

## Talos Vulnerability Report

TALOS-2020-0987

### Accusoft ImageGear PCX uncompress\_scan\_line buffer copy operation code execution vulnerability

FEBRUARY 10, 2020

#### CVE NUMBER

CVE-2020-6064

#### Summary

An exploitable out-of-bounds write vulnerability exists in the `uncompress_scan_line` function of the `igcore19d.dll` library of Accusoft ImageGear, version 19.5.0. A specially crafted PCX file can cause an out-of-bounds write, resulting in a remote code execution. An attacker needs to provide a malformed file to the victim to trigger the vulnerability.

#### Tested Versions

Accusoft ImageGear 19.5.0

#### Product URLs

<https://www.accusoft.com/products/imagegear/overview/>

#### 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-787: Out-of-bounds Write

#### Details

The ImageGear library is a document imaging developer toolkit providing all kinds of functionality related to image conversion, creation, editing, annotation, etc. It supports more than 100 formats, including many image formats, DICOM, PDF, Microsoft Office and others.

There is a vulnerability in the `uncompress_scan_line` function, due to missing sanity checks. A specially crafted PCX file can lead to an out-of-bounds write, which can result in remote code execution.

Trying to load a malformed PCX file via `IG_load_file` function, we end up in the following situation:

```
eax=00000004 ebx=0cd53ff8 ecx=00000005 edx=00658000 esi=0cd3a8f8 edi=00000701
eip=5690a98c esp=0093ee1c ebp=0093ee94 iopl=0         nv up ei pl nz na pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010207
igCore19d!IG_mpi_page_set+0xdf5fc:
5690a98c 880c43      mov     byte ptr [ebx+eax*2],cl    ds:002b:0cd54000=??
0:000> kb
# ChildEBP RetAddr  Args to Child
WARNING: Stack unwind information not available. Following frames may be wrong.
00 0093ee94 5690b446 0093f414 1000001b 0093eef4 igCore19d!IG_mpi_page_set+0xdf5fc
01 0093eebc 5690a26a 0093f414 1000001b 0b6baff8 igCore19d!IG_mpi_page_set+0xe00b6
02 0093f38c 568007c9 0093f414 0b6baff8 00000001 igCore19d!IG_mpi_page_set+0xdeeda
03 0093f3c4 5683fb97 00000000 0b6baff8 0093f414 igCore19d!IG_image_savelist_get+0xb29
04 0093f640 5683f4f9 00000000 09bf5fa8 00000001 igCore19d!IG_mpi_page_set+0x14807
05 0093f660 567d6007 00000000 09bf5fa8 00000001 igCore19d!IG_mpi_page_set+0x14169
06 0093f680 00d859ac 09bf5fa8 0093f76c 0093f790 igCore19d!IG_load_file+0x47
07 0093f780 00d861a7 09bf5fa8 0093f8b4 00000021 simple_exe_141+0x159ac
08 0093f94c 00d86cbe 00000005 09ba2f50 09a85f38 simple_exe_141+0x161a7
09 0093f960 00d86b27 9c768deb 00d815e1 00d815e1 simple_exe_141+0x16cbe
0a 0093f9bc 00d869bd 0093f9cc 00d86d38 0093f9dc simple_exe_141+0x16b27
0b 0093f9c4 00d86d38 0093f9dc 76cd6359 00655000 simple_exe_141+0x169bd
0c 0093f9cc 76cd6359 00655000 76cd6340 0093fa38 simple_exe_141+0x16d38
0d 0093f9dc 77577b74 00655000 e8b707c2 00000000 KERNEL32!BaseThreadInitThunk+0x19
0e 0093fa38 77577b44 ffffffff 77598f20 00000000 ntdll!_RtlUserThreadStart+0x2f
0f 0093fa48 00000000 00d815e1 00655000 00000000 ntdll!_RtlUserThreadStart+0x1b
```

As we can see, an out-of-bounds operation occurred.

