Talos Vulnerability Report

TALOS-2020-0990

Accusoft ImageGear JPEG SOFx Code Execution Vulnerability

FEBRUARY 10, 2020

CVE NUMBER

CVE-2020-6066

Summary

An exploitable out-of-bounds write vulnerability exists in the igcore19d.dll JPEG SOFx parser of the Accusoft ImageGear 19.5.0 library. A specially crafted JPEG 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 JPEG raster image parser. A specially crafted JPEG file can lead to an out-of-bounds write resulting in remote code execution.

If we try to load a malformed JPEG file via the `IG_load_file` function we end up in the following situation:

```
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=0bae1000 edx=00000000 esi=00000000 edi=0bae0ffe
eip=5b9c2181 esp=012ff514 ebp=012ff65c 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+0xb6df1:
5b9c2181 66890471        mov     word ptr [ecx+esi*2],ax  ds:002b:0bae1000=????
```

The calculated address `0x0bae1000` points to the page with the PAGE_GUARD flag causing an access violation. Checking attributes related with the buffer we can see : 0:000> !heap -p -a 0bae1000 address 0bae1000 found in _DPH_HEAP_ROOT @ 5c21000 in busy allocation (DPH_HEAP_BLOCK: UserAddr UserSize - VirtAddr VirtSize) be8164c: badf000 2000 - bade000 4000 5bbfab70 verifier!AVrfDebugPageHeapAllocate+0x00000240 77378fcb ntdll!RtlDebugAllocateHeap+0x00000039 772cbb0d ntdll!RtlpAllocateHeap+0x000000ed 772cb02f ntdll!RtlpAllocateHeapInternal+0x0000022f 772cadee ntdll!RtlAllocateHeap+0x0000003e 5b56daff MSVCR110!malloc+0x00000049 5b90582e igCore19d!AF_memm_alloc+0x0000001e 5b9c59aa igCore19d!IG_mpi_page_set+0x000ba61a 5b9bef63 igCore19d!IG_mpi_page_set+0x000b3bd3 5b9d6115 igCore19d!IG_mpi_page_set+0x000cad85 5b9d605c igCore19d!IG_mpi_page_set+0x000caccc 5b9d40b1 igCore19d!IG_mpi_page_set+0x000c8d21 5b9d57da igCore19d!IG_mpi_page_set+0x000ca44a 5b8e07c9 igCore19d!IG_image_savelist_get+0x00000b29 5b91fb97 igCore19d!IG_mpi_page_set+0x00014807 5b91f4f9 igCore19d!IG_mpi_page_set+0x00014169 5b8b6007 igCore19d!IG_load_file+0x00000047 00ef59ac simple_exe_141+0x000159ac 00ef61a7 simple_exe_141+0x000161a7 00ef6cbe simple_exe_141+0x00016cbe 00ef6b27 simple_exe_141+0x00016b27 00ef69bd simple_exe_141+0x000169bd 00ef6d38 simple_exe_141+0x00016d38 74f56359 KERNEL32!BaseThreadInitThunk+0x00000019 772f7b74 ntdll!__RtlUserThreadStart+0x0000002f 772f7b44 ntdll!_RtlUserThreadStart+0x0000001b

that an attempt of write operation was performed just after the available buffer memory range. Further analysis revealed that:

```
the buffer is allocated based on values : (SOFx->X_image * 2) * (SOFx->COMPS[1]->compNr +1)
SOFx->X_image             : offset -> 0xA5
SOFx->COMPS[1]->compNr : offset -> 0xAD
```

