Talos Vulnerability Report

TALOS-2020-1176

Accusoft ImageGear TIFF index record out-of-bounds write vulnerability

FEBRUARY 9, 2021

CVE NUMBER

CVE-2020-13561

Summary

An out-of-bounds write vulnerability exists in the TIFF parser 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-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, including many image formats, DICOM, PDF, Microsoft Office and others.

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

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

```
(4c0c.82e8): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000001 ebx=00000003 ecx=000003fa edx=00000400 esi=00000002 edi=19f11000
eip=7b4f0203 esp=00cfed7c ebp=00cfed90 iopl=0         nv up ei pl nz na pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010207
MSVCR110!memset+0x24:
7b4f0203 f3aa            rep stos byte ptr es:[edi]

0:000> dd edi
19f11000  ???????? ???????? ???????? ????????
19f11010  ???????? ???????? ???????? ????????
19f11020  ???????? ???????? ???????? ????????
19f11030  ???????? ???????? ???????? ????????
```

An out-of-bounds write operation occurred during the memset above.

Further analysis reveals that if the file contains a malformed record, in our case it is :

```
00F6h: 41 41 41 41 41 41 41 41 41 41 41 41              AAAAAAAAAAAA
```

then the next 0x10 bytes are treated in a special way. In our PoC mentioned 0x10 bytes present as follow:

```
0102h: 05 00 00 00 00 00 00 00 00 00 00 00 00 00 41 41  ..............AA
```

Let's call this the `index record`. Based on the values in this `index record`, the amount of indexes are stored into an `index_values` buffer. Let's take a closer look at the code where all this happens:

```
igCore19d - image base : 0x79D10000

.text:79E30852 loc_79E30852:                          ; CODE XREF: sub_79E304B0+3E7↓j
.text:79E30852                 movzx   eax, byte ptr [esi+ebx] ; ebx = record beg ( 5 )
.text:79E30852                                        ; esi = 1
.text:79E30856                 mov     [ebp+var_518], eax
.text:79E3085C                 cmp     eax, 1
.text:79E3085F                 jl      short loc_79E30893
.text:79E30861                 lea     edi, [ebp+index_values]
.text:79E30867                 lea     edi, [edi+ecx*2]
.text:79E3086A                 mov     ecx, eax
.text:79E3086C                 mov     eax, esi
.text:79E3086E                 movzx   edx, ax
.text:79E30871                 mov     eax, edx
.text:79E30873                 shl     edx, 10h
.text:79E30876                 or      eax, edx
.text:79E30878                 shr     ecx, 1
.text:79E3087A                 rep stosd
.text:79E3087C                 adc     ecx, ecx
.text:79E3087E                 rep stosw
.text:79E30881                 mov     ecx, [ebp+var_528]
.text:79E30887                 add     ecx, [ebp+var_518]
.text:79E3088D                 mov     [ebp+var_528], ecx
.text:79E30893
.text:79E30893 loc_79E30893:                          ; CODE XREF: sub_79E304B0+3AF↑j
.text:79E30893                 inc     esi
.text:79E30894                 cmp     esi, 10h
.text:79E30897                 jle     short loc_79E30852 ; ebx = record beg ( 5 )
```

The code above iterates through all of the elements in the `index record` and for each of these elements it copies the loop counter to the `index_values` address an amount of times that's equal to the value of the `index record` element value.

For example:

If the first `index record` value is : 0x5 and its at position 1 (loop counter 1 (`esi`)), then 5 times the value 0x1 (WORD length) will be stored in successive positions of the `index_values` "array". So, after the first iteration the `index_values` array will look as follows:

```
index_values = {0x0001,0x0001,0x0001,0x0001,0x0001 }
```

and so on.

