Talos Vulnerability Report

TALOS-2020-1196

Accusoft ImageGear PSD Header processing out-of-bounds write vulnerability

FEBRUARY 9, 2021

CVE NUMBER

CVE-2020-13585

Summary

An out-of-bounds write vulnerability exists in the PSD Header processing functionality of Accusoft ImageGear 19.8. A specially crafted malformed file can lead to code execution. An attacker can provide a malicious file to trigger this vulnerability.

Tested Versions

Accusoft ImageGear 19.8

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-131 - Incorrect Calculation of Buffer Size

Details

The ImageGear library is a document-imaging developer toolkit that offers image conversion, creation, editing, annotation and more. It supports more than 100 formats such as DICOM, PDF, Microsoft Office and others.

There is a vulnerability in the `psd_header_processing` function, due to a buffer overflow caused by a missing check of the allocation size.
A specially crafted PSD file can lead to an out-of-bounds write which can result in a memory corruption.

Trying to load a malformed PSD file, we end up in the following situation:

```
(12748.f48c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0d7a9f66 ebx=00000003 ecx=08641000 edx=00000002 esi=0a184f88 edi=00000002
eip=5e36ec9b esp=0019f618 ebp=0019f66c iopl=0         nv up ei pl nz ac pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010217
igCore19d!IG_mpi_page_set+0xf2f4b:
5e36ec9b 8801            mov     byte ptr [ecx],al          ds:002b:08641000=??
```

When we look at the `ecx` memory allocation we can see the buffer allocated is very small, only 1 byte:

```
0:000> !heap -p -a ecx
address 08641000 found in
_DPH_HEAP_ROOT @ 2cc1000
in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                              8522924:          8640ff8                1 -          8640000            2000
5e56ab70 verifier!AVrfDebugPageHeapAllocate+0x00000240
7701909b ntdll!RtlDebugAllocateHeap+0x00000039
76f6bbad ntdll!RtlpAllocateHeap+0x000000ed
76f6b0cf ntdll!RtlpAllocateHeapInternal+0x0000022f
76f6ae8e ntdll!RtlAllocateHeap+0x0000003e
5e13dcff MSVCR110!malloc+0x00000049
5e2761de igCore19d!AF_memm_alloc+0x0000001e
5e36e35c igCore19d!IG_mpi_page_set+0x000f260c
5e36d67a igCore19d!IG_mpi_page_set+0x000f192a
5e36d3c4 igCore19d!IG_mpi_page_set+0x000f1674
5e36bbd8 igCore19d!IG_mpi_page_set+0x000efe88
5e36b54a igCore19d!IG_mpi_page_set+0x000ef7fa
5e2510d9 igCore19d!IG_image_savelist_get+0x00000b29
5e290557 igCore19d!IG_mpi_page_set+0x00014807
5e28feb9 igCore19d!IG_mpi_page_set+0x00014169
5e225777 igCore19d!IG_load_file+0x00000047
004020f4 Fuzzme!fuzzme+0x000000d4
00402524 Fuzzme!fuzzme+0x00000504
00407aaa Fuzzme!fuzzme+0x00005a8a
76146359 KERNEL32!BaseThreadInitThunk+0x00000019
76f97c24 ntdll!__RtlUserThreadStart+0x0000002f
76f97bf4 ntdll!_RtlUserThreadStart+0x0000001b
```

The content of the buffer is the following

```
0:000> db 8640ff8
08640ff8  de ad 66 de ad 66 de ad-?? ?? ?? ?? ?? ?? ?? ??  ..f..f..????????
```

The pattern of the three bytes 0xde, 0xad and 0x66 are coming directly from the file. But we need to understand how and why is that happening.