To reach the path to this function, some conditions are required:

- `bits_per_pixel` must be set to 2
- `nplanes` must be 2

The pseudo-code of this vulnerable function looks like this:

```

LINE1 int __stdcall uncompress_scan_line(table_function *ptr_function, int a1, pcx_object *p_pcx_data, int a4, IGDIB0ject *IGDIB0bject)
LINE2 {
LINE3     unsigned int _buffer_size_complete_scan_line; // edi
LINE4     byte *complete_scan_line_buffer; // esi
LINE5     bool v7; // zf
LINE6     byte *buff_mem_overwritten; // ebx
LINE7     byte *v9; // eax
LINE8     int v10; // eax
LINE9     int BytesPerLine; // eax
LINE10    int v12; // edi
LINE11    byte *v13; // ecx
LINE12    byte *v14; // ecx
LINE13    char v15; // bl
LINE14    byte *v16; // ecx
LINE15    byte *v17; // esi
LINE16    byte *v18; // eax
LINE17    char v19; // bl
LINE18    int v20; // esi
LINE19    unsigned int v21; // esi
LINE20    bool v22; // cf
LINE21    void *v23; // ecx
LINE22    unsigned int i; // eax
LINE23    int v26[13]; // [esp+Ch] [ebp-6Ch]
LINE24    int v27; // [esp+40h] [ebp-38h]
LINE25    byte *v28; // [esp+48h] [ebp-30h]
LINE26    int sizeX; // [esp+4Ch] [ebp-2Ch]
LINE27    int v30; // [esp+50h] [ebp-28h]
LINE28    unsigned int buffer_size_complete_scan_line; // [esp+54h] [ebp-24h]
LINE29    unsigned int v32; // [esp+58h] [ebp-20h]
LINE30    size_t invalid_size; // [esp+5Ch] [ebp-1Ch]
LINE31    byte *v34; // [esp+60h] [ebp-18h]
LINE32    byte *_complete_scan_line_buffer; // [esp+64h] [ebp-14h]
LINE33    byte *v36; // [esp+68h] [ebp-10h]
LINE34    byte *v37; // [esp+6Ch] [ebp-Ch]
LINE35    byte *v38; // [esp+70h] [ebp-8h]
LINE36    byte *v39; // [esp+74h] [ebp-4h]
LINE37
LINE38    v34 = 0;
LINE39    sizeX = getSizeX_0(IGDIB0bject);
LINE40    buffer_size_complete_scan_line = (unsigned __int16)p_pcx_data->BytesPerLine [6]
LINE41    * (unsigned __int8)p_pcx_data->color_planes;
LINE42    _buffer_size_complete_scan_line = buffer_size_complete_scan_line;
LINE43    invalid_size = compute_size_based_imagewidth.bitspersample(IGDIB0bject);
LINE44    sub_77C4AF60((int)ptr_function, a1, (int)v26, 5 * buffer_size_complete_scan_line, 1); [7]
LINE45    complete_scan_line_buffer = 0;
LINE46    v7 = p_pcx_data->encoding == 0;
LINE47    _complete_scan_line_buffer = 0;
LINE48    if ( !v7 )
LINE49    {
LINE50        complete_scan_line_buffer = AF_memm_alloc( [4]
LINE51            a1,
LINE52            _buffer_size_complete_scan_line,
LINE53            (int)"..\\..\\..\\..\\Common\\Formats\\pcxread.c",
LINE54            835);
LINE55        _complete_scan_line_buffer = complete_scan_line_buffer;
LINE56        if ( !_complete_scan_line_buffer )
LINE57            v34 = (byte *)kind_of_print_error(
LINE58                (int)"..\\..\\..\\..\\Common\\Formats\\pcxread.c",
LINE59                837,
LINE60                -1000,
LINE61                0,
LINE62                _buffer_size_complete_scan_line,
LINE63                a1,
LINE64                0);
LINE65    }
LINE66    buff_mem_overwritten = AF_memm_alloc(a1, invalid_size, (int)"..\\..\\..\\..\\Common\\Formats\\pcxread.c", 839); [5]
LINE67    v28 = buff_mem_overwritten;
LINE68    if ( buff_mem_overwritten )
LINE69        v9 = v34;
LINE70    else
LINE71        v9 = (byte *)kind_of_print_error(
LINE72            (int)"..\\..\\..\\..\\Common\\Formats\\pcxread.c",
LINE73            842,
LINE74            -1000,
LINE75            0,
LINE76            invalid_size,
LINE77            a1,
LINE78            0);
LINE79    if ( !v9 )
LINE80    {
LINE81        v27 = 0;
LINE82        v30 = 0;
LINE83        if ( getSizeY_0(IGDIB0bject) )
LINE84        {
LINE85            while ( !p_pcx_data->encoding )
LINE86            {
LINE87                complete_scan_line_buffer = (byte *)sub_77C4B280(v26, _buffer_size_complete_scan_line);
LINE88                _complete_scan_line_buffer = complete_scan_line_buffer;
LINE89                if ( !_complete_scan_line_buffer )
LINE90                {
LINE91                    v10 = kind_of_print_error(
LINE92                        (int)"..\\..\\..\\..\\Common\\Formats\\pcxread.c",
LINE93                        856,
LINE94                        -2051,
LINE95                        0,
LINE96                        _buffer_size_complete_scan_line,
LINE97                        a1,
LINE98                        0);
LINE99                LABEL_13:
LINE100                if ( v10 )
LINE101                    goto LABEL_25;
LINE102                }
LINE103                if ( p_pcx_data->bits_per_pixel == 1 ) [2]
LINE104                {
LINE105                    BytesPerLine = (unsigned __int16)p_pcx_data->BytesPerLine;
LINE106                    v12 = 0;
LINE107                    v34 = 8*complete_scan_line_buffer[BytesPerLine];
LINE108                    v13 = 8*complete_scan_line_buffer[BytesPerLine + BytesPerLine];
LINE109                    v38 = v13;
LINE110                    v14 = 8*v13[BytesPerLine];
LINE111                    v39 = complete_scan_line_buffer;
LINE112                    v37 = v14;
LINE113                    v36 = buff_mem_overwritten;
LINE114                    if ( sizeX )
LINE115                    {
LINE116                        v32 = ((unsigned int)(sizeX - 1) >> 1) + 1;
LINE117                        do

