Talos Vulnerability Report

TALOS-2021-1227

Accusoft ImageGear TIFF Header count processing out-of-bounds write vulnerability

MARCH 30, 2021

CVE NUMBER

CVE-2021-21773

Summary

An out-of-bounds write vulnerability exists in the TIFF header count-processing functionality of Accusoft ImageGear 19.8. 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 19.8

Product URLs

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-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 `tiff_read_sub_1` function, due to a buffer overflow caused by a missing null check for a size field.
A specially crafted TIFF file can lead to an out-of-bounds write which can result in memory corruption.

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

```
(8920.8408): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=10000021 ebx=0019fc3c ecx=00000000 edx=0adb4ff8 esi=00190001 edi=0adb0ff8
eip=7c1ab972 esp=0019f5f8 ebp=0019f604 iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010202
igCore19d!IG_cpm_profiles_reset+0x10b62:
7c1ab972 66897708        mov     word ptr [edi+8],si      ds:002b:0adb1000=????
```

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

```
0:000> !heap -p -a edi
    address 0adb0ff8 found in
    _DPH_HEAP_ROOT @ 4ea1000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                                 a993d9c:          adb0ff8                1 -          adb0000             2000
    7c0ea8b0 verifier!AVrfDebugPageHeapAllocate+0x00000240
    774eef8e ntdll!RtlDebugAllocateHeap+0x00000039
    77456150 ntdll!RtlpAllocateHeap+0x000000f0
    774557fe ntdll!RtlpAllocateHeapInternal+0x000003ee
    774553fe ntdll!RtlAllocateHeap+0x0000003e
    7c53dcff MSVCR110!malloc+0x00000049
    7c1b61de igCore19d!AF_memm_alloc+0x0000001e
    7c2c6ff0 igCore19d!IG_mpi_page_set+0x0010b2a0
    7c2c6f9a igCore19d!IG_mpi_page_set+0x0010b24a
    7c2cc0a3 igCore19d!IG_mpi_page_set+0x00110353
    7c2c604b igCore19d!IG_mpi_page_set+0x0010a2fb
    7c1910d9 igCore19d!IG_image_savelist_get+0x00000b29
    7c1d0557 igCore19d!IG_mpi_page_set+0x00014807
    7c1cfeb9 igCore19d!IG_mpi_page_set+0x00014169
    7c165777 igCore19d!IG_load_file+0x00000047
    004021f9 Fuzzme!fuzzme+0x00000019
    00402504 Fuzzme!fuzzme+0x00000324
    0040666d Fuzzme!fuzzme+0x0000448d
    7719fa29 KERNEL32!BaseThreadInitThunk+0x00000019
    774775f4 ntdll!__RtlUserThreadStart+0x0000002f
    774775c4 ntdll!_RtlUserThreadStart+0x0000001b
```

The crash is happening in the following pseudo code of the function `init_io_buffer` in LINE7 as `edi` contains the `io_buff` pointer:

```
LINE1  dword init_io_buffer(mys_table_function *mys_table_function_obj,uint kind_of_heap,io_buffer *io_buff
LINE2                      ,int size_to_allocate,short constant_1)
LINE3  {
LINE4
LINE5    set_into_param2_endian_type(mys_table_function_obj,&io_buff->endian_type);
LINE6    io_buff->ptr_mys_table_function = mys_table_function_obj;
LINE7    io_buff->field_0x8 = constant_1;
LINE8    io_buff->kind_of_heap = kind_of_heap;
LINE9    io_buff->buffer_size = size_to_allocate;
LINE10   io_buff->field_0xc = 0;
LINE11   [...]
LINE12 }
LINE13
```

The root cause is coming from the caller function `tiff_read_sub_1` which is responsible for the allocation of `io_buff`. The pseudo-code of `tiff_read_sub_1`:

```
LINE14 void tiff_read_sub_1(mys_table_function *mys_table_function_obj,uint kind_of_heap,
LINE15                      mys_tags_data *tags_data,undefined4 param_4,HIGDIBINFO Obj_HIGDIBINFO,
LINE16                      short *param_6)
LINE17 {
LINE18   io_buffer *ptr_io_buff;
LINE19   byte **dst;
LINE20   byte **error_message;
LINE21   size_t size;
LINE22   BYTE *pBVar1;
LINE23   int iVar2;
LINE24   dword dVar3;
LINE25   byte *pbVar4;
LINE26   int iVar5;
LINE27   int iVar6;
LINE28   io_buffer *piVar7;
LINE29   uint uVar8;
LINE30   HIGDIBINFO pIVar9;
LINE31   undefined4 uVar10;
LINE32   int local_28;
LINE33   byte *local_24;
LINE34   byte *local_20;
LINE35   int *local_1c;
LINE36   int local_18;
LINE37   uint size_to_allocate;
LINE38   uint local_c;
LINE39
LINE40   local_20 = NULL;
LINE41   size_to_allocate = 0;
LINE42   local_c = 0;
LINE43   local_24 = NULL;
LINE44   local_1c = NULL;
LINE45   ptr_io_buff = (io_buffer *)
LINE46                 AF_memm_alloc(kind_of_heap,
LINE47                               (uint)*(ushort *)&tags_data->SamplesPerPixel_Or_CountValue * 0x34,
LINE48                               (dword)"..\\..\\..\\..\\Common\\Formats\\tifread.c");
LINE49   if (ptr_io_buff == NULL) {
LINE50     AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\tifread.c",0x178b,-1000,0,0,0,NULL);
LINE51     AF_error_check();
LINE52     return;
LINE53   }
LINE54  [...]
LINE55 LAB_101771f6:
LINE56           dVar3 = init_io_buffer(mys_table_function_obj,kind_of_heap,ptr_io_buff,
LINE57                                  (int)*error_message * 5,1);
LINE58  [...]
LINE59 }
```

