Talos Vulnerability Report

TALOS-2021-1371

Accusoft ImageGear TIFF YCbCr image parser out-of-bounds write vulnerability

FEBRUARY 23, 2022

CVE NUMBER

CVE-2021-21942

Summary

An out-of-bounds write vulnerability exists in the TIFF YCbCr image parser functionality of Accusoft ImageGear 19.10. A specially-crafted file can lead to remote code execution. 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-122 - Heap-based Buffer Overflow

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 TIFF file can lead to a heap-based buffer overflow in the TIFF YCbCr image parser, due to a missing size check. We will assume the following TIFF's tag values: `PlanarConfiguration == 1` and `PhotometricInterpretation == 6`, meaning the image data is in the YCbCr color space in chunky format.

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

```
(1dec.1f7c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0b2c9000 ebx=0000002a ecx=00000080 edx=0bce8ff8 esi=000000ff edi=0bca2ff0
eip=6f959aff esp=0019f578 ebp=0019f604 iopl=0         nv up ei pl nz ac pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010217
igCore19d!IG_mpi_page_set+0x10dacf:
6f959aff 8808            mov     byte ptr [eax],cl          ds:002b:0b2c9000=??
```

The access violation originates at `[1]` in the `TIFF_YCbCr_to_RGB` function:

```
int TIFF_YCbCr_to_RGB
            (mys_tags_data *mys_tags_data,undefined4 param_2,byte **array_of_horiz_input,
            byte **array_of_horiz_buff,uint param_5)

{
[...]

local_8 = DAT_102bcea8 ^ (uint)&stack0xfffffffc;
array_of_horiz_buff_ = array_of_horiz_buff;
probably_subsampling_factor = 0x10001;
YCBCR_pos_related = 1;
                    /* ID_TIF_YCBCR_COEFFICIENTS */
if ((mys_tags_data->IFD_Record == (IFD_Record *)0x0) ||
    (pTVar1 = kind_of_lookup_tags(mys_tags_data->IFD_Record,0x47,0), pTVar1 == (TIF_Record *)0x0))
{
    LumaRed = 0.299;
    LumaBlue = 0.114;
    Green_value = 0.587;
}
else {
    sVar3 = TIFF_INDEX_TO_TYPE_SIZE[(uint)pTVar1->tag_type * 2] * pTVar1->count;
    if (0x18 < sVar3) {
    sVar3 = 0x18;
    }
    OS_memcpy(&local_20,(void *)pTVar1->ptr_value,sVar3);
    LumaRed = (float)((double)local_20 + (double)(&DAT_1022e910)[-(local_20 >> 0x1f)]) /
              (float)((double)local_1c + (double)(&DAT_1022e910)[-(local_1c >> 0x1f)]);
    Green_value = (float)((double)local_18 + (double)(&DAT_1022e910)[-(local_18 >> 0x1f)]) /
                  (float)((double)local_14 + (double)(&DAT_1022e910)[-(local_14 >> 0x1f)]);
    LumaBlue = (float)((double)local_10 + (double)(&DAT_1022e910)[-(local_10 >> 0x1f)]) /
              (float)((double)local_c + (double)(&DAT_1022e910)[-(local_c >> 0x1f)]);
}
                    /* ID_TIF_YCBCR_SUBSAMPLING */
if ((mys_tags_data->IFD_Record != (IFD_Record *)0x0) &&
    (pTVar1 = kind_of_lookup_tags(mys_tags_data->IFD_Record,0x48,0), pTVar1 != (TIF_Record *)0x0))
{
    sVar3 = TIFF_INDEX_TO_TYPE_SIZE[(uint)pTVar1->tag_type * 2] * pTVar1->count;
    if (4 < sVar3) {
    sVar3 = 4;
    }
    OS_memcpy(&probably_subsampling_factor,(void *)pTVar1->ptr_value,sVar3);
}
                    /* ID_TIF_YCBCR_POS */
if ((mys_tags_data->IFD_Record != (IFD_Record *)0x0) &&
    (pTVar1 = kind_of_lookup_tags(mys_tags_data->IFD_Record,0x49,0), pTVar1 != (TIF_Record *)0x0))
{
    sVar3 = TIFF_INDEX_TO_TYPE_SIZE[(uint)pTVar1->tag_type * 2] * pTVar1->count;
    if (2 < sVar3) {
    sVar3 = 2;
    }
    OS_memcpy(&YCBCR_pos_related,(void *)pTVar1->ptr_value,sVar3);
}
                    /* ID_TIF_REFERENCE_BLACK_WHITE */
if ((mys_tags_data->IFD_Record != (IFD_Record *)0x0) &&
