Talos Vulnerability Report

TALOS-2020-1017

Accusoft ImageGear TIFF fill_in_raster buffer copy operation code execution vulnerability

MAY 5, 2020

CVE NUMBER

CVE-2020-6094

Summary

An exploitable code execution vulnerability exists in the TIFF fill_in_raster function of the igcore19d.dll library of Accusoft ImageGear 19.4, 19.5 and 19.6. A specially crafted TIFF file can cause an out-of-bounds write, resulting in remote code execution. An attacker can provide a malicious file to trigger this vulnerability.

Tested Versions

Accusoft ImageGear 19.4
Accusoft ImageGear 19.5
Accusoft ImageGear 19.6

Product URLs

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-190 - Integer Overflow or Wraparound

Details

This document image processing toolkit allows you to quickly integrate document handling functions like image conversion, creation, editing, annotations, viewing, scanning, and printing to your application. With ImageGear, you can read and write more than 100 different document and image file formats.

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 an integer overflow. A specially crafted TIFF file can lead to an out-of-bounds write which can result in remote code execution.

```
eax=0e7cd002 ebx=00000080 ecx=00000138 edx=00000080 esi=00000004 edi=400000b4
eip=5fbe3be0 esp=006ff0c4 ebp=006ff0d0 iopl=0         nv up ei ng nz na pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010287
igCore19d!IG_mpi_page_set+0x8850:
5fbe3be0 0058ff          add     byte ptr [eax-1],bl        ds:002b:0e7cd001=??
```

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

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

```
LINE1  void __cdecl fill_in_raster(int samplePerPixel,4_entries *param_2,int ImageWidth,byte *heap_buffer)
[4]
LINE2  {
LINE3    int rounded_depth;
LINE4    int *piVar1;
LINE5    short *psVar2;
LINE6    byte *buffer;
LINE7    int iVar3;
LINE8    int iVar4;
LINE9
LINE10   rounded_depth = get_from_max_depth(samplePerPixel,param_2);
LINE11   samplePerPixel*ImageWidth = samplePerPixel * ImageWidth;
[3]
LINE12   if (samplePerPixel < 3) {
LINE13     throw_some_exception
LINE14               (0xffffffe6f,0,samplePerPixel,3,"..\\..\\..\\..\\Common\\Core\\Raster.cpp",0x414);
LINE15   }
LINE16   iVar4 = 1 << ((char)param_2->field_0x4 - 1U & 0x1f);
LINE17   iVar3 = 1 << ((char)param_2->field_0x8 - 1U & 0x1f);
LINE18   if (rounded_depth == 8) {
LINE19     if (0 < samplePerPixel*ImageWidth) {
LINE20       buffer = heap_buffer + 2;
[5]
LINE21       do {
LINE22         buffer[-1] = buffer[-1] + (char)iVar4;
[1]
LINE23         *buffer = *buffer + (char)iVar3;
LINE24         buffer = buffer + samplePerPixel;
LINE25       } while ((int)(buffer + (-2 - (int)heap_buffer)) < samplePerPixel*ImageWidth);
[2]
LINE26     }
LINE27   }
LINE28   else {
LINE29     if (rounded_depth == 0x10) {
LINE30       rounded_depth = 0;
LINE31       if (0 < samplePerPixel*ImageWidth) {
LINE32         psVar2 = (short *)(heap_buffer + 4);
LINE33         do {
LINE34           psVar2[-1] = psVar2[-1] + (short)iVar4;
LINE35           *psVar2 = *psVar2 + (short)iVar3;
LINE36           psVar2 = psVar2 + samplePerPixel;
LINE37           rounded_depth = rounded_depth + samplePerPixel;
LINE38         } while (rounded_depth < samplePerPixel*ImageWidth);
LINE39         return;
LINE40       }
LINE41     }
LINE42     else {
LINE43       if ((rounded_depth == 0x20) && (rounded_depth = 0, 0 < samplePerPixel*ImageWidth)) {
LINE44         piVar1 = (int *)(heap_buffer + 8);
LINE45         do {
LINE46           piVar1[-1] = piVar1[-1] + iVar4;
LINE47           *piVar1 = *piVar1 + iVar3;
LINE48           piVar1 = piVar1 + samplePerPixel;
LINE49           rounded_depth = rounded_depth + samplePerPixel;
LINE50         } while (rounded_depth < samplePerPixel*ImageWidth);
LINE51         return;
LINE52       }
LINE53     }
LINE54   }
LINE55   return;
LINE56 }
```

In this algorithm we can observe a function `fill_in_raster`, whose objective is to process TIFF data, which is crashing while filling the buffer `buffer` in [1]. This buffer is derived from `heap_buffer` passed as argument in the function in [4] and assigned to it in [5].