The vulnerability exists in the code below:

```
.text:79E30990 loc_79E30990:                          ; CODE XREF: sub_79E304B0+56B↓j
.text:79E30990                 lea     ecx, [ebx+ebx]
.text:79E30993                 movzx   edx, di
.text:79E30996                 movzx   eax, [ebp+ecx+index_values] ; [WORD]index values
.text:79E3099E                 mov     [ebp+var_518], edx
.text:79E309A4                 cmp     eax, edx
.text:79E309A6                 jnz     short loc_79E30A00
.text:79E309A8                 jmp     short loc_79E309B0 ; ebx = counter
.text:79E309A8 ; ---------------------------------------------------------------------------
.text:79E309AA                 align 10h
.text:79E309B0
.text:79E309B0 loc_79E309B0:                          ; CODE XREF: sub_79E304B0+4F8↑j
.text:79E309B0                                        ; sub_79E304B0+54E↓j
.text:79E309B0                 inc     ebx            ; ebx = counter
.text:79E309B1                 mov     [ebp+ecx+var_50C], si
.text:79E309B9                 cmp     di, 0Ch
.text:79E309BD                 jnb     short loc_79E309F0
.text:79E309BF                 mov     ecx, edx
.text:79E309C1                 mov     eax, 800h
.text:79E309C6                 shr     eax, cl
.text:79E309C8                 mov     ecx, 0Bh
.text:79E309CD                 sub     ecx, edx
.text:79E309CF                 push    eax            ; size_t
.text:79E309D0                 movzx   eax, si
.text:79E309D3                 shl     eax, cl
.text:79E309D5                 mov     ecx, [ebp+buffer]
.text:79E309DB                 add     ecx, 282h
.text:79E309E1                 push    edi            ; char
.text:79E309E2                 add     eax, ecx
.text:79E309E4                 push    eax            ; void *
.text:79E309E5                 call    memset_wrapper
.text:79E309EA                 mov     edx, [ebp+var_518]
.text:79E309F0
.text:79E309F0 loc_79E309F0:                          ; CODE XREF: sub_79E304B0+50D↑j
.text:79E309F0                 lea     ecx, [ebx+ebx]
.text:79E309F3                 inc     esi
.text:79E309F4                 movzx   eax, [ebp+ecx+index_values]
.text:79E309FC                 cmp     eax, edx
.text:79E309FE                 jz      short loc_79E309B0 ; ebx = counter
```

We can try to present it as a pesudo-code:

```
Line 1  edi = index_values[0];
Line 2  esi = 0;
Line 3  for(ebx = 0; all_elements ;ebx+=2)
Line 4  {
Line 5          if(edi == index_values[ebx])
Line 6          {
Line 7                  do
Line 8                  {
Line 9                          ebx++;
Line 10
Line 11                         memset(buffer+0x282+(esi<<(0xB-edi)),edi,0x800>>edi);
Line 12                         esi++;
Line 13
Line 14                 }while( edi == index_values[ebx*2]);
Line 15
Line 16
Line 17         }
Line 18         esi +=2;
Line 19         edi++;
Line 20 }
```

The code will loop over `index_values` and will load the stored value in the `edi` register. As we can see, inside the loop at `line 11`, `memset` writes `edi` to `buffer` and the size of the write is based on shifting `0x800` right with the `edi` register. The offset of the buffer to write to is also moved ahead by the `esi` right shifted by `0xB` - `edi`. This means that both the content the size of the write and the offset from the destination buffer are controlled by the attacker.