The `tiff_read_sub_1` function starts by allocating in LINE45 the memory buffer which can be overwritten represented by the variable `ptr_io_buff` . The size used for the memory allocation is directly read from the file, and corresponds to the `count` value of TIFF tags ImageWidth or ImageLength. By default this value is set to `1` according to documentation. There is no check against the buffer size before the call to the `init_io_buffer` in LINE56. The function `AF_memm_alloc` is a wrapper to `malloc`, thus when passing a null value it returns a buffer of one byte.

So, when calling `init_io_buffer`, that function assumes the `ptr_io_buff` points to a properly allocated `io_buffer` structure, and initializes some of its fields by writing inside the pointed structure. Since in our case the buffer is only 1 byte long, most of the assignments happening inside that function (e.g. LINE7 and LINE10 are writing constant values 1 and 0 respectively) 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: 1249

    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: 9397

    Key  : Analysis.Memory.CommitPeak.Mb
    Value: 72

    Key  : Analysis.System
    Value: CreateObject

    Key  : Timeline.OS.Boot.DeltaSec
    Value: 261278

    Key  : Timeline.Process.Start.DeltaSec
    Value: 2029

    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: 1.0.1.1


ADDITIONAL_XML: 1

OS_BUILD_LAYERS: 1

NTGLOBALFLAG:  2100000

APPLICATION_VERIFIER_FLAGS:  0

APPLICATION_VERIFIER_LOADED: 1

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 7c1ab972 (igCore19d!IG_cpm_profiles_reset+0x00010b62)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 00000001
   Parameter[1]: 0adb1000
Attempt to write to address 0adb1000

FAULTING_THREAD:  00008408

PROCESS_NAME:  Fuzzme.exe

WRITE_ADDRESS:  0adb1000

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:  0adb1000

STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
0019f604 7c2c7648     0019fc3c 10000021 0adb0ff8 igCore19d!IG_cpm_profiles_reset+0x10b62
0019f650 7c2c6f9a     0019fc3c 10000021 0adb4ff8 igCore19d!IG_mpi_page_set+0x10b8f8
0019f678 7c2cc0a3     0019fc3c 10000021 0019f6d0 igCore19d!IG_mpi_page_set+0x10b24a
0019f6a0 7c2c604b     0019fc3c 10000021 0ab7dd68 igCore19d!IG_mpi_page_set+0x110353
0019fbb4 7c1910d9     0019fc3c 0ab7dd68 00000001 igCore19d!IG_mpi_page_set+0x10a2fb
0019fbec 7c1d0557     00000000 0ab7dd68 0019fc3c igCore19d!IG_image_savelist_get+0xb29
0019fe68 7c1cfeb9     00000000 05295f80 00000001 igCore19d!IG_mpi_page_set+0x14807
0019fe88 7c165777     00000000 05295f80 00000001 igCore19d!IG_mpi_page_set+0x14169
0019fea8 004021f9     05295f80 0019febc 00000001 igCore19d!IG_load_file+0x47
0019fec0 00402504     05295f80 05293fc0 051e3f28 Fuzzme!fuzzme+0x19
0019ff28 0040666d     00000005 051daee8 051e3f28 Fuzzme!fuzzme+0x324
0019ff70 7719fa29     00268000 7719fa10 0019ffdc Fuzzme!fuzzme+0x448d
0019ff80 774775f4     00268000 0f9e4eaf 00000000 KERNEL32!BaseThreadInitThunk+0x19
0019ffdc 774775c4     ffffffff 77497336 00000000 ntdll!__RtlUserThreadStart+0x2f
0019ffec 00000000     004066f5 00268000 00000000 ntdll!_RtlUserThreadStart+0x1b


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  igCore19d!IG_cpm_profiles_reset+10b62

MODULE_NAME: igCore19d

IMAGE_NAME:  igCore19d.dll
```

```
FAILURE_BUCKET_ID:  INVALID_POINTER_WRITE_AVRF_c0000005_igCore19d.dll!IG_cpm_profiles_reset

OS_VERSION:  10.0.19041.1

BUILDLAB_STR:  vb_release

OSPLATFORM_TYPE:  x86

OSNAME:  Windows 10

IMAGE_VERSION:  19.8.0.0

FAILURE_ID_HASH:  {749e662e-5382-aab8-f02d-cecd73653ce6}

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

Timeline

2021-01-15 - Vendor Disclosure
2021-02-05 - Vendor Patched
2021-03-30 - Public Release

CREDIT

Discovered by Emmanuel Tacheau of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2020-1226

NEXT REPORT TALOS-2021-1232