Operations related with the above buffer are made inside a while loop which looks as follows:

```
Line 1  int __stdcall sub_5DD71490(int *a1, _DWORD *a2, __int16 a3, int a4, int a5, int (__stdcall *a6)(signed int, int, int *, _DWORD *))
Line 2  {
Line 3
Line 4          while ( !a3 )
Line 5          {
Line 6  LABEL_112:
Line 7            v63 = 0;
Line 8            v142 = 0;
Line 9            if ( __nr_comp <= 0 )
Line 10                 goto LABEL_214;
Line 11           __v64 = (struct_v64 *)v152;
Line 12           v65 = 0;
Line 13           do
Line 14           {
Line 15                 v66 = __v64->someArray;
Line 16                 i__ = 80 * v65;
Line 17                 loop_index = 0;
Line 18                 v137 = 80 * v65;
Line 19                 if ( *(_DWORD *)(80 * v65 + v66 + 28) <= 0 )
Line 20                   goto LABEL_212;
Line 21                 v68 = v65;
Line 22                 v124 = v68 * 4;
Line 23                 v69 = v166[v65];
Line 24                 v70 = v164[v68] - v69;
Line 25                 v127__ = v69;
Line 26                 inside_loop_index = loop_index;
Line 27                 value_var = v70;
Line 28                 do
Line 29                 {
Line 30                   baseAddr = __v64->someArray;
Line 31                   value_ecx = v127__ + inside_loop_index;
Line 32                   value_esi = *(_DWORD *)(baseAddr + i__ + 44);
Line 33                   someStruct = (struct_v75 *)(i__ + baseAddr);
Line 34                   basePtr = someStruct->basePtr;
Line 35                   v77 = someStruct->dword18 == 0;
Line 36                   v78 = someStruct->dword18 < 0;
Line 37                   _buffer = &basePtr[value_esi * (value_ecx + value_var)];
Line 38                   v80 = &basePtr[value_ecx * value_esi];
Line 39                   i__ = v137;
Line 40                   buffer = _buffer;
Line 41                   v144 = v80;
Line 42                   v159 = 0;
Line 43                   if ( !v78 && !v77 )
Line 44                   {
Line 45                         v155 = _buffer - 1;
Line 46                         while ( 1 )
Line 47                         {
Line 48                           if ( v158 < 16 )
Line 49                           {
Line 50
Line 51 LABEL_196:
Line 52                           if ( *v152 <= 8 )
Line 53                                 buffer[index] = (unsigned __int8)(v109 + ((_BYTE)v99 << v131));
Line 54                           else
Line 55                                 buffer[index] = v109 + ((_WORD)v99 << v131);
Line 56                                 }
Line 57 LABEL_210:
Line 58                   someArray = __v64->someArray;
Line 59                   inside_loop_index = loop_index + 1;
Line 60                   loop_index = inside_loop_index;
Line 61                 }
Line 62                 while ( inside_loop_index < *(_DWORD *)(i__ + someArray + 28) );
Line 63                 __nr_comp = nr_comp;
Line 64                 v63 = (int)v142;
Line 65 LABEL_212:
Line 66                 ++v63;
Line 67                 *(_DWORD *)(__v64->someArray + i__ + 32) += 2 * *(_DWORD *)(__v64->someArray + i__ + 24);
Line 68                 v65 = (signed __int16)v63;
Line 69                 v142 = (_DWORD *)v63;
Line 70           }
Line 71           while ( (signed __int16)v63 < __nr_comp );
Line 72
```

Important for us is while loop `line 62`, which is controlled by :

```
        (SOFx->COMPS[1]->Vert & 0xF)
         offset - > 0xAC
```

Each loop cycle the buffer base address is increased by :

```
        SOFx->X_image
```

which in our case equals 0x1000. We can quickly estimate that after second iteration an out-of-bounds write operation will appear in `line 53`.

As we can see an attacker controls all presented variables just by proper file content manipulation. Increasing the loop count via the `SOFx->COMPS[1]->Vert` variable allows an attacker to cause an out-of-bounds write leading to memory corruption which can result in remote code execution.

