Talos Vulnerability Report

TALOS-2020-1004

Accusoft ImageGear ICO ico_read buffer size computation code execution vulnerability

MAY 5, 2020

CVE NUMBER

CVE-2020-6082

Summary

An exploitable out-of-bounds write vulnerability exists in the `ico_read` function of the igcore19d.dll library of Accusoft ImageGear 19.6.0. A specially crafted ICO 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 19.4.0
Accusoft ImageGear 19.5.0
Accusoft ImageGear 19.6.0

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-190: Integer Overflow or Wraparound

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 `ico_read` function, due to an invalid comparison check. A specially crafted ICO file can lead to an out-of-bounds write, which can result in remote code execution.

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

```
eax=00000033 ebx=155d0080 ecx=125c0ffb edx=00000018 esi=089cf002 edi=155d1001
eip=5f353caa esp=00eff238 ebp=00eff294 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010246
igCore19d!IG_mpi_page_set+0xa8afa:
5f353caa 8846fe          mov     byte ptr [esi-2],al        ds:002b:089cf000=??

0:000> kb
 # ChildEBP RetAddr  Args to Child
WARNING: Stack unwind information not available. Following frames may be wrong.
00 00eff294 5f35333a 00eff7e4 1000001b 0e51aff8 igCore19d!IG_mpi_page_set+0xa8afa
01 00eff75c 5f2804a9 00eff7e4 0e51aff8 00000001 igCore19d!IG_mpi_page_set+0xa818a
02 00eff794 5f2bf8f7 00000000 0e51aff8 00eff7e4 igCore19d!IG_image_savelist_get+0xb29
03 00effa10 5f2bf259 00000000 09fe7fd8 00000001 igCore19d!IG_mpi_page_set+0x14747
04 00effa30 5f255fb7 00000000 09fe7fd8 00000001 igCore19d!IG_mpi_page_set+0x140a9
05 00effa50 00365d5c 09fe7fd8 00effb3c 00effb60 igCore19d!IG_load_file+0x47
06 00effb50 003661a7 09fe7fd8 00effc84 00000021 Fuzzme!fuzzme+0x3c [c:\work\git_vrt\fuzzme\fuzzme.cpp @ 62]
07 00effd1c 00366cbe 00000005 09f94f88 09e79f40 Fuzzme!main+0x2d7 [c:\work\git_vrt\fuzzme\fuzzme.cpp @ 141]
08 00effd30 00366b27 12a856c4 003615e1 003615e1 Fuzzme!invoke_main+0x1e
[d:\agent\_work\3\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl @ 78]
09 00effd8c 003669bd 00effd9c 00366d38 00effdac Fuzzme!__scrt_common_main_seh+0x157
[d:\agent\_work\3\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl @ 288]
0a 00effd94 00366d38 00effdac 764d6359 00c2a000 Fuzzme!__scrt_common_main+0xd
[d:\agent\_work\3\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl @ 331]
0b 00effd9c 764d6359 00c2a000 764d6340 00effe08 Fuzzme!mainCRTStartup+0x8
[d:\agent\_work\3\s\src\vctools\crt\vcstartup\src\startup\exe_main.cpp @ 17]
0c 00effdac 77a37b74 00c2a000 f7f09f6c 00000000 KERNEL32!BaseThreadInitThunk+0x19
0d 00effe08 77a37b44 ffffffff 77a58f15 00000000 ntdll!__RtlUserThreadStart+0x2f
0e 00effe18 00000000 003615e1 00c2a000 00000000 ntdll!_RtlUserThreadStart+0x1b
```

