Talos Vulnerability Report

TALOS-2022-1526

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

JULY 18, 2022

CVE NUMBER

CVE-2022-29465

Summary

An out-of-bounds write vulnerability exists in the PSD Header processing memory allocation functionality of Accusoft ImageGear 20.0. A specially-crafted malformed file can lead to memory corruption. An attacker can provide a malicious file to trigger this vulnerability.

Tested Versions

Accusoft ImageGear 20.0

Product URLs

ImageGear - https://www.accusoft.com/products/imagegear-collection/

CVSSv3 Score

8.1 - CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H

CWE

CWE-119 - Improper Restriction of Operations within the Bounds of a Memory Buffer

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 `allocation_function_mem` function, due to a buffer overflow caused by a missing check for memory bounds within a heap buffer.
A specially-crafted PSD file can lead to an out-of-bounds write, which can result in memory corruption.

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

```
(26ec.2464): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0b905fc0 ebx=00000009 ecx=0c35c000 edx=00000008 esi=0c19ee98 edi=00000000
eip=6d7cf76b esp=0019f650 ebp=0019f6a4 iopl=0         nv up ei pl nz ac pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010217
igcore20d!IG_mpi_page_set+0xf35db:
6d7cf76b 8801            mov     byte ptr [ecx],al          ds:002b:0c35c000=??
```

When we look at the `ecx` memory allocation we can see the buffer allocated is very small, only 4 bytes:

```
0:000> !heap -p -a ecx
    address 0c35c000 found in
    _DPH_HEAP_ROOT @ 2cb1000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -
VirtAddr         VirtSize)
                                 c360b94:          c35bff8                4 -
c35b000             2000
    6db3a8b0 verifier!AVrfDebugPageHeapAllocate+0x00000240
    77c0f10e ntdll!RtlDebugAllocateHeap+0x00000039
    77b770f0 ntdll!RtlpAllocateHeap+0x000000f0
    77b76e3c ntdll!RtlpAllocateHeapInternal+0x0000104c
    77b75dde ntdll!RtlAllocateHeap+0x0000003e
    6d50daff MSVCR110!malloc+0x00000049
    6d6d663e igcore20d!AF_memm_alloc+0x0000001e
    6d7cee2c igcore20d!IG_mpi_page_set+0x000f2c9c
    6d7ce14a igcore20d!IG_mpi_page_set+0x000f1fba
    6d7cde94 igcore20d!IG_mpi_page_set+0x000f1d04
    6d7cc5d8 igcore20d!IG_mpi_page_set+0x000f0448
    6d7cbf4a igcore20d!IG_mpi_page_set+0x000efdba
    6d6b1399 igcore20d!IG_image_savelist_get+0x00000b29
    6d6f09e7 igcore20d!IG_mpi_page_set+0x00014857
    6d6f0349 igcore20d!IG_mpi_page_set+0x000141b9
    6d685777 igcore20d!IG_load_file+0x00000047
*** WARNING: Unable to verify checksum for Fuzzme.exe
    00402239 Fuzzme!fuzzme+0x00000019
    00402544 Fuzzme!fuzzme+0x00000324
    004062a0 Fuzzme!fuzzme+0x00004080
    762ffa29 KERNEL32!BaseThreadInitThunk+0x00000019
    77b97a9e ntdll!__RtlUserThreadStart+0x0000002f
    77b97a6e ntdll!_RtlUserThreadStart+0x0000001b
```