The crash is happening in the following pseudo code of the function `psd_header_processing`:

```
LINE 1    void psd_header_processing
LINE 2                   (mys_table_function *mys_table_func_obj,int param_2,uint kind_of_heap,int param_4,
LINE 3                   int *param_5,IGDIBStd *IGDIBStd_obj)
LINE 4    {
LINE 5      int *piVar1;
LINE 6      short sVar2;
LINE 7      size_t sVar3;
LINE 8      byte bVar4;
LINE 9      ushort uVar5;
LINE 10     uint uVar6;
LINE 11     dword dVar7;
LINE 12     dword _width_from_image;
LINE 13     byte *_oobw_buffer;
LINE 14     byte *_data_buffer;
LINE 15     uint *_buffer_from_file;
LINE 16     size_t sVar8;
LINE 17     int *piVar9;
LINE 18     BYTE *dst;
LINE 19     undefined4 *mem_to_free;
LINE 20     int iVar10;
LINE 21     undefined2 *puVar11;
LINE 22     undefined4 *puVar12;
LINE 23     int iVar13;
LINE 24     uint uVar14;
LINE 25     short sVar15;
LINE 26     int index;
LINE 27     ushort *puVar16;
LINE 28     size_t size;
LINE 29     short *psVar17;
LINE 30     byte *dest_buffer;
LINE 31     ushort *puVar18;
LINE 32     uint uVar19;
LINE 33     int iVar20;
LINE 34     int iVar21;
LINE 35     uint _alloc_size;
LINE 36     void *_num_integer_to_read;
LINE 37     size_t size_00;
LINE 38     uint *_store_value_from_file;
LINE 39     int iVar22;
LINE 40     uint **ppuVar23;
LINE 41     uint uVar24;
LINE 42     size_t *psVar25;
LINE 43     byte *pbVar26;
LINE 44     undefined auVar27 [12];
LINE 45     undefined auVar28 [16];
LINE 46     undefined4 uVar29;
LINE 47     int local_3c;
LINE 48     int local_38;
LINE 49     uint _long_value_read;
LINE 50     short *_short_value_read;
LINE 51     uint local_2c;
LINE 52     dword _length_from_image;
LINE 53     uint local_24;
LINE 54     void *_index_loop;
LINE 55     int local_1c;
LINE 56     uint _some_max_value_from_param;
LINE 57
LINE 58     local_38 = 0;
LINE 59     local_2c = 0;
LINE 60     iVar21 = 1;
LINE 61     uVar6 = getColorSpace((HIGDIBINFO)IGDIBStd_obj);
LINE 62     local_24 = FUN_1002dde0(uVar6);
LINE 63     if (*(short *)(param_4 + 0x18) == 0x10) {
LINE 64       iVar21 = 2;
LINE 65     }
LINE 66     sVar2 = *(short *)(param_4 + 0x1a);
LINE 67     if (param_5 == (int *)0x0) {
LINE 68       _some_max_value_from_param = (uint)*(ushort *)(param_4 + 0xc);
LINE 69       if ((int)local_24 < (int)_some_max_value_from_param) {
LINE 70         local_2c = (uint)(*(int *)(param_4 + 0x98) != 0);
LINE 71       }
LINE 72     }
LINE 73     else {
LINE 74       _some_max_value_from_param = (uint)*(ushort *)(param_5 + 4);
LINE 75       index = 0;
LINE 76       iVar20 = 0;
LINE 77       while (index < (int)_some_max_value_from_param) {
LINE 78         if (*(short *)(param_5[5] + 8 + iVar20) == -1) goto LAB_1015e2f1;
LINE 79         index = index + 1;
LINE 80         iVar20 = iVar20 + 0x10;
LINE 81       }
LINE 82       if ((*(int *)(param_2 + 0x18) != 0) && (index = 0, *(ushort *)(param_5 + 4) != 0)) {
LINE 83         puVar16 = (ushort *)(param_5[5] + 8);
LINE 84         do {
LINE 85           if ((int)local_24 < (int)(uint)*puVar16) goto LAB_1015e2f1;
LINE 86           index = index + 1;
LINE 87           puVar16 = puVar16 + 8;
LINE 88         } while (index < (int)_some_max_value_from_param);
LINE 89       }
LINE 90     }
LINE 91   LAB_1015e316:
LINE 92     dVar7 = getWidth((HIGDIBINFO)IGDIBStd_obj);
LINE 93     _length_from_image = getLength((HIGDIBINFO)IGDIBStd_obj);
LINE 94     _width_from_image = getWidth((HIGDIBINFO)IGDIBStd_obj);
LINE 95     _bit_depth = get_bit_depth((HIGDIBINFO)IGDIBStd_obj);
LINE 96     _alloc_size = (int)(_width_from_image * _bit_depth + 0x1f) >> 3 & 0xfffffffc;                              [5]
LINE 97     _oobw_buffer = AF_memm_alloc(kind_of_heap,_alloc_size,                                                     [4]
LINE 98                                  (dword)"..\\..\\..\\..\\Common\\Formats\\psdread.c");
LINE 99     _data_buffer = AF_memm_alloc(kind_of_heap,_some_max_value_from_param * 0x28,
LINE 100                                 (dword)"..\\..\\..\\..\\Common\\Formats\\psdread.c"
LINE 101    [...]
LINE 102
LINE 103            else {
LINE 104              _short_value_read = (short *)0x0;
LINE 105              ___oobw_buffer = _oobw_buffer;                                                                   [3]
LINE 106              if (0 < (int)dVar7) {
LINE 107                do {
LINE 108                  if (param_5 == (int *)0x0) {
LINE 109                    if (local_2c == 0) {
LINE 110                      if (iVar20 == 1) {
LINE 111                        index_loop = 0;
LINE 112                        if (num_channel_image != 0) {
LINE 113                          do {                                                                             [2]
LINE 114                            if (mem_to_free[index_loop] == -1) {
LINE 115                              *___oobw_buffer = 0;
LINE 116                            }
LINE 117                            else {
```

```
LINE 118                              *___oobw_buffer =                                                                  [1]
LINE 119                                   *(byte *)((int)_short_value_read +
LINE 120                                             *(int *)(_data_buffer +
LINE 121                                                     mem_to_free[index_loop] * 0x28 + 0x1c));
LINE 122                            }
LINE 123                            index_loop = index_loop + 1;
LINE 124                            ___oobw_buffer = ___oobw_buffer + 1;
LINE 125                          } while (index_loop < (int)num_channel_image);                                       [2]
LINE 126                        }
LINE 127                      }
LINE 128    [...]
LINE 129  }
```