Crash Information

```
(7588.5f40): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=0bd0d000 edx=00000000 esi=00000000 edi=0bd0cffe
eip=5b9c2181 esp=00afeed0 ebp=00aff018 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+0xb6df1:
5b9c2181 66890471        mov     word ptr [ecx+esi*2],ax  ds:002b:0bd0d000=????
0:000> kb
 # ChildEBP RetAddr  Args to Child
WARNING: Stack unwind information not available. Following frames may be wrong.
00 00aff018 5b9bf1e6 09864f60 00aff044 00000001 igCore19d!IG_mpi_page_set+0xb6df1
01 00aff0a4 5b9d6115 00000003 5b9d2100 0b708720 igCore19d!IG_mpi_page_set+0xb3e56
02 00aff0c0 5b9d605c 0b708720 09864f60 0000ffda igCore19d!IG_mpi_page_set+0xcad85
03 00aff0e4 5b9d40b1 0b708720 09864f60 00aff10c igCore19d!IG_mpi_page_set+0xcaccc
04 00aff104 5b9d57da 00afffc3 1000001b 0987cf70 igCore19d!IG_mpi_page_set+0xc8d21
05 00aff144 5b8e07c9 1000001b 0987cf70 00000001 igCore19d!IG_mpi_page_set+0xca44a
06 00aff17c 5b91fb97 00000000 0987cf70 00aff1cc igCore19d!IG_image_savelist_get+0xb29
07 00aff3f8 5b91f4f9 00000000 09ed1fb8 00000001 igCore19d!IG_mpi_page_set+0x14807
08 00aff418 5b8b6007 00000000 09ed1fb8 00000001 igCore19d!IG_mpi_page_set+0x14169
09 00aff438 00ef59ac 09ed1fb8 00aff524 00aff548 igCore19d!IG_load_file+0x47
0a 00aff538 00ef61a7 09ed1fb8 00aff66c 00000021 simple_exe_141+0x159ac
0b 00aff704 00ef6cbe 00000004 09e7ef90 09d59f20 simple_exe_141+0x161a7
0c 00aff718 00ef6b27 c72c7e22 00ef15e1 00ef15e1 simple_exe_141+0x16cbe
0d 00aff774 00ef69bd 00aff784 00ef6d38 00aff794 simple_exe_141+0x16b27
0e 00aff77c 00ef6d38 00aff794 74f56359 008e4000 simple_exe_141+0x169bd
0f 00aff784 74f56359 008e4000 74f56340 00aff7f0 simple_exe_141+0x16d38
10 00aff794 772f7b74 008e4000 0ea51c40 00000000 KERNEL32!BaseThreadInitThunk+0x19
11 00aff7f0 772f7b44 ffffffff 77318f0c 00000000 ntdll!__RtlUserThreadStart+0x2f
12 00aff800 00000000 00ef15e1 008e4000 00000000 ntdll!_RtlUserThreadStart+0x1b
0:000> lmva eip
Browse full module list
start    end        module name
5b8a0000 5bbe9000   igCore19d   (export symbols)       d:\projects\ImageGear\current\Build\Bin\x86\igCore19d.dll
        Loaded symbol image file: d:\projects\ImageGear\current\Build\Bin\x86\igCore19d.dll
        Image path: d:\projects\ImageGear\current\Build\Bin\x86\igCore19d.dll
        Image name: igCore19d.dll
        Browse all global symbols  functions  data
        Timestamp:        Fri Nov 22 15:45:29 2019 (5DD7F489)
        CheckSum:         00356062
        ImageSize:        00349000
        File version:     19.5.0.0
        Product version:  19.5.0.0
        File flags:       0 (Mask 3F)
        File OS:          4 Unknown Win32
        File type:        2.0 Dll
        File date:        00000000.00000000
        Translations:     0409.04b0
        Information from resource tables:
                CompanyName:      Accusoft Corporation
                ProductName:      Accusoft ImageGear
                InternalName:     igcore19d.dll
                OriginalFilename: igcore19d.dll
                ProductVersion:   19.5.0.0
                FileVersion:      19.5.0.0
                FileDescription:  Accusoft ImageGear CORE DLL
                LegalCopyright:   Copyright 1996-2019 Accusoft Corporation. All rights reserved.
                LegalTrademarks:  ImageGearÆ and AccusoftÆ are registered trademarks of Accusoft Corporation
```

Timeline

2020-01-27 - Vendor Disclosure

2020-02-10 - Public Release

CREDIT

Discovered by Emmanuel Tacheau and a member of Cisco Talos.