    (pTVar1 = kind_of_lookup_tags(mys_tags_data->IFD_Record,0x4a,0), pTVar1 != (TIF_Record *)0x0))
{
    sVar3 = TIFF_INDEX_TO_TYPE_SIZE[(uint)pTVar1->tag_type * 2] * pTVar1->count;
    if (0x30 < sVar3) {
    sVar3 = 0x30;
    }
    OS_memcpy(local_50,(void *)pTVar1->ptr_value,sVar3);
    kind_of_lookup_tags(mys_tags_data->IFD_Record,0x4a,0);
}
LumaGreen_ = Green_value;
local_70 = (int)(short)probably_subsampling_factor.horiz *
            (int)(short)probably_subsampling_factor.vert;
                    /* Handle Planar Config == 1 */
if (mys_tags_data->ID_TIF_PLANAR_CONFIG == 1) {
    if (0 < (int)param_5) {
    local_68 = param_5;
    do {
        Cb_value = (float)(int)(short)((*array_of_horiz_input)[local_70] - 0x80);
        Cr_value = (float)(int)(short)((*array_of_horiz_input)[local_70 + 1] - 0x80);
        n_of_vert_done = 0;
        dst_ptr = array_of_horiz_buff_;
        if (0 < (short)probably_subsampling_factor.vert) {
        do {
            n_of_horiz_done = 0;
            if (0 < (short)probably_subsampling_factor.horiz) {
            do {
                Y_Value = (float)(uint)**array_of_horiz_input;
                *array_of_horiz_input = *array_of_horiz_input + 1;
                temp_computation =
                    SEXT24((short)(int)(Y_Value + (2.0 - LumaBlue * 2.0) * (float)(int)Cb_value));
                if ((int)temp_computation < 0) {
                probably_B_value = 0;
                }
                else {
                probably_B_value = temp_computation & 0xff;
                if (0xff < (int)temp_computation) {
                    probably_B_value = 0xff;
                }
                }
                temp_computation =
                    SEXT24((short)(int)((2.0 - LumaRed * 2.0) * (float)(int)Cr_value + Y_Value));
                if ((int)temp_computation < 0) {
                probably_R_value = 0;
                }
                else {
                probably_R_value = temp_computation & 0xff;
                if (0xff < (int)temp_computation) {
                    probably_R_value = 0xff;
                }
                }
                Green_value = (float)(int)(short)(int)(((Y_Value -
                                                        (float)probably_B_value * LumaBlue) -
                                                    (float)probably_R_value * LumaRed) /
                                                    LumaGreen_);
                if ((int)Green_value < 0) {
                probably_G_value = probably_G_value & 0xffffff00;
                }
                else {
                probably_G_value = (uint)Green_value & 0xff;
                if (0xff < (int)Green_value) {
                    probably_G_value = 0xff;
                }
                }
```

```
                **dst_ptr = (byte)probably_R_value;
                *dst_ptr = *dst_ptr + 1;
                **dst_ptr = (byte)probably_G_value;                                                    [1]
                *dst_ptr = *dst_ptr + 1;
                **dst_ptr = (byte)probably_B_value;
                *dst_ptr = *dst_ptr + 1;
                n_of_horiz_done = n_of_horiz_done + 1;
            } while (n_of_horiz_done < (short)probably_subsampling_factor.horiz);                      [2]
[...]
}
```

The `TIFF_YCbCr_to_RGB` function converts an image from the YCbCr color space to RGB. Indeed, the crash at `[1]` is related to the G component of the RGB conversion. However, based on the TIFF file, it could have happend in any of the other two components.