As we can see, an out-of-bounds write operation occurred.
The pseudo-code of this vulnerable function looks like this:

```
LINE1   int __stdcall ico_read(table_function *table_func, int a1, int a3, ICOFile *ICOFileData, IGDIBOject *a2, ICOPalette *icoPaletteData)
LINE2   {
LINE3     int _num_entries; // eax
LINE4     int _bitBitCount; // esi
LINE5     int num_entries; // edi
LINE6     int result; // eax
LINE7     int v10; // eax
LINE8     unsigned int size_buffer_1; // edi
LINE9     unsigned int size_buffer_3; // ebx
LINE10    byte *buffer_1; // edi
LINE11    table_function *_table_func; // esi
LINE12    byte *buffer_3; // ebx
LINE13    int current_offset; // ecx
LINE14    unsigned __int8 v17; // bl
LINE15    int next_offset; // eax
LINE16    int __bitBitCount; // edx
LINE17    _BYTE *v20; // edx
LINE18    _BYTE *buffer_3_next_entry; // ecx
LINE19    int v22; // edi
LINE20    int tmp_biwidth; // esi
LINE21    char v24; // cl
LINE22    int v25; // esi
LINE23    unsigned __int8 v26; // dl
LINE24    byte *v27; // eax
LINE25    byte *v28; // edi
LINE26    char v29; // cl
LINE27    byte v30; // al
LINE28    char v31; // al
LINE29    unsigned __int8 v32; // bh
LINE30    char *_buffer_1_next_second_entry; // esi
LINE31    char *_buffer_3_next_entry; // edi
LINE32    byte *v35; // ecx
LINE33    char v36; // al
LINE34    char v37; // al
LINE35    bool __biWidth; // zf
LINE36    int v39; // eax
LINE37    int v40; // [esp+Ch] [ebp-50h]
LINE38    unsigned int size; // [esp+10h] [ebp-4Ch]
LINE39    int biHeight; // [esp+1Ch] [ebp-40h]
LINE40    int v43; // [esp+20h] [ebp-3Ch]
LINE41    int _biWidth; // [esp+24h] [ebp-38h]
LINE42    int v45; // [esp+28h] [ebp-34h]
LINE43    int tmp_offset; // [esp+2Ch] [ebp-30h]
LINE44    unsigned int size_buffer_2; // [esp+30h] [ebp-2Ch]
LINE45    size_t _size_buffer_3; // [esp+34h] [ebp-28h]
LINE46    int v49; // [esp+38h] [ebp-24h]
LINE47    int v50; // [esp+3Ch] [ebp-20h]
LINE48    int biBitCount; // [esp+40h] [ebp-1Ch]
LINE49    byte *buffer_2; // [esp+44h] [ebp-18h]
LINE50    byte *__buffer_1; // [esp+48h] [ebp-14h]
LINE51    char v54; // [esp+4Ch] [ebp-10h]
LINE52    byte *v55; // [esp+4Ch] [ebp-10h]
LINE53    byte *v56; // [esp+50h] [ebp-Ch]
LINE54    byte *v57; // [esp+50h] [ebp-Ch]
LINE55    int biWidth; // [esp+54h] [ebp-8h]
[10]
LINE56    int v59; // [esp+58h] [ebp-4h]
LINE57    byte *_buffer_3; // [esp+70h] [ebp+14h]
LINE58
LINE59    _num_entries = get_field34(a2);               // set to 4 or 2 depending on bibitcount value
[6]
LINE60    _bitBitCount = (unsigned __int16)ICOFileData->TBitmapInfoHeader.biBitCount;
LINE61    num_entries = _num_entries;
LINE62    biBitCount = (unsigned __int16)ICOFileData->TBitmapInfoHeader.biBitCount;
LINE63    result = sub_650884B0(table_func, a2);
LINE64    if ( !result )
LINE65    {
LINE66      biHeight = getSizeY_0(a2);
LINE67      biWidth = getBiWidth(a2);
LINE68      integer_value = num_entries * getBiWidth(a2);
[5]
LINE69      size_buffer_1 = ((8 * integer_value + 31) >> 3) & 0xFFFFFFFC;
[4]
LINE70      var_4c = ((8 * integer_value + 31) >> 3) & 0xFFFFFFFC;
LINE71      size_buffer_2 = ((getBiWidth(a2) + 31) >> 3) & 0xFFFFFFFC;
LINE72      size_buffer_3 = ((_bitBitCount * getBiWidth(a2) + 31) >> 3) & 0xFFFFFFFC;   // biBitCount * biWidth / 8
[7]