The crash is happening at [1]. We can see the write into the buffer is happening through a do-while loop controlled by the `num_channel_image` variable [2], taken directly from the file. Going backward we can see the `___oobw_buffer`, previously assigned from `_oobw_buffer` [3], is allocated through a call to `AF_memm_alloc` [4] with a size of `_alloc_size` [5].
The size of the buffer is directly computed from a value issued from the file and the issue is happening when `_bit_depth` is null. We can see that `igCore19d!AF_memm_alloc` is a wrapper for `malloc`:

```
LINE 132  BYTE * AF_memm_alloc(uint kind_of_heap,size_t size,dword param3)
LINE 133  {
LINE 134    LPCRITICAL_SECTION *pp_Var1;
LINE 135    uint *puVar2;
LINE 136    byte *mem_alloc;
LINE 137    uint *puVar3;
LINE 138    struct_a8 *buffer_size_a8;
LINE 139    uint uVar4;
LINE 140    struct_b4_size *buffer_b4_size;
LINE 141
LINE 142    wrapper_EnterCriticalSection(Count_CriticalSectionUse[0x5a1]);
LINE 143    mem_alloc = (byte *)malloc(size);                                                                           [6]
LINE 144    if (mem_alloc == (byte *)0x0) {
LINE 145      wrapper_LeaveCriticalSection(Count_CriticalSectionUse[0x5a1]);
LINE 146      return (BYTE *)0x0;
LINE 147    }
LINE 148    [...]
LINE 149        return mem_alloc;
LINE 150      }
LINE 151    [...]
LINE 152  }
```

The pseudo code for `igCore19d!AF_memm_alloc` does not check for a null `size` parameter [6] and thus is returning what `malloc` returns. The issue is that `malloc(0)` returns a non-null value, thus the program assumes the allocation succeeded, however this is not true. The buffer allocated in this case is a very small chunk of 1 byte in size in a Windows environment. So if `num_channel_image` is bigger than or equal to `3`, the do-while loop [2] will eventually write out-of-bounds in the heap, possibly leading to arbitrary code execution.