We can observe the do-while loop controlled by the variable `samplePerPixel*ImageWidth` in [2]. This variable is the product of two TIFF tags values computed in [3], where `samplePerPixel` value is read from the TIFF tags `SamplesPerPixel` and `ImageWidth` is read from the TIFF tags `ImageWidth`.

Now, if we take a closer look at the buffer itself, we can see the size is quite small, in our case, 134 bytes.

```
0:000> !heap -p -a eax
        address 0c944002 found in
        _DPH_HEAP_ROOT @ 4ec1000
        in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                                                    c9f0a90:          c943ec8              134 -          c943000
2000
        5fb0ab70 verifier!AVrfDebugPageHeapAllocate+0x00000240
        77ab8fcb ntdll!RtlDebugAllocateHeap+0x00000039
        77a0bb0d ntdll!RtlpAllocateHeap+0x000000ed
        77a0b02f ntdll!RtlpAllocateHeapInternal+0x0000022f
        77a0adee ntdll!RtlAllocateHeap+0x0000003e
        5ef2dcff MSVCR110!malloc+0x00000049
        5f2a563e igCore19d!AF_memm_alloc+0x0000001e
        5f3b578f igCore19d!IG_mpi_page_set+0x0010a5df
        5f3b518a igCore19d!IG_mpi_page_set+0x00109fda
        5f3ba203 igCore19d!IG_mpi_page_set+0x0010f053
        5f3b423b igCore19d!IG_mpi_page_set+0x0010908b
        5f2804a9 igCore19d!IG_image_savelist_get+0x00000b29
        5f2bf8f7 igCore19d!IG_mpi_page_set+0x00014747
        5f2bf259 igCore19d!IG_mpi_page_set+0x000140a9
        5f255fb7 igCore19d!IG_load_file+0x00000047
        00365d5c Fuzzme!fuzzme+0x0000003c [c:\work\git_vrt\fuzzme\fuzzme.cpp @ 62]
        003661a7 Fuzzme!main+0x000002d7 [c:\work\git_vrt\fuzzme\fuzzme.cpp @ 141]
        00366cbe Fuzzme!invoke_main+0x0000001e [d:\agent\_work\3\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl @ 78]
        00366b27 Fuzzme!__scrt_common_main_seh+0x00000157 [d:\agent\_work\3\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl @ 288]
        003669bd Fuzzme!__scrt_common_main+0x0000000d [d:\agent\_work\3\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl @ 331]
        00366d38 Fuzzme!mainCRTStartup+0x00000008 [d:\agent\_work\3\s\src\vctools\crt\vcstartup\src\startup\exe_main.cpp @ 17]
        764d6359 KERNEL32!BaseThreadInitThunk+0x00000019
        77a37b74 ntdll!__RtlUserThreadStart+0x0000002f
        77a37b44 ntdll!_RtlUserThreadStart+0x0000001b
```

When going back to the code, the size of `heap_buffer` is computed previously through the function `IGDIBStd::compute_size`, which is the vulnerable function:

```
LINE58 size = compute_size_from_bibitWidth_operations(IGDIBStd_Object);
LINE59
LINE60 uint __thiscall IGDIBStd::compute_size(IGDIBStd *this)
LINE61 {
LINE62   return (int)(this->color_depth * this->SamplesPerPixel * this->biWidth + 0x1f) >> 3 & 0xfffffffc;
[6]
LINE63 }
```

The `this->SamplesPerPixel` and `this->biWidth` are directly controlled by the TIFF file data, while `this->color_depth` is derived from `SamplePerPixel` and equal to `8` in this case.

The issue is that this formula is prone to integer overflow [6] since, by choosing the correct values in the TIFF file, an attacker could make this function to return a very small value, causing thus a buffer to be allocated with a size which is too small. The loop in the `fill_in_raster` function will then write out of bounds at [1].

Timeline

2020-02-19 - Vendor Disclosure

2020-04-30 - Vendor Patched

2020-05-05 - Public Release

CREDIT

Discovered by Emmanuel Tacheau of Cisco Talos.