LINE73      _size_buffer_3 = size_buffer_3;
LINE74      buffer_1 = AF_memm_alloc(a1, size_buffer_1, (int)"..\\..\\..\\..\\Common\\Formats\\icoread.c", 830);
[8]
LINE75      __buffer_1 = buffer_1;
LINE76      buffer_2 = AF_memm_alloc(a1, size_buffer_2, (int)"..\\..\\..\\..\\Common\\Formats\\icoread.c", 831);
LINE77      _table_func = table_func;
LINE78      buffer_3 = AF_memm_alloc(a1, size_buffer_3, (int)"..\\..\\..\\..\\Common\\Formats\\icoread.c", 832);
[9]
LINE79      _buffer_3 = buffer_3;
LINE80      current_offset = get_current_offset(table_func);
LINE81      tmp_offset = current_offset + biHeight * _size_buffer_3;
LINE82      v43 = 0;
LINE83      if ( biHeight > 0 )
LINE84      {
LINE85      [...]
LINE86          if ( biWidth > 0 )
[11]
LINE87          {
LINE88            v55 = buffer_2;
LINE89            v57 = _buffer_3;
LINE90            _buffer_1_next_second_entry = (char *)(buffer_1 + 2);
LINE91            _buffer_3_next_entry = (char *)(_buffer_3 + 1);
LINE92            _biWidth = biWidth;
LINE93            v35 = buffer_2;
LINE94            v45 = (int)(_buffer_3 + 1);
LINE95            do
[2]
LINE96            {
LINE97              if ( __bitBitCount == 24 )
LINE98              {
LINE99                *(_buffer_1_next_second_entry - 2) = _buffer_3_next_entry[1];
[1]
LINE100               *(_buffer_1_next_second_entry - 1) = *_buffer_3_next_entry;
LINE101               v36 = *(_buffer_3_next_entry - 1);
LINE102             }
LINE103             else
LINE104             {
LINE105               *(_buffer_1_next_second_entry - 2) = icoPaletteData->palette_entry[(unsigned __int8)(v32 & *v57) >> v31].rgbRed;
LINE106               *(_buffer_1_next_second_entry - 1) = icoPaletteData->palette_entry[(unsigned __int8)(v32 & *v57) >> v31].rgbGreen;
LINE107               _buffer_3_next_entry = (char *)v45;
```

```
LINE108               v36 = icoPaletteData->palette_entry[(unsigned __int8)(v32 & *v57) >> v31].rgbBlue;
LINE109               v35 = v55;
LINE110             }
LINE111             *_buffer_1_next_second_entry = v36;
LINE112             v37 = (unsigned __int8)(v17 & *v35) >> v59--;
LINE113             v17 >>= 1;
LINE114             _buffer_1_next_second_entry[1] = v37 - 1;
LINE115             if ( !v17 )
LINE116             {
LINE117               ++v55;
LINE118               v59 = 7;
LINE119               v17 = 0x80;
LINE120             }
LINE121             __bitBitCount = biBitCount;
LINE122             v32 >>= biBitCount;
LINE123             v31 = v50 - biBitCount;
LINE124             v50 -= biBitCount;
LINE125             if ( !v32 )
LINE126             {
LINE127               v31 = 8 - biBitCount;
LINE128               v32 = -1 << (8 - biBitCount);
LINE129               ++v57;
LINE130               v50 = 8 - biBitCount;
LINE131             }
LINE132             v35 = v55;
LINE133             _buffer_3_next_entry += 3;
LINE134             _buffer_1_next_second_entry += 4;
LINE135             __biWidth = _biWidth-- == 1;
[3]
LINE136             v45 = (int)_buffer_3_next_entry;
LINE137           }
LINE138           while ( !__biWidth );
[2]
LINE139 LABEL_29:
LINE140           buffer_1 = __buffer_1;
LINE141 LABEL_30:
LINE142           _table_func = table_func;
LINE143         }
LINE144         [...]
LINE145     }
LINE146 }
```