The problem resides in the size of the buffers contained in the `dst_ptr` array. This variable has `YCbCrSubsampleVert` buffers that would contain the linear trasformation from YCbCr to RGB. From now on, a single buffer in `dst_ptr` will be called `single_dst_ptr`. The loop that populates `single_dst_ptr`, 3 bytes at a time, is perfomed `YCbCrSubsampleHoriz` times. So the `single_dst_ptr` should have a size greater than `3*YCbCrSubsampleHoriz`. The data used to populate `single_dst_ptr` are contained in `array_of_horiz_input`.

The `TIFF_YCbCr_to_RGB` function is called at `[6]` in the `TIFF_parse` function, where the `single_dst_ptr` buffers and the `array_of_horiz_input` are allocated:

```
void TIFF_parse(mys_table_function *param_1,uint param_2,mys_tags_data *TIFF_tags,undefined4 param_4,
            HIGDIBINFO param_5,subsapling_Y_Cb_Cr *YCbCr_subsamp)

{
io_buffer *io_buff;
byte **src_buff;
byte **lpExtraText;
size_t width_allocation_buff_size;
byte *vert_buff;
int iVar1;
dword dVar2;
dword dVar3;
int horiz;
int iVar4;
io_buffer *piVar5;
int vert_index_;
int vert_index;
int width_index;
int local_28;
byte *multiplier;
byte *local_20;
byte **arr_of_dest_buff;
int local_18;
uint width_buff_size;
dword local_c;
int error_code;

local_20 = (byte *)0x0;
width_buff_size = 0;
local_c = 0;
multiplier = (byte *)0x0;
arr_of_dest_buff = (byte **)0x0;
if (*(ushort *)&TIFF_tags->ID_TIF_SAMPLES_PER_PIXEL == 0) {
    AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\tifread.c",0x1793,-0x80d,0,0,0,(LPCHAR)0x0);
    AF_error_check();
    return;
}
io_buff = (io_buffer *)
            AF_memm_alloc(param_2,(uint)*(ushort *)&TIFF_tags->ID_TIF_SAMPLES_PER_PIXEL * 0x34);
if (io_buff == (io_buffer *)0x0) {
    AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\tifread.c",0x1799,-1000,0,0,0,(LPCHAR)0x0);
    AF_error_check();
    return;
}
src_buff = (byte **)AF_memm_alloc(param_2,(uint)*(ushort *)&TIFF_tags->ID_TIF_SAMPLES_PER_PIXEL << 2);
if (src_buff == (byte **)0x0) {
    horiz = 0x17a1;
    lpExtraText = src_buff;
}
else {
    OS_memset(src_buff,0,(uint)*(ushort *)&TIFF_tags->ID_TIF_SAMPLES_PER_PIXEL << 2);
    lpExtraText = (byte **)AF_memm_alloc(param_2,(uint)*(ushort *)
                                                    &TIFF_tags->ID_TIF_SAMPLES_PER_PIXEL << 2);
    if (lpExtraText != (byte **)0x0) {
    if (TIFF_tags->ID_TIF_PHOTO_INTERP == IG_TIF_PHOTO_YCBCR) {
        horiz = (int)(short)YCbCr_subsamp->horiz;
        multiplier = (byte *)((int)((TIFF_tags->ID_TIF_IMAGE_WIDTH - 1) + horiz) / horiz);
        *lpExtraText = (byte *)(((short)YCbCr_subsamp->vert * horiz + 2) * (int)multiplier);
        width_buff_size = IO_raster_size_get(param_5);
        arr_of_dest_buff = (byte **)AF_memm_alloc(param_2,(int)(short)YCbCr_subsamp->vert << 2);
                    /* the times four is required in order to store a pointer for each vertical
                    component */
        if (arr_of_dest_buff != (byte **)0x0) {
        vert_index = 0;
        if (0 < (short)YCbCr_subsamp->vert) {
            width_allocation_buff_size = width_buff_size + 0x80;                                    [3]
            do {
            vert_buff = (byte *)AF_memm_alloc(param_2,width_allocation_buff_size);
            arr_of_dest_buff[vert_index] = vert_buff;
            if (vert_buff == (byte *)0x0) {
                arr_of_dest_buff = (byte **)0x0;
                error_code = 0x17db;
                goto LAB_10177d49;
            }
            vert_index = vert_index + 1;
            } while (vert_index < (short)YCbCr_subsamp->vert);
        }
        goto LAB_10177d86;
        }
        width_allocation_buff_size = (int)(short)YCbCr_subsamp->vert << 2;
        error_code = 0x17d1;
LAB_10177d49:
        AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\tifread.c",error_code,-1000,0,param_2,
                          width_allocation_buff_size,(LPCHAR)arr_of_dest_buff);
        AF_memm_free(param_2,io_buff);
        AF_memm_free(param_2,src_buff);
    }
    else {
LAB_10177d86:
        if (TIFF_tags->ID_TIF_PLANAR_CONFIG == 1) {
        if (TIFF_tags->ID_TIF_PHOTO_INTERP == IG_TIF_PHOTO_YCBCR) {
            *lpExtraText = (byte *)(((int)(short)YCbCr_subsamp->vert *
                                     (int)(short)YCbCr_subsamp->horiz + 2) * (int)multiplier);      [4]
        }
        else {
            [...]
            }
        }
        dVar3 = IOb_init(param_1,param_2,io_buff,(int)*lpExtraText * 5,1);
        dVar3 = (dword)(0 < (int)dVar3);
        width_index = 0;
        local_c = dVar3;
        for (local_18 = 0;
            (dVar3 == 0 &&
            (local_18 < (int)TIFF_tags->from_ID_TIF_STRIP_OFFSET_or_ID_TIF_TILE_OFFSETS));
            local_18 = local_18 + 1) {
            if ((TIFF_tags->ID_TIF_TILE_OFFSETS != 0) &&
            (local_18 < (int)TIFF_tags->from_ID_TIF_STRIP_OFFSET_or_ID_TIF_TILE_OFFSETS_2)) {
            perform_some_read_or_write_intofile
                        (io_buff,*(int *)(TIFF_tags->ID_TIF_TILE_OFFSETS + local_18 * 4) +
                                TIFF_tags->IFD_Offset,0,0);
            }
            local_28 = 0;
            if (0 < (int)TIFF_tags->ID_TIF_ROWS_PER_STRIP) {
            while (width_index < (int)TIFF_tags->ID_TIF_IMAGE_HEIGHT) {
                vert_buff = (byte *)get_data_from_file(io_buff,(uint)*lpExtraText);                 [5]
                *src_buff = vert_buff;
```

```
                if (vert_buff == (byte *)0x0) {
                AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\tifread.c",0x1813,-0x803,0,
                                  (AT_INT)*lpExtraText,0,"Unexpected end of data");
                local_c = dVar3 + 1;
                dVar3 = local_c;
                break;
                }
                if (TIFF_tags->ID_TIF_PHOTO_INTERP == IG_TIF_PHOTO_YCBCR) {
                TIFF_YCbCr_to_RGB
                           (TIFF_tags,param_4,src_buff,(int *)arr_of_dest_buff,(uint)multiplier);      [6]
                [...]
}
```