```

```

LINE118         {
LINE119             v15 = 2
LINE120             * (((unsigned __int8)(*v34 & byte_77E9E184[v12]) >> (7 - v12)) | (2
LINE121             * (((unsigned __int8)(*v38 & byte_77E9E184[v12])
>> (7 - v12)) | (2 * (((unsigned __int8)(*v14 & byte_77E9E184[v12]) >> (7 - v12))))));
LINE122             v16 = v37;
LINE123             v17 = v34;
LINE124             v18 = v36 + 1;
LINE125             *v36 = (((unsigned __int8)(*v39 & byte_77E9E184[v12]) >> (7 - v12)) | v15;
LINE126             v36 = v18;
LINE127             v19 = (unsigned __int8)(*v16 & byte_77E9E185[v12]) >> (6 - v12);
LINE128             v14 = v37;
LINE129             *v18 = (((unsigned __int8)(*v39 & byte_77E9E185[v12]) >> (6 - v12)) | (2
LINE130             * (((unsigned __int8)(*v17 & byte_77E9E185[v12])
>> (6 - v12)) | (2 * (((unsigned __int8)(*v38 & byte_77E9E185[v12]) >> (6 - v12)) | (2 * v19)))));
LINE131             if ( v12 == 6 )
LINE132             {
LINE133                 ++v39;
LINE134                 ++v38;
LINE135                 v12 = 0;
LINE136                 ++v14;
LINE137                 v34 = v17 + 1;
LINE138                 v37 = v14;
LINE139             }
LINE140             else
LINE141             {
LINE142                 v12 += 2;
LINE143             }
LINE144             v7 = v32-- == 1;
LINE145             v36 = v18 + 1;
LINE146             }
LINE147             while ( !v7 );
LINE148             buff_mem_overwritten = v28;
LINE149             _buffer_size_complete_scan_line = buffer_size_complete_scan_line;
LINE150         }
LINE151     }
LINE152     else
LINE153     {
LINE154         for ( i = 0; i < _buffer_size_complete_scan_line; ++i ) [3]
LINE155         {
LINE156             buff_mem_overwritten[2 * i] = complete_scan_line_buffer[i] >> 4; [1]
LINE157             buff_mem_overwritten[2 * i + 1] = complete_scan_line_buffer[i] & 0xF;
LINE158         }
LINE159     }
LINE160     v20 = v30;
LINE161     if ( !sub_77C494C0((int)ptr_function, (int)buff_mem_overwritten, v30, invalid_size) )
LINE162     {
LINE163         v21 = v20 + 1;
LINE164         v30 = v21;
LINE165         v22 = v21 < getSizeY_0(IGDI80bject);
LINE166         complete_scan_line_buffer = _complete_scan_line_buffer;
LINE167         if ( v22 )
LINE168             continue;
LINE169     }
LINE170     goto LABEL_25;
LINE171 }
LINE172 v10 = sub_77D3A2C0((int)v26, complete_scan_line_buffer, _buffer_size_complete_scan_line, (int)&v27);
LINE173 goto LABEL_13;
LINE174 }
LINE175 }
LINE176 LABEL_25:
LINE177 if ( _complete_scan_line_buffer && p_pcx_data->encoding )
LINE178     sub_77C55F40((void *)a1, _complete_scan_line_buffer, (int)".....\\Common\\Formats\\pcxread.c", 910);
LINE179 if ( buff_mem_overwritten )
LINE180     sub_77C55F40((void *)a1, buff_mem_overwritten, (int)".....\\Common\\Formats\\pcxread.c", 912);
LINE181 sub_77C4AC30(v26);
LINE182 return sub_77C2AA00(v23);
LINE183 }

