Talos Vulnerability Report

TALOS-2020-0992

Accusoft ImageGear PNG pngread width code execution vulnerability

JANUARY 27, 2020

CVE NUMBER

CVE-2020-6068

Summary

An exploitable out-of-bounds write vulnerability exists in the igcore19d.dll PNG pngread parser of the Accusoft ImageGear 19.5.0 library. A specially crafted PNG 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, DPNGM, PDF, Microsoft Office and others.

There is a vulnerability in the PNG raster image parser. A specially crafted PNG file can lead to an out-of-bounds write resulting in remote code execution.

If we try to load a malformed PNG 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=00000004 ebx=00000002 ecx=0bbdfff8 edx=000001f4 esi=00000001 edi=fffffffe
eip=5b914006 esp=00afd480 ebp=00afd494 iopl=0         nv up ei pl nz ac po cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010213
igCore19d!IG_mpi_page_set+0x8c76:
5b914006 897cb104        mov     dword ptr [ecx+esi*4+4],edi ds:002b:0bbe0000=????????
```

Checking the capacity of the buffer pointed by `ecx`:

```
0:000> !heap -p -a 0x0bbdfff8
        address 0bbdfff8 found in
        _DPH_HEAP_ROOT @ cd1000
        in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                                                        ba23374:          bbdfff8                8 -          bbdf000
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
        5b91c428 igCore19d!IG_mpi_page_set+0x00011098
        5b8f84de igCore19d!IG_cpm_profiles_reset+0x0000dfae
        5b9f0c04 igCore19d!IG_mpi_page_set+0x000e5874
        5b9ee32c igCore19d!IG_mpi_page_set+0x000e2f9c
        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
```

We can see that it is equal to `8 bytes` and that the calculated address in the crashing line points out of its range.

Further analysis revealed that space for that buffer is allocated in the following place:

```
Line 1  int __thiscall sub_5DCCC3D0(char *this, int a2, char *a3)
Line 2  {
Line 3    int v3; // ecx
Line 4    char *_this; // ST04_4
Line 5    size_t mem_size; // ST04_4
Line 6    int v6; // eax
Line 7    char *v8; // [esp+0h] [ebp-4h]
Line 8
Line 9    v8 = this;
Line 10   if ( !sub_5DC6C860(a3, (int)&v8) )
Line 11   {
Line 12         **(_DWORD **)(a2 + 4) = v8;
Line 13         if ( sub_5DC684E0((int)a3) == 1 )
Line 14         {
Line 15           _this = v8;
Line 16           *(_DWORD *)(a2 + 100) = 1;
Line 17           mem_size = 4 * getWidth(_this) + 16;
Line 18           v6 = sub_5DC6CFD0(v8);
Line 19           *(_DWORD *)(a2 + 104) = AF_memm_alloc(v6, mem_size, (int)"..\\..\\..\\..\\Common\\Io\\dfltldcb.c", 162);
Line 20         }
Line 21   }
Line 22   return sub_5DC8AA00(v3);
Line 23 }
```

At `line 17` we can see a `mem_size` calculation which is made based on value returned from `getWidth` function. This function returns [UINT]IDHR->Width value directly from file.

In our case its value is equal `0xfffffffe` (offset : 0x10). Doing the necessary math we end up with `mem_size` equal `0x00000008 = 0xfffffffe * 4 + 16`.

With that in mind, let us return to vulnerable function:

```
Line 1  int __cdecl sub_5DCC3F20(unsigned __int8 *a1, _DWORD *inBuffer, int inStore_Value)
Line 2  {
Line 3    int store_value; // edi
Line 4    int index; // esi
Line 5    int v12; // ebx
Line 6    _DWORD *buffer; // ecx
Line 7    int result; // eax
Line 8    int v15; // [esp+Ch] [ebp-8h]
Line 9    signed int __i; // [esp+10h] [ebp-4h]
Line 10
Line 11   store_value = inStore_Value;
Line 12   v4 = 0;
Line 13   for ( __i = 1; ; __i += 2 )
Line 14   {
Line 15         (...)
Line 16   }
Line 17   buffer = inBuffer;
Line 18   result = index + 3;
Line 19   inBuffer[index] = store_value;
Line 20   buffer[index + 1] = store_value;              // !!! VULN !!!
Line 21   buffer[index + 2] = store_value;
Line 22   *buffer = index + 3;
Line 23   return result;
Line 24 }
```

In the above pseudo-code we can see that there is no check whether `buffer` size is bigger-equal than 12 bytes and because of that at `line 20` an out-of-bounds write occurs. As we can see, an attacker controls all presented variables just by proper file content manipulation.

Increasing the loop count via the `v81` variable, an attacker can cause an out-of-bounds write leading to memory corruption which can result in remote code execution.

Crash Information