Buffer is allocated as shown below:

```
0:000> !heap -p -a 128817fa
        address 128817fa found in
        _DPH_HEAP_ROOT @ 3581000
        in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                                                                127b2f70:         12881578              a82 -         12881000
2000
        78dfa8b0 verifier!AVrfDebugPageHeapAllocate+0x00000240
        770cec5e ntdll!RtlDebugAllocateHeap+0x00000039
        770360f0 ntdll!RtlpAllocateHeap+0x000000f0
        7703579e ntdll!RtlpAllocateHeapInternal+0x000003ee
        7703539e ntdll!RtlAllocateHeap+0x0000003e
        7b4cdaff MSVCR110!malloc+0x00000049
        79d761de igCore19d!AF_memm_alloc+0x0000001e
        79e30643 igCore19d!IG_mpi_page_set+0x000b48f3
        79e80b6e igCore19d!IG_mpi_page_set+0x00104e1e
        79e83918 igCore19d!IG_mpi_page_set+0x00107bc8
        79e8c0e9 igCore19d!IG_mpi_page_set+0x00110399
        79e8604b igCore19d!IG_mpi_page_set+0x0010a2fb
        79d510d9 igCore19d!IG_image_savelist_get+0x00000b29
        79d90557 igCore19d!IG_mpi_page_set+0x00014807
        79d8feb9 igCore19d!IG_mpi_page_set+0x00014169
        79d25777 igCore19d!IG_load_file+0x00000047
        005220d0 Fuzzme!fuzzme+0x000000b0
        00522502 Fuzzme!fuzzme+0x000004e2
        00527a5a Fuzzme!fuzzme+0x00005a3a
        75d9f989 KERNEL32!BaseThreadInitThunk+0x00000019
        770574a4 ntdll!__RtlUserThreadStart+0x0000002f
        77057474 ntdll!_RtlUserThreadStart+0x0000001b
```

and is size is constant `0A82h`:

```
.text:79E3059D                 push    296h            ; int
.text:79E305A2                 push    offset aCommonFormatsJ ; "..\\..\\..\\..\\Common\\Formats\\jpeg_d"...
.text:79E305A7                 push    0A82h           ; size_t
.text:79E305AC                 push    edi             ; int
.text:79E305AD                 call    AF_memm_alloc
.text:79E305B2                 mov     [ebp+esi*4+var_540], eax
```

Tracing subsequent loop executions we will see that `memset` will be called with the following values:

```
Loop values during iterations:

#1
memset(buffer+0x282,1,0x400)

#2
memset(buffer+0x282+0x400,1,0x400)

#3
memset(buffer+0x282+0x800,1,0x400) <=== overflow
```

Since the size of the memset is `0x800` shifted right by the value stored in `index_values`, in our example above this is 0x0001, so after 2 iterations, the buffer will be set at the end of the allocated memory, meaning the 3rd memset will result in an out-of-bounds wite.