```

In this algorithm we can observe a function `uncompress_scan_line`, whose objective is to decompress the pcx data, is crashing while filling the buffer `buff_mem_overwritten` in [1]. The path taken depends of the value from the pcx header `bits_per_pixel` not equal to 1 as we can see in [2]. The out-of-bounds occurs at [3], because of the lack of constraints against the two buffers `complete_scan_line_buffer` and `buff_mem_overwritten`. The two buffers, allocated respectively in [4] and [5], are using a size computed in [6] and [7]. As there is no length comparison between the two buffers, the access write violation may occur.

The size for the buffer `buff_mem_overwritten` is computed in the final destination function named `compute_size_for_pcx`:

```

LINE186 unsigned int __thiscall compute_size_for_pcx(IGDI80bject *this)
LINE187 {
LINE188     return ((this->sizeX * this->size_of_table_round * this->depth + 31) >> 3) & 0xFFFFFFF; [7]
LINE189     //
LINE190     // round_valued_bitperpixel_plane = round_max_bits_per_sample(bitspersample);
LINE191     //
LINE192     // sizeX = pcx_object->Xmax - pcx_object->Xmin + 1;
LINE193     // size_table_round = 1 is for up to 256 colors
LINE194     // return_1_if_for_colors_less_than_256
LINE195     // product_bit_per_pixel_nplanes = 1 or product_bit_per_pixel_nplanes = 4 or product_bit_per_pixel_nplanes = 8
LINE196     //
LINE197     //
LINE198     // depth is calculated from int __cdecl round_depth(int depth)
LINE199 }

```

In the formula in [7], what if `sizeX` is set to value 1 causing some allocation size based only against the two field left. `sizeX` is computed using the formula `sizeX = pcx_object->Xmax - pcx_object->Xmin + 1`; as we can see in the following pseudo code in function `build_IGDI80bject_pcxrelatedobject`.

So, if `Xmax` and `Xmin` are equal, `sizeX` will be assigned 1 at [8], which leads then to a very small buffer to be allocated, making the OOBW easier to trigger.

```

LINE308 int __stdcall build_IGDI8Object_pcxrelatedobject(int a1, pcx_object *pcx_object, ColorMapTable *pColorMapTable, IGDIBObject
*IGDI8Object)
LINE309 {
LINE310     pcx_object * _pcx_object; // edx
LINE311     int total_bits_per_pixel; // edi
LINE312     __int16 sizeX; // bx
LINE313     __int16 v7; // ax
LINE314     int v8; // eax
LINE315     int v9; // eax
LINE316     void *v10; // ecx
LINE317     void *dest_buffer; // eax
LINE318     int _size; // [esp+4h] [ebp-10h]
LINE319     __int16 sizeY; // [esp+18h] [ebp+Ch]
LINE320
LINE321     _pcx_object = pcx_object;
LINE322     total_bits_per_pixel = (unsigned __int8)pcx_object->bits_per_pixel * (unsigned __int8)pcx_object->color_planes;
LINE323     sizeX = pcx_object->xmax - pcx_object->xmin + 1; [8]
LINE324     sizeY = pcx_object->ymax - pcx_object->ymin + 1;
LINE325     [...]
LINE326 }

