_t` is located. No `AC_t` were specified in this example. Instead, `eax` contains the specified `SOS`'s `AC` value:

```
0:000> r eax
eax=00000005
```

The `eax` value is used, starting from `esi+0x38`, as an element index. So, the fifth element is taken. In this case the element is `0x6f453880`, a function pointer.

At `[1]` the associated huffman pointer is loaded and used to calculate, among the other thing, the index value at `[2]` used for accesing the `JPEG_ZIGZAG_MAP` buffer, offset by the `SOS`'s `Ss` value. The `JPEG_ZIGZAG_MAP` is a buffer of short with 64 elements, and the biggest value is `0x3F`. The `JPEG_ZIGZAG_MAP` buffer is used to get the correct index to access `stack_image_row_temp`, a stack buffer with 64 short elements. Because the biggest element in `JPEG_ZIGZAG_MAP` is `0x3F`(decimal `63`) this would ensure that the `stack_image_row_temp` buffer is accessed at most to its last element. But, because the huffman table assumptions are broken it is possible for `_offset_buffer_idx`, the pointer that accesses the `JPEG_ZIGZAG_MAP`, to have a negative value. The negative value would then bypass the checks perfomed at `[3]` because they are a signed comparison that only check the maximum range values. This allows us to obtain values outside the `JPEG_ZIGZAG_MAP` buffer range, and consequentially write out of the `stack_image_row_temp` bounds.

Here are the relevant assembly instructions related to the check at `[3]` and the point at which the crash happens at `[4]`:

```
mov     edx, dword ptr [ebp-0B0h]                       ; _offset_buffer_idx
[...]
mov     ecx, dword ptr [ebp-0A8h]                       ; JPEG_ZIGZAG_MAP + Ss
mov     edi, dword ptr [ebp-9Ch]                        ; SOS_Ss_plus_done,  it should be equal to Ss
movsx   eax, dx                                         ; off_idx_16_sign_ext = (take 16 sign extended bits) _offset_buffer_idx
add     edi, edx                                        ; calculated_offset = _offset_buffer_idx + SOS_Ss_plus_done
lea     ecx, [ecx+eax*2]                                ; zig_zag_value_ptr =
                                                        ;       JPEG_ZIGZAG_MAP + Ss + off_idx_16_sign_ext * 2

mov     eax, dword ptr [ebp-11Ch]                       ; SOS_Se
mov     dword ptr [ebp-0A8h], ecx                       ;
cmp     di, ax                                          ; (calculated_offset & 0xffff) < SOS_Se
jg      SKIP_WRITE                                      ;   if false go to SKIP_WRITE
cmp     di, 40h                                         ; (calculated_offset & 0xffff) <= 0x40,   otherwise SKIP_WRITE
jge     SKIP_WRITE                                      ;   if false go to SKIP_WRITE
mov     cx, word ptr [ebp-10Ch]                         ; SOS_Al
mov     dx, word ptr [ebp-94h]                          ; calculated_value = value to be written into the buffer
shl     dx, cl                                          ; calculated_value = calculated_value << SOS_Al
mov     ecx, dword ptr [ebp-0A8h]                       ; load zig_zag_value_ptr
movsx   eax, word ptr [ecx]                             ; zig_zag_index_value = (take 16 sign extended bits) dereference zig_zag_value_ptr
mov     word ptr [ebp+eax*2-84h], dx                    ; stack_image_row_temp[zig_zag_index_value*2] = calculated_value
```

In the assembly above, it is possible to see that the perfomed checks are signed, and only the maximum of the allowed ranges are checked. This allows negative indexes to pass the check and reach the `JPEG_ZIGZAG_MAP` access. Furthermore, because the buffer are shorts, the index value is multiplied by two in order to seek the correct element. This would allow a negative index to transform into a positive one, if the index provided is small enough to cause an overflow. A potential attacker would be able to control the index used to access `JPEG_ZIGZAG_MAP`, and thus be able to obtain as index for `stack_image_row_temp` a value outside the range of the buffer iteself, giving the capability to write outside that stack buffer.

