Talos Vulnerability Report

TALOS-2020-0993

Accusoft ImageGear JPEG jpegread precision code execution vulnerability

FEBRUARY 10, 2020

CVE NUMBER

CVE-2020-6069

Summary

An exploitable out-of-bounds write vulnerability exists in the igcore19d.dll JPEG jpegread precision 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:

```
(397c.4b88): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=ffff887f ebx=00000000 ecx=ffffe380 edx=0c067004 esi=00000fff edi=ffff887f
eip=5b9d4669 esp=0100f084 ebp=0100f09c 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+0xc92d9:
5b9d4669 66895afc        mov     word ptr [edx-4],bx      ds:002b:0c067000=????
```

Checking the buffer capacity pointed to by edx, we can see:

```
0:000> !heap -p -a 0c067000
        address 0c067000 found in
        _DPH_HEAP_ROOT @ 3811000
        in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                                                              c060bfc:          c066d48              2b4 -          c066000
2000
                 unknown!fillpattern
        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
        5b9d24ad igCore19d!IG_mpi_page_set+0x000c711d
        5b9befc5 igCore19d!IG_mpi_page_set+0x000b3c35
        5b9d6115 igCore19d!IG_mpi_page_set+0x000cad85
        5b9d5f3d igCore19d!IG_mpi_page_set+0x000cabad
        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
```

The calculated address `0c067000` is outside of the allocation for this buffer space.

Going back to the allocation, we land in the following place:

```
Line 1  int __stdcall sub_5B9D2100(int a1, void *a2, int a3, int a4)
Line 2  {
Line 3          (...)
Line 4    mem_size = getMemSize((_DWORD *)v4->dword84);
Line 5    v4->dword8C = mem_size;
Line 6    v31 = AF_memm_alloc(v4->dword80, mem_size + 24, (int)"..\\..\\..\\..\\Common\\Formats\\jpgread.c", 3348);
Line 7    (...)
Line 8  }
```

The `getMemSize` function at `line 4` calculates the returned `mem_size` as follows:

```
precision < 0x8  = 0x8
precision < 0x10 = 0x10
precision < 0x20 = 0x20
        exception

(((value_base_on_precision * nr_comp * x_image) + 0x1F) >> 3 ) & 0FFFFFFFCh
0x29c = (((8 * 3 * 0xde) + 0x1F ) >> 3) & 0x0FFFFFFC
```

Where the variable/fields are located in the file at offsets:

```
        0xA2 : precision : SOFx[0]->precision = 7
        0xA5 : SOFx[0]->x_image = 0xDE
        0xA7 : SOFx[0]->nr_comp = 3
```

Next, a fixed value of `24` is added to returned value and we end up with `0x2b4` allocated bytes.

Let use take a look now at the vulnerable function:

```
Line 1  signed __int16 *__cdecl sub_5B9D45C0(int a1, int loop_limit, signed __int16 *a3, signed __int16 *a4, signed __int16 *a5, int a6,
struct_a7 *a7, int a8)
Line 2  {
Line 3    signed __int16 *result; // eax
Line 4    int v9; // edi
Line 5    int v10; // esi
Line 6    int v11; // kr00_4
Line 7    int v12; // ecx
Line 8    int v13; // eax
Line 9    int v14; // ebx
Line 10   bool v15; // zf
Line 11   int v16; // [esp+0h] [ebp-Ch]
Line 12   int v17; // [esp+4h] [ebp-8h]
Line 13   int v18; // [esp+8h] [ebp-4h]
Line 14   _WORD *buffer; // [esp+28h] [ebp+1Ch]
Line 15
Line 16   result = 0;
Line 17   v16 = 0;
Line 18   v17 = 0;
Line 19   v18 = 0;
Line 20   if ( loop_limit > 0 )
Line 21   {
Line 22         buffer = (_WORD *)(a6 + 4);
Line 23         result = a5;
Line 24         do
Line 25         {
Line 26           v9 = *a3 + 2048;
Line 27           v10 = *a4;
Line 28           v11 = 5742 * *result;
Line 29           v12 = v9 - (2925 * *result + 1410 * v10) / 4096;
Line 30           v13 = v9 + 7258 * v10 / 4096;
Line 31           v14 = v9 + v11 / 4096;
Line 32           if ( v9 + v11 / 4096 >= 0 )
Line 33           {
Line 34                 if ( v14 > 4095 )
Line 35                   LOWORD(v14) = 4095;
Line 36           }
Line 37           else
Line 38           {
Line 39                 LOWORD(v14) = 0;
Line 40           }
Line 41           *(buffer - 2) = v14;
Line 42           if ( v12 >= 0 )
Line 43           {
Line 44                 if ( v12 > 4095 )
Line 45                   LOWORD(v12) = 4095;
Line 46           }
Line 47           else
Line 48           {
Line 49                 LOWORD(v12) = 0;
Line 50           }
Line 51           *(buffer - 1) = v12;
Line 52           if ( v13 >= 0 )
Line 53           {
Line 54                 if ( v13 > 4095 )
Line 55                   LOWORD(v13) = 4095;
Line 56           }
Line 57           else
Line 58           {
Line 59                 LOWORD(v13) = 0;
Line 60           }
Line 61           *buffer = v13;
Line 62           v18 += a7->dword34;
Line 63           if ( v18 == a8 )
Line 64           {
Line 65                 ++a3;
Line 66                 v18 = 0;
Line 67           }
Line 68           v17 += a7->dword84;
Line 69           if ( v17 == a8 )
Line 70           {
Line 71                 ++a4;
Line 72                 v17 = 0;
Line 73           }
Line 74           v16 += a7->dwordD4;
Line 75           if ( v16 == a8 )
Line 76           {
Line 77                 v16 = 0;
Line 78                 result = a5 + 1;
Line 79                 ++a5;
Line 80           }
Line 81           else
Line 82           {
Line 83                 result = a5;
Line 84           }
Line 85           v15 = loop_limit-- == 1;
Line 86           buffer += 3;
Line 87         }
Line 88         while ( !v15 );
Line 89   }
Line 90   return result;
Line 91 }
```

The `loop_limit` variable is equal to `0xA5 : SOFx[0]->x_image = 0xDE`. Each loop cycle, the `buffer` pointer is increased by `3 * WORD = 6` at `line 86`.

So for buffer with capacity `0x2B4` bytes this loop is able to execute a maximum of `0x2B4 / 6 = 00000073` times but `loop_limit` equals `0xDE` and is decrement by 1 during each cycle, which leads to an out-of-bounds write and memory corruption.

As we can see an attacker controls all presented variables just by proper file content manipulation.

Increasing the loop count via the `loop_limit` variable an attacker can cause an out-of-bounds write leading to memory corruption, which can result in remote code execution.

Crash Information

```
(5e3c.4934): Access violation - code c0000005 (first/second chance not available)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
Time Travel Position: 17BF44:0
eax=ffff887f ebx=00000000 ecx=ffffe380 edx=198ed004 esi=00000fff edi=ffff887f
eip=5b9d4669 esp=00daf180 ebp=00daf198 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
igCore19d!IG_mpi_page_set+0xc92d9:
5b9d4669 66895afc        mov     word ptr [edx-4],bx      ds:002b:198ed000=????

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.