The issue is happening in the function `allocation_function_mem`, represented by the following pseudo-code (this is a very large function where a lot of code was removed to make it simpler to read):

```
LINE1    AT_ERRCOUNT
LINE2        allocation_function_mem
LINE3    (mys_table_function *mys_table_function,int param_2,uint heap,psd_header
*psd_header,
LINE4     int *param_5,HIGDIBINFO higdibinfo)
LINE5
LINE6    {
[...]
LINE69       bcheck_format = False;
LINE70       bitsperchanneleq16_is2_or1 = 1;
LINE71       enumIGColorSpaceIDs = DIB_colorspace_get(higdibinfo);
LINE72       color_count = iIG_util_colorspace_color_count_get(enumIGColorSpaceIDs);
LINE73       if (psd_header->bits_per_channel == 16) {
LINE74           bitsperchanneleq16_is2_or1 = 2;
LINE75       }
LINE76       _color_mode = psd_header->color_mode;
LINE77       if (param_5 == (int *)0x0) {
LINE78           _num_channel_image = (uint)psd_header->num_channel_image;
LINE79           if ((int)color_count < (int)_num_channel_image) {
LINE80               bcheck_format = (uint)(psd_header->field53_0x98 != 0);
LINE81           }
LINE82       }
LINE83       else {
[...]
LINE99       }
LINE100  start_alloc:
LINE101      __num_channel_image = _num_channel_image;
LINE102      width = DIB_width_get(higdibinfo);
LINE103      _height_from_dib = DIB_height_get(higdibinfo);
LINE104      _witdh = DIB_width_get(higdibinfo);
LINE105      _product_of_numchannel_bits_per_channel =
IGDIBStd::DIB_bit_depth_get(higdibinfo);
LINE106      _size_buffer_alloc = (int)(_witdh *
_product_of_numchannel_bits_per_channel + 0x1f) >> 3 & 0xfffffffc;
LINE108      psd_buffer_oobw = (undefined4 *)AF_memm_alloc(heap,_size_buffer_alloc);
LINE109      channel_image_data = (channel_image_struct
*)AF_memm_alloc(heap,__num_channel_image * 0x28);
LINE110      /* Check for allocation buffer */
LINE111      if ((psd_buffer_oobw == (undefined4 *)0x0) || (channel_image_data ==
(channel_image_struct *)0x0))
LINE112      {
LINE113          heap = 0;
LINE114          _index_channel_loop = 0x9f8;
LINE115  ErrorPsdRead:
LINE116
AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\psdread.c",_index_channel_loop,-
1000,0,heap,
LINE117                  0,(LPCHAR)0x0);
LINE118      }
LINE119      else {
LINE120          /* Allocation buffer succeeded */
LINE121          OS_memset(psd_buffer_oobw,0,_size_buffer_alloc);
LINE122          OS_memset(channel_image_data,0,__num_channel_image * 0x28);
[...]
LINE452          local_34 = 0;
LINE453          if (0 < _height_from_dib) {
LINE454              do {
[...]
```

```
LINE469                 if (_color_mode == CMYK) {
[...]
LINE535                 }
LINE536                 else if (_color_mode == 0x8b7) {
[...]
LINE679                 }
LINE680                 else {
LINE681                     _index_width = (ushort *)0x0;
LINE682                     if (0 < width) {
LINE683                         _index_channel_loop = ____num_channel_image * 2;
LINE684                         table_per_channel = psd_buffer_oobw;
LINE686                         do {
LINE687                             if (param_5 == (int *)0x0) {
LINE688                                 if (bcheck_format == False) {
LINE689                                     if (bitsperchanneleq16_is2_or1 == 1) {
LINE690                                         index_num_channel_image = 0;
LINE691                                         currentMemory = table_per_channel;
LINE692                                         if (0 < (int)____num_channel_image)
{
LINE693                                             do {
LINE694                                                 if ((*chanel_image_data)

[index_num_channel_image] == 0xffffffff) {
LINE695                                                     *(undefined
*)currentMemory = 0;
LINE696                                                 }
LINE697                                                 else {
LINE698                                                     *(undefined
*)currentMemory =
LINE699                                                         *(undefined *)
LINE700                                                         ((int)_index_width
+
LINE701                                                          *(int
*)&channel_image_data
LINE702
[(*chanel_image_data)[index_num_channel_image]].field_0x1c)
LINE703                                                         ;
LINE704                                                 }
LINE705                                                 index_num_channel_image =
index_num_channel_image + 1;
LINE706                                                 currentMemory = (undefined4
*)((int)currentMemory + 1);
LINE707                                             } while
(index_num_channel_image < (int)____num_channel_image);
LINE708                                             goto next_index_channel_loop;
LINE709                                         }
LINE710                                     }
LINE711                                     else if (bitsperchanneleq16_is2_or1 ==
2) {
[...]
LINE731                                     }
LINE732                                 }
[...]
LINE774                             }
[...]
LINE852                             else if (bitsperchanneleq16_is2_or1 == 2) {
[...]
LINE881  next_index_channel_loop:
LINE882                                 _index_channel_loop = ____num_channel_image
* 2;
```

```
LINE883                                 }
LINE884                                 _index_width = (ushort *)((int)_index_width +
1);
LINE885                                 table_per_channel = (undefined4 *)
((int)table_per_channel + ____num_channel_image);
[...]
LINE888                             } while ((int)_index_width < width);
LINE889                         }
LINE890                     }
[...]
LINE1019                } while( true );
LINE1020            }
[...]
LINE1036        }
LINE1037 end:
LINE1038        _product_of_numchannel_bits_per_channel = AF_error_check();
LINE1039        return _product_of_numchannel_bits_per_channel;
LINE1040 Setbcheck_format:
LINE1041        bcheck_format = True;
LINE1042        goto start_alloc;
LINE1043 }
```

The out-of-bounds write is happening in the pointer variable `currentMemory` at LINE698. The `currentMemory` pointer value is set by the variable `table_per_channel` at LINE691. This variable `table_per_channel` is initialized the first time at LINE684 with the pointer variable named `psd_buffer_oobw`, then incremented by `____num_channel_image` at LINE885 through a do-while loop starting at LINE686 and ending at LINE888, controlled by the value of `width`. The variable `____num_channel_image` is a pseudo variable corresponding to `num_channel_image` at LINE78 along the code, read directly from the file.

The `psd_buffer_oobw` buffer is created at LINE108 with a size set from the variable `_size_buffer_alloc`, which is computed at LINE106. This is derived from `width`, which is directly read from the file, and `_product_of_numchannel_bits_per_channel`, which is computed earlier in the program. That said this `_product_of_numchannel_bits_per_channel` is set to the product of `num_channel_image`, read from the file, and `bits_per_channel`, also read directly from the file, allowing an attacker to completely control the out-of-bounds write.