Crash Information

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


KEY_VALUES_STRING: 1

    Key  : AV.Fault
    Value: Write

    Key  : Analysis.CPU.mSec
    Value: 2952

    Key  : Analysis.DebugAnalysisManager
    Value: Create

    Key  : Analysis.Elapsed.mSec
    Value: 10202

    Key  : Analysis.Init.CPU.mSec
    Value: 375

    Key  : Analysis.Init.Elapsed.mSec
    Value: 27637

    Key  : Analysis.Memory.CommitPeak.Mb
    Value: 135

    Key  : Timeline.OS.Boot.DeltaSec
    Value: 155289

    Key  : Timeline.Process.Start.DeltaSec
    Value: 27

    Key  : WER.OS.Branch
    Value: rs5_release

    Key  : WER.OS.Timestamp
    Value: 2018-09-14T14:34:00Z

    Key  : WER.OS.Version
    Value: 10.0.17763.1

    Key  : WER.Process.Version
    Value: 1.0.1.1


NTGLOBALFLAG:  2100000

APPLICATION_VERIFIER_FLAGS:  0

APPLICATION_VERIFIER_LOADED: 1

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 6f4451e9 (igCore19d!IG_mpi_page_set+0x000b91b9)
ExceptionCode: c0000005 (Access violation)
ExceptionFlags: 00000000
NumberParameters: 2
Parameter[0]: 00000001
Parameter[1]: 00195260
Attempt to write to address 00195260

FAULTING_THREAD:  00002098

PROCESS_NAME:  Fuzzme.exe

WRITE_ADDRESS:  00195260

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:  00195260

STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
0019fa8c 6f4400a0     0adb6f60 0019fab4 00000000 igCore19d!IG_mpi_page_set+0xb91b9
0019fb14 6f4574d5     00000002 6f453160 0a27c720 igCore19d!IG_mpi_page_set+0xb4070
0019fb30 6f45734a     0a27c720 0adb6f60 0000ffda igCore19d!IG_mpi_page_set+0xcb4a5
0019fb54 6f455161     0a27c720 0adb6f60 0019fb7c igCore19d!IG_mpi_page_set+0xcb31a
0019fb74 6f456b7a     0019ffc2 1000001d 0a921f70 igCore19d!IG_mpi_page_set+0xc9131
0019fbb4 6f3613d9     1000001d 0a921f70 00000001 igCore19d!IG_mpi_page_set+0xcab4a
0019fbec 6f3a08d7     00000000 0a921f70 0019fc3c igCore19d!IG_image_savelist_get+0xb29
0019fe68 6f3a0239     00000000 0019ff10 00000001 igCore19d!IG_mpi_page_set+0x148a7
0019fe88 6f335757     00000000 0019ff10 00000001 igCore19d!IG_mpi_page_set+0x14209
0019fea8 00402219     0019ff10 0019febc 00000001 igCore19d!IG_load_file+0x47
0019fec0 00402524     0019ff10 05267fe0 051cdf50 Fuzzme!fuzzme+0x19
0019ff28 0040668d     00000005 051c6f70 051cdf50 Fuzzme!fuzzme+0x324
0019ff70 76f20419     00265000 76f20400 0019ffdc Fuzzme!fuzzme+0x448d
0019ff80 772f72ed     00265000 369f943e 00000000 KERNEL32!BaseThreadInitThunk+0x19
0019ffdc 772f72bd     ffffffff 773165d3 00000000 ntdll!__RtlUserThreadStart+0x2f
0019ffec 00000000     00406715 00265000 00000000 ntdll!_RtlUserThreadStart+0x1b


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  igCore19d!IG_mpi_page_set+b91b9

MODULE_NAME: igCore19d

IMAGE_NAME:  igCore19d.dll

FAILURE_BUCKET_ID:  INVALID_POINTER_WRITE_AVRF_c0000005_igCore19d.dll!IG_mpi_page_set

OS_VERSION:  10.0.17763.1

BUILDLAB_STR:  rs5_release
```

```
OSPLATFORM_TYPE:  x86

OSNAME:  Windows 10

IMAGE_VERSION:  19.10.0.0

FAILURE_ID_HASH:  {39ff52ad-9054-81fd-3e4d-ef5d82e4b2c1}

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

Timeline

2021-09-22 - Initial contact

2021-09-23 - Vendor acknowledged and and confirmed under review with engineering team

2021-11-30 - 60 day follow up

2021-12-01 - Vendor advised release planned for Q1 2022

2021-12-07 - 30 day disclosure extension granted

2022-01-06 - Final disclosure notification

2022-02-23 - Public disclosure

CREDIT

Discovered by Francesco Benvenuto of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2021-1375

NEXT REPORT TALOS-2021-1413