Crash Information

```
(d344.7590): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000001 ebx=00000003 ecx=000003fa edx=00000400 esi=00000002 edi=1a69d000
eip=7b4f0203 esp=010fe92c ebp=010fe940 iopl=0         nv up ei pl nz na pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010207
MSVCR110!memset+0x24:
7b4f0203 f3aa            rep stos byte ptr es:[edi]
0:000> !analyze -v
*******************************************************************************
*                                                                             *
*                        Exception Analysis                                   *
*                                                                             *
*******************************************************************************


KEY_VALUES_STRING: 1

        Key  : AV.Fault
        Value: Write

        Key  : Analysis.CPU.mSec
        Value: 5062

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

        Key  : Analysis.DebugData
        Value: CreateObject

        Key  : Analysis.DebugModel
        Value: CreateObject

        Key  : Analysis.Elapsed.mSec
        Value: 251661

        Key  : Analysis.Memory.CommitPeak.Mb
        Value: 183

        Key  : Analysis.System
        Value: CreateObject

        Key  : Timeline.OS.Boot.DeltaSec
        Value: 3329254

        Key  : Timeline.Process.Start.DeltaSec
        Value: 1240

        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: 1.0.0.2


ADDITIONAL_XML: 1

OS_BUILD_LAYERS: 1

NTGLOBALFLAG:  2000000

APPLICATION_VERIFIER_FLAGS:  0

APPLICATION_VERIFIER_LOADED: 1

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 7b4f0203 (MSVCR110!memset+0x00000024)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 00000001
   Parameter[1]: 1a69d000
Attempt to write to address 1a69d000

FAULTING_THREAD:  00007590

PROCESS_NAME:  Fuzzme.exe

WRITE_ADDRESS:  1a69d000

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:  1a69d000

STACK_TEXT:
010fe92c 79d1fc03     1a69cffa 00000001 00000400 MSVCR110!memset+0x24
WARNING: Stack unwind information not available. Following frames may be wrong.
010fe940 79e309ea     1a69cffa 00000001 00000400 igCore19d+0xfc03
010feeb0 79e80b6e     1a6aef60 010ff554 00000010 igCore19d!IG_mpi_page_set+0xb4c9a
010feed8 79e83918     010ff554 00000000 1a60ef90 igCore19d!IG_mpi_page_set+0x104e1e
010fef94 79e8c0e9     010ff554 1a576d68 010fefe8 igCore19d!IG_mpi_page_set+0x107bc8
010fefb8 79e8604b     010ff554 1000002f 1a576d68 igCore19d!IG_mpi_page_set+0x110399
010ff4cc 79d510d9     010ff554 1a576d68 00000001 igCore19d!IG_mpi_page_set+0x10a2fb
010ff504 79d90557     00000000 1a576d68 010ff554 igCore19d!IG_image_savelist_get+0xb29
010ff780 79d8feb9     00000000 0a1c2f80 00000001 igCore19d!IG_mpi_page_set+0x14807
010ff7a0 79d25777     00000000 0a1c2f80 00000001 igCore19d!IG_mpi_page_set+0x14169
010ff7c0 005220d0     0a1c2f80 010ff7d4 0a102f48 igCore19d!IG_load_file+0x47
010ff7e0 00522502     0a1c2f80 010ff848 00000021 Fuzzme!fuzzme+0xb0
010ff870 00527a5a     00000005 0a102f48 0a109f18 Fuzzme!fuzzme+0x4e2
010ff8b8 75d9f989     00ea6000 75d9f970 010ff924 Fuzzme!fuzzme+0x5a3a
010ff8c8 770574a4     00ea6000 ef6ddc25 00000000 KERNEL32!BaseThreadInitThunk+0x19
010ff924 77057474     ffffffff 77077354 00000000 ntdll!__RtlUserThreadStart+0x2f
010ff934 00000000     00527ae2 00ea6000 00000000 ntdll!_RtlUserThreadStart+0x1b
```

```
STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  MSVCR110!memset+24

MODULE_NAME: MSVCR110

IMAGE_NAME:  MSVCR110.dll

FAILURE_BUCKET_ID:  INVALID_POINTER_WRITE_STRING_DEREFERENCE_AVRF_c0000005_MSVCR110.dll!memset

OS_VERSION:  10.0.19041.1

BUILDLAB_STR:  vb_release

OSPLATFORM_TYPE:  x86

OSNAME:  Windows 10

IMAGE_VERSION:  11.0.50727.1

FAILURE_ID_HASH:  {56b0676a-0fb6-0730-2b71-15517df752d2}

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

0:000> lmv a 79e80b6e
Browse full module list
start    end        module name
79d10000 7a059000   igCore19d   (export symbols)       d:\projects\ImageGear\v19\Build\Bin\x86\igCore19d.dll
        Loaded symbol image file: d:\projects\ImageGear\v19\Build\Bin\x86\igCore19d.dll
        Image path: d:\projects\ImageGear\v19\Build\Bin\x86\igCore19d.dll
        Image name: igCore19d.dll
        Browse all global symbols  functions  data
        Timestamp:        Thu Aug 20 20:45:15 2020 (5F3EC4BB)
        CheckSum:         003494E0
        ImageSize:        00349000
        File version:     19.8.0.0
        Product version:  19.8.0.0
        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:   19.8.0.0
                FileVersion:      19.8.0.0
                FileDescription:  Accusoft ImageGear CORE DLL
                LegalCopyright:   Copyright 1996-2020 Accusoft Corporation. All rights reserved.
                LegalTrademarks:  ImageGear® and Accusoft® are registered trademarks of Accusoft Corporation
```

Timeline

2020-10-26 - Vendor Disclosure

2021-02-05 - Vendor patched

2021-02-09 - Public Release

CREDIT

Discovered by Emmanuel Tacheau and a member of Cisco Talos.