Crash Information

```
0:000> !analyze -v
*******************************************************************************
*                                                                             *
*                        Exception Analysis                                   *
*                                                                             *
*******************************************************************************


KEY_VALUES_STRING: 1

    Key  : AV.Fault
    Value: Write

    Key  : Analysis.CPU.mSec
    Value: 2561

    Key  : Analysis.DebugAnalysisProvider.CPP
    Value: Create: 8007007e on DESKTOP-4DAOCFH

    Key  : Analysis.DebugData
    Value: CreateObject

    Key  : Analysis.DebugModel
    Value: CreateObject

    Key  : Analysis.Elapsed.mSec
    Value: 10487

    Key  : Analysis.Memory.CommitPeak.Mb
    Value: 167

    Key  : Analysis.System
    Value: CreateObject

    Key  : Timeline.OS.Boot.DeltaSec
    Value: 2012569

    Key  : Timeline.Process.Start.DeltaSec
    Value: 1560

    Key  : WER.OS.Branch
    Value: 19h1_release

    Key  : WER.OS.Timestamp
    Value: 2019-03-18T12:02:00Z

    Key  : WER.OS.Version
    Value: 10.0.18362.1

    Key  : WER.Process.Version
    Value: 1.0.0.2


ADDITIONAL_XML: 1

OS_BUILD_LAYERS: 1

NTGLOBALFLAG:  2000000

APPLICATION_VERIFIER_FLAGS:  0

APPLICATION_VERIFIER_LOADED: 1

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 5e36ec9b (igCore19d!IG_mpi_page_set+0x000f2f4b)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 00000001
   Parameter[1]: 08641000
Attempt to write to address 08641000

FAULTING_THREAD:  0000f48c

PROCESS_NAME:  Fuzzme.exe

WRITE_ADDRESS:  08641000

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:  08641000

STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
0019f66c 5e36d67a     0019fc04 0a184f88 1000002a igCore19d!IG_mpi_page_set+0xf2f4b
0019f698 5e36d3c4     0019fc04 0a274fe0 1000002a igCore19d!IG_mpi_page_set+0xf192a
0019f6ec 5e36bbd8     0019fc04 0a274fe0 1000002a igCore19d!IG_mpi_page_set+0xf1674
0019f724 5e36b54a     0019fc04 0019f74c 0019f774 igCore19d!IG_mpi_page_set+0xefe88
0019fb7c 5e2510d9     0019fc04 0a274fe0 00000001 igCore19d!IG_mpi_page_set+0xef7fa
0019fbb4 5e290557     00000000 0a274fe0 0019fc04 igCore19d!IG_image_savelist_get+0xb29
0019fe30 5e28feb9     00000000 09616fa0 00000001 igCore19d!IG_mpi_page_set+0x14807
0019fe50 5e225777     00000000 09616fa0 00000001 igCore19d!IG_mpi_page_set+0x14169
0019fe70 004020f4     09616fa0 0019fe84 0951cf28 igCore19d!IG_load_file+0x47
0019fe94 00402524     09616fa0 09614fe0 00000021 Fuzzme!fuzzme+0xd4
0019ff28 00407aaa     00000005 0951cf28 09523f30 Fuzzme!fuzzme+0x504
0019ff70 76146359     00276000 76146340 0019ffdc Fuzzme!fuzzme+0x5a8a
0019ff80 76f97c24     00276000 495462c0 00000000 KERNEL32!BaseThreadInitThunk+0x19
0019ffdc 76f97bf4     ffffffff 76fb8fe3 00000000 ntdll!__RtlUserThreadStart+0x2f
0019ffec 00000000     00407b32 00276000 00000000 ntdll!_RtlUserThreadStart+0x1b


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  igCore19d!IG_mpi_page_set+f2f4b

MODULE_NAME: igCore19d

IMAGE_NAME:  igCore19d.dll
```

```
FAILURE_BUCKET_ID:  INVALID_POINTER_WRITE_AVRF_c0000005_igCore19d.dll!IG_mpi_page_set

OS_VERSION:  10.0.18362.1

BUILDLAB_STR:  19h1_release

OSPLATFORM_TYPE:  x86

OSNAME:  Windows 10

IMAGE_VERSION:  19.8.0.0

FAILURE_ID_HASH:  {39ff52ad-9054-81fd-3e4d-ef5d82e4b2c1}

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

Timeline

2020-11-17 - Vendor Disclosure
2021-02-05 - Vendor Patche
2021-02-09 - Public Release

CREDIT

Discovered by Emmanuel Tacheau of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2020-1182

NEXT REPORT TALOS-2020-1223