In this algorithm we can observe a function `ico_read`, whose objective is to copy content from `buffer_3` into `buffer_1`, is crashing while filling the buffer `buffer_1` in [1].

This copy is controlled by a do while loop [2], which terminates when the decremented variable `__biWidth` in [3] is 0. We can observe that the size computed for `buffer_1` in [4] is controlled by `integer_value`, while the size of `buffer_3` [7] is computed from `biBitCount` and `biWidth`.
The `integer_value` variable is the result of `num_entries` times `biBitCount` [5], where `num_entries` is computed at [6] and is a constant that is either '4' or '2', depending of the `biBitCount` value obtained from the file.

An integer overflow can happen in [4] when calculating the size for `buffer_1`, which is obtained by multiplying `integer_value` by 8:

```
size_buffer_1 = ((8 * integer_value + 31) >> 3) & 0xFFFFFFFC;
```

The overflow happens when `integer_value` is bigger than 0x1fffffffc, which is possible by controlling `biWidth` and `biBitCount`. If an overflow happens, the buffer `buffer_1` will have a smaller size than `buffer_3`, leading to an out-of-bounds write during the copy at [1].

Thus by carefully manipulating `biBitCount` and `biWidth`, an attacker could exploit this memory corruption to execute arbitrary code.

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: 4

        Key  : Analysis.Memory.CommitPeak.Mb
        Value: 83

        Key  : Analysis.System
        Value: CreateObject

        Key  : Timeline.OS.Boot.DeltaSec
        Value: 424445

        Key  : Timeline.Process.Start.DeltaSec
        Value: 126


ADDITIONAL_XML: 1

APPLICATION_VERIFIER_LOADED: 1

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 5f353caa (igCore19d!IG_mpi_page_set+0x000a8afa)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 00000001
   Parameter[1]: 089cf000
Attempt to write to address 089cf000

FAULTING_THREAD:  00003a18

PROCESS_NAME:  Fuzzme.exe

WRITE_ADDRESS:  089cf000

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:  089cf000

STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
00eff294 5f35333a 00eff7e4 1000001b 0e51aff8 igCore19d!IG_mpi_page_set+0xa8afa
00eff75c 5f2804a9 00eff7e4 0e51aff8 00000001 igCore19d!IG_mpi_page_set+0xa818a
00eff794 5f2bf8f7 00000000 0e51aff8 00eff7e4 igCore19d!IG_image_savelist_get+0xb29
00effa10 5f2bf259 00000000 09fe7fd8 00000001 igCore19d!IG_mpi_page_set+0x14747
00effa30 5f255fb7 00000000 09fe7fd8 00000001 igCore19d!IG_mpi_page_set+0x140a9
00effa50 00365d5c 09fe7fd8 00effb3c 00effb60 igCore19d!IG_load_file+0x47
00effb50 003661a7 09fe7fd8 00effc84 00000021 Fuzzme!fuzzme+0x3c
00effd1c 00366cbe 00000005 09f94f88 09e79f40 Fuzzme!main+0x2d7
00effd30 00366b27 12a856c4 003615e1 003615e1 Fuzzme!invoke_main+0x1e
00effd8c 003669bd 00effd9c 00366d38 00effdac Fuzzme!__scrt_common_main_seh+0x157
00effd94 00366d38 00effdac 764d6359 00c2a000 Fuzzme!__scrt_common_main+0xd
00effd9c 764d6359 00c2a000 764d6340 00effe08 Fuzzme!mainCRTStartup+0x8
00effdac 77a37b74 00c2a000 f7f09f6c 00000000 KERNEL32!BaseThreadInitThunk+0x19
00effe08 77a37b44 ffffffff 77a58f15 00000000 ntdll!__RtlUserThreadStart+0x2f
00effe18 00000000 003615e1 00c2a000 00000000 ntdll!_RtlUserThreadStart+0x1b


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  igCore19d!IG_mpi_page_set+a8afa

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

2020-02-11 - Vendor Disclosure

2020-04-30 - Vendor Patched

2020-05-05 - Public Release

CREDIT

Discovered by Emmanuel Tacheau of Cisco Talos.