The `currentMemory` is controlled by a second `do-while` loop, starting at LINE693, ending at LINE707 controlled by the value of `num_channel_image` boundary. The issue is happening when a value of ‘1’ is set to the `bits_per_channel` header of the PSD file, as the size computed at LINE106 can be smaller than `num_channel_image` due to the round in the formula with the constant ‘0xfffffffc’ at LINE106. Thus a missing check for a minimum size in the code is enabling the vulnerability to trigger. The assignments happening inside that function are out-of-bounds heap writes, which lead to memory corruption and possibly code execution.

Crash Information

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


KEY_VALUES_STRING: 1

    Key  : AV.Fault
    Value: Write

    Key  : Analysis.CPU.mSec
    Value: 5468

    Key  : Analysis.DebugAnalysisManager
    Value: Create

    Key  : Analysis.Elapsed.mSec
    Value: 14507

    Key  : Analysis.Init.CPU.mSec
    Value: 8155

    Key  : Analysis.Init.Elapsed.mSec
    Value: 93632

    Key  : Analysis.Memory.CommitPeak.Mb
    Value: 163

    Key  : Timeline.OS.Boot.DeltaSec
    Value: 381908

    Key  : Timeline.Process.Start.DeltaSec
    Value: 50

    Key  : WER.OS.Branch
    Value: vb_release

    Key  : WER.OS.Timestamp
    Value: 2019-12-06T14:06:00Z

    Key  : WER.OS.Version
    Value: 10.0.19041.1

    Key  : WER.Process.Version
    Value: 1.0.2.0


NTGLOBALFLAG:  2000000

PROCESS_BAM_CURRENT_THROTTLED: 0

PROCESS_BAM_PREVIOUS_THROTTLED: 0

APPLICATION_VERIFIER_FLAGS:  0
```

```
APPLICATION_VERIFIER_LOADED: 1

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 6d7cf76b (igcore20d!IG_mpi_page_set+0x000f35db)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 00000001
   Parameter[1]: 0c35c000
Attempt to write to address 0c35c000

FAULTING_THREAD:  00002464

PROCESS_NAME:  Fuzzme.exe

WRITE_ADDRESS:  0c35c000

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:  0c35c000

STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
0019f6a4 6d7ce14a     0019fc3c 0c19ee98 1000001e igcore20d!IG_mpi_page_set+0xf35db
0019f6d0 6d7cde94     0019fc3c 0ba78fe0 1000001e igcore20d!IG_mpi_page_set+0xf1fba
0019f724 6d7cc5d8     0019fc3c 0ba78fe0 1000001e igcore20d!IG_mpi_page_set+0xf1d04
0019f75c 6d7cbf4a     0019fc3c 0019f784 0019f7ac igcore20d!IG_mpi_page_set+0xf0448
0019fbb4 6d6b1399     0019fc3c 0ba78fe0 00000001 igcore20d!IG_mpi_page_set+0xefdba
0019fbec 6d6f09e7     00000000 0ba78fe0 0019fc3c
igcore20d!IG_image_savelist_get+0xb29
0019fe68 6d6f0349     00000000 05760fd0 00000001 igcore20d!IG_mpi_page_set+0x14857
0019fe88 6d685777     00000000 05760fd0 00000001 igcore20d!IG_mpi_page_set+0x141b9
0019fea8 00402239     05760fd0 0019febc 762ff550 igcore20d!IG_load_file+0x47
0019fec0 00402544     05760fd0 0019fef8 056c0f48 Fuzzme!fuzzme+0x19
0019ff28 004062a0     00000005 056baf68 056c0f48 Fuzzme!fuzzme+0x324
0019ff70 762ffa29     002b7000 762ffa10 0019ffdc Fuzzme!fuzzme+0x4080
0019ff80 77b97a9e     002b7000 65c54d1e 00000000 KERNEL32!BaseThreadInitThunk+0x19
0019ffdc 77b97a6e     ffffffff 77bb8a36 00000000 ntdll!__RtlUserThreadStart+0x2f
0019ffec 00000000     00406328 002b7000 00000000 ntdll!_RtlUserThreadStart+0x1b


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  igcore20d!IG_mpi_page_set+f35db

MODULE_NAME: igcore20d

IMAGE_NAME:  igcore20d.dll

FAILURE_BUCKET_ID:
INVALID_POINTER_WRITE_AVRF_c0000005_igcore20d.dll!IG_mpi_page_set

OS_VERSION:  10.0.19041.1
```

```
BUILDLAB_STR:  vb_release

OSPLATFORM_TYPE:  x86

OSNAME:  Windows 10

IMAGE_VERSION:  20.0.0.0

FAILURE_ID_HASH:  {fe8f80f8-683f-d41f-7c33-712a409d5fb5}

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

Timeline

2022-06-13 - Vendor Disclosure

2022-07-18 - Public Release

CREDIT

Discovered by Emmanuel Tacheau of Cisco Talos.