This function prepares the required arguments in order to call `TIFF_YCbCr_to_RGB`. At `[3]` the size of a `single_dst_ptr` is calculated, which is essentially `0x80 + IO_raster_size_get(param_5)`, where `IO_raster_size_get` return value, in this specific context, can be simplified as `((((8 * SamplesPerPixel) * ImageWidth) + 0x1f) >> 3) & 0xfffffffc`, where `SamplesPerPixel` and `ImageWidth` are specified through the TIFF's tags. Instead, the `array_of_horiz_input` buffer is allocate at `[5]`, and has as size the results of the computation at `[4]`.

At `[2]` the condition of the loop is such that the number of iteration is equal to the `YCbCrSubsampleHoriz` value. In the loop, for each iteration, 3 bytes of a `single_dst_ptr` are filled. The problem is that the program does not enforce any relationship between `ImageWidth`, the value used to compute the size of a `single_dst_ptr`, and `YCbCrSubsampleHoriz`, the value that specifies how many triplets of bytes are going to be written into a `single_dst_ptr`. Because the program does not enforce any relationship between `ImageWidth` and `YCbCrSubsampleHoriz`, it would be possible to write more bytes than the ones that can be stored in a `single_dst_ptr`, leading to a heap-based buffer overflow. Because the values written in a `single_dst_ptr` are linear transformations of YCbCr data to RGB, an attacker is able to calculate which are, starting from the desired RGB value, the YCbCr values to provide.