```

# Crash Information

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

KEY_VALUES_STRING: 1

    Key : AV.Fault
    Value: Write

    Key : Analysis.CPU.Sec
    Value: 0

    Key : Analysis.DebugAnalysisProvider.CPP
    Value: Create: 8007007e on DESKTOP-PJK7PVH

    Key : Analysis.DebugData
    Value: CreateObject

    Key : Analysis.DebugModel
    Value: CreateObject

    Key : Analysis.Elapsed.Sec
    Value: 8

    Key : Analysis.Memory.CommitPeak.Mb
    Value: 78

    Key : Analysis.System
    Value: CreateObject

    Key : Timeline.OS.Boot.DeltaSec
    Value: 302677

    Key : Timeline.Process.Start.DeltaSec
    Value: 991

ADDITIONAL_XML: 1

APPLICATION_VERIFIER_LOADED: 1

EXCEPTION_RECORD: (.exr -1)
ExceptionAddress: 5690a98c (igCore19d!IG_mpi_page_set+0x000df5fc)
ExceptionCode: c0000005 (Access violation)
ExceptionFlags: 00000000
NumberParameters: 2
    Parameter[0]: 00000001
    Parameter[1]: 0cd54000
Attempt to write to address 0cd54000

FAULTING_THREAD: 00003b54

PROCESS_NAME: simple.exe_141.exe

WRITE_ADDRESS: 0cd54000

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

EXCEPTION_CODE_STR: c0000005

EXCEPTION_PARAMETER1: 00000001

EXCEPTION_PARAMETER2: 0cd54000

STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
0093ee94 5690ba46 0093f414 1000001b 0093eef4 igCore19d!IG_mpi_page_set+0xdf5fc
0093eebc 5690a26a 0093f414 1000001b 0b6baff8 igCore19d!IG_mpi_page_set+0xe00b6
0093f38c 568007c9 0093f414 0b6baff8 00000001 igCore19d!IG_mpi_page_set+0xdeeda
0093f3c4 5683fb97 00000000 0b6baff8 0093f414 igCore19d!IG_image_savelist_get+0xb29
0093f640 5683f4f9 00000000 09bf5fa8 00000001 igCore19d!IG_mpi_page_set+0x14807
0093f600 567d6007 00000000 09bf5fa8 00000001 igCore19d!IG_mpi_page_set+0x14169
0093f680 00d859ac 09bf5fa8 0093f76c 0093f790 igCore19d!IG_load_file+0x47
0093f780 00d861a7 09bf5fa8 0093f8b4 00000021 simple_exe_141+0x159ac
0093f94c 00d86cbe 00000005 09ba2f50 09a85f38 simple_exe_141+0x161a7
0093f960 00d86b27 9c768deb 00d815e1 00d815e1 simple_exe_141+0x16cbe
0093f9bc 00d869bd 0093f9cc 00d86d38 0093f9dc simple_exe_141+0x16b27
0093f9c4 00d86d38 0093f9dc 76cd6359 00655000 simple_exe_141+0x169bd
0093f9cc 76cd6359 00655000 76cd6340 0093fa38 simple_exe_141+0x16d38
0093f9dc 77577b74 00655000 e8b707c2 00000000 KERNEL32!BaseThreadInitThunk+0x19
0093fa38 77577b44 ffffffff 77598f20 00000000 ntdll!_RtlUserThreadStart+0x2f
0093fa48 00000000 00d815e1 00655000 00000000 ntdll!_RtlUserThreadStart+0x1b

STACK_COMMAND: ~0s ; .cxr ; kb

SYMBOL_NAME: igCore19d!IG_mpi_page_set+df5fc

MODULE_NAME: igCore19d

IMAGE_NAME: igCore19d.dll

FAILURE_BUCKET_ID: INVALID_POINTER_WRITE_AVRF_c0000005_igCore19d.dll!IG_mpi_page_set

OS_VERSION: 10.0.18362.239

BUILDLAB_STR: 19h1_release_svc_prod1

OSPLATFORM_TYPE: x86

OSNAME: Windows 10

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

Followup: MachineOwner
-----
```

#### Timeline

2020-01-27 - Vendor Disclosure

2020-02-10 - Public Release

#### CREDIT

Discovered by Emmanuel Tacheau of Cisco Talos.

---

VULNERABILITY REPORTS

PREVIOUS REPORT

NEXT REPORT

TALOS-2020-0986

TALOS-2020-0989

---