```
eax=00000004 ebx=00000002 ecx=27095ff8 edx=000001f4 esi=00000001 edi=fffffffe
eip=5dcc4006 esp=0100d2b0 ebp=0100d2c4 iopl=0         nv up ei pl nz ac po cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000213
igCore19d!IG_mpi_page_set+0x8c76:
5dcc4006 897cb104        mov     dword ptr [ecx+esi*4+4],edi ds:002b:27096000=????????
0:000> !analyze -v
*******************************************************************************
*                                                                             *
*                        Exception Analysis                                   *
*                                                                             *
*******************************************************************************


KEY_VALUES_STRING: 1

        Key  : AV.Fault
        Value: Read

        Key  : Analysis.CPU.Sec
        Value: 0

        Key  : Analysis.DebugAnalysisProvider.CPP
        Value: Create: 8007007e on DESKTOP-E4N8506

        Key  : Analysis.DebugData
        Value: CreateObject

        Key  : Analysis.DebugModel
        Value: CreateObject

        Key  : Analysis.Elapsed.Sec
        Value: 0

        Key  : Analysis.Memory.CommitPeak.Mb
        Value: 433

        Key  : Analysis.System
        Value: CreateObject

        Key  : Timeline.OS.Boot.DeltaSec
        Value: 541573


ADDITIONAL_XML: 1

APPLICATION_VERIFIER_LOADED: 1

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 5dcc4006 (igCore19d!IG_mpi_page_set+0x00008c76)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 00000000
   Parameter[1]: 27096000
Attempt to read from address 27096000

FAULTING_THREAD:  000043dc

PROCESS_NAME:  igFuzzer.exe

READ_ADDRESS:  27096000

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:  00000000

EXCEPTION_PARAMETER2:  27096000

STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
0100d2c4 5dcc48b1 26e75ff8 27095ff8 fffffffe igCore19d!IG_mpi_page_set+0x8c76
0100d2d8 5dccc969 26e75ff8 27095ff8 fffffffe igCore19d!IG_mpi_page_set+0x9521
0100d2f8 5dca94fe 0100f674 26e75ff8 00000000 igCore19d!IG_mpi_page_set+0x115d9
0100d318 5dd9e0c9 0100f500 26e75ff8 00000000 igCore19d!IG_cpm_profiles_reset+0xefce
0100d334 5dd9fe7f 257e9fa8 0100f500 26e75ff8 igCore19d!IG_mpi_page_set+0xe2d39
0100efb0 5dda0c74 0100f500 1000001b 2469dfe8 igCore19d!IG_mpi_page_set+0xe4aef
0100efe4 5dd9e32c 0100f500 1000001b 2469dfe8 igCore19d!IG_mpi_page_set+0xe58e4
0100f478 5dc907c9 0100f500 2469dfe8 00000001 igCore19d!IG_mpi_page_set+0xe2f9c
0100f4b0 5dccfb97 00000000 2469dfe8 0100f500 igCore19d!IG_image_savelist_get+0xb29
0100f72c 5dccf4f9 00000000 20f04f98 00000001 igCore19d!IG_mpi_page_set+0x14807
0100f74c 5dc66007 00000000 20f04f98 00000001 igCore19d!IG_mpi_page_set+0x14169
0100f76c 00ef59ac 20f04f98 0100f858 0100f87c igCore19d!IG_load_file+0x47
0100f86c 00ef61a7 20f04f98 0100f9a0 00000021 igFuzzer+0x159ac
0100fa38 00ef6cbe 00000005 1a420f40 1a287f40 igFuzzer+0x161a7
0100fa4c 00ef6b27 997563fa 00ef15e1 00ef15e1 igFuzzer+0x16cbe
0100faa8 00ef69bd 0100fab8 00ef6d38 0100fac8 igFuzzer+0x16b27
0100fab0 00ef6d38 0100fac8 74f56359 00d55000 igFuzzer+0x169bd
0100fab8 74f56359 00d55000 74f56340 0100fb24 igFuzzer+0x16d38
0100fac8 772f7b74 00d55000 aaeb7cb0 00000000 KERNEL32!BaseThreadInitThunk+0x19
0100fb24 772f7b44 ffffffff 77318f13 00000000 ntdll!__RtlUserThreadStart+0x2f
0100fb34 00000000 00ef15e1 00d55000 00000000 ntdll!_RtlUserThreadStart+0x1b


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  igCore19d!IG_mpi_page_set+8c76

MODULE_NAME: igCore19d

IMAGE_NAME:  igCore19d.dll

FAILURE_BUCKET_ID:  INVALID_POINTER_READ_AVRF_c0000005_igCore19d.dll!IG_mpi_page_set

OSPLATFORM_TYPE:  x86

OSNAME:  Windows 8

FAILURE_ID_HASH:  {bfd6b5ab-5824-8327-06e6-1c2f38a120f0}

Followup:     MachineOwner
---------
```

```
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.