The reported problem could have been avoided by enforcing the TIFF's specification. Indeed, this program, in the `YCbCrSubSampling` tag prospective, does not respect the specification. For example, the `YCbCrSubSampling` values and their relathionship with `ImageWidth` and `ImageLength` are not checked. By enforcing the specification, the sizing problem presented, which led to the heap-based buffer overflow, would hardly exist.

We considered only the branches involving `PlanarConfiguration == 1`, but the same bug, allegedly, is also present in the branch with `PlanarConfiguration == 2`.

Crash Information

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


KEY_VALUES_STRING: 1

    Key  : AV.Fault
    Value: Write

    Key  : Analysis.CPU.mSec
    Value: 2468

    Key  : Analysis.DebugAnalysisManager
    Value: Create

    Key  : Analysis.Elapsed.mSec
    Value: 9869

    Key  : Analysis.Init.CPU.mSec
    Value: 718

    Key  : Analysis.Init.Elapsed.mSec
    Value: 101983

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

    Key  : Timeline.OS.Boot.DeltaSec
    Value: 39115

    Key  : Timeline.Process.Start.DeltaSec
    Value: 101

    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:  2000000

APPLICATION_VERIFIER_FLAGS:  0

APPLICATION_VERIFIER_LOADED: 1

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 6f959aff (igCore19d!IG_mpi_page_set+0x0010dacf)
ExceptionCode: c0000005 (Access violation)
ExceptionFlags: 00000000
NumberParameters: 2
Parameter[0]: 00000001
Parameter[1]: 0b2c9000
Attempt to write to address 0b2c9000

FAULTING_THREAD:  00001f7c

PROCESS_NAME:  Fuzzme.exe

WRITE_ADDRESS:  0b2c9000

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:  0b2c9000

STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
0019f604 6f9582b7     0019f6d0 0ae38d68 0bca2ff0 igCore19d!IG_mpi_page_set+0x10dacf
0019f650 6f957afa     0019fc3c 10000021 0bcccff0 igCore19d!IG_mpi_page_set+0x10c287
0019f678 6f95ccad     0019fc3c 10000021 0019f6d0 igCore19d!IG_mpi_page_set+0x10baca
0019f6a0 6f956bab     0019fc3c 10000021 0ae38d68 igCore19d!IG_mpi_page_set+0x110c7d
0019fbb4 6f8213d9     0019fc3c 0ae38d68 00000001 igCore19d!IG_mpi_page_set+0x10ab7b
0019fbec 6f8608d7     00000000 0ae38d68 0019fc3c igCore19d!IG_image_savelist_get+0xb29
0019fe68 6f860239     00000000 05317fb0 00000001 igCore19d!IG_mpi_page_set+0x148a7
0019fe88 6f7f5757     00000000 05317fb0 00000001 igCore19d!IG_mpi_page_set+0x14209
0019fea8 00402219     05317fb0 0019febc 00000001 igCore19d!IG_load_file+0x47
0019fec0 00402524     05317fb0 05319fe0 0527df50 Fuzzme!fuzzme+0x19
0019ff28 0040668d     00000005 05276f30 0527df50 Fuzzme!fuzzme+0x324
0019ff70 765f0419     00291000 765f0400 0019ffdc Fuzzme!fuzzme+0x448d
0019ff80 777a72ed     00291000 96d6a6d3 00000000 KERNEL32!BaseThreadInitThunk+0x19
0019ffdc 777a72bd     ffffffff 777c65cf 00000000 ntdll!__RtlUserThreadStart+0x2f
0019ffec 00000000     00406715 00291000 00000000 ntdll!_RtlUserThreadStart+0x1b


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  igCore19d!IG_mpi_page_set+10dacf

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

2021-09-03 - Initial contact
2021-09-07 - Vendor acknowledged and created support ticket
2021-09-10 - Vendor closed support ticket and confirmed under review with engineering team
2021-11-30 - 60 day follow up
2021-12-02 - 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.