Talos Vulnerability Report

TALOS-2020-1182

Accusoft ImageGear SGI RLE decompression out-of-bounds write vulnerability

FEBRUARY 9, 2021

CVE NUMBER

CVE-2020-13571

Summary

An out-of-bounds write vulnerability exists in the SGI RLE decompression 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-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 `sgiread` function, due to a buffer overflow caused by a missing check of the input size.
A specially crafted SGI file can lead to an out-of-bounds write which can result in a memory corruption.

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

```
This exception may be expected and handled.
eax=26224e4f ebx=67676767 ecx=00001f4f edx=00003e4f esi=26222f00 edi=124b7000
eip=7a40df22 esp=006ff390 ebp=006ff3a8 iopl=0         nv up ei ng nz na pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010287
MSVCR110!memcpy+0x2a:
7a40df22 f3a4            rep movs byte ptr es:[edi],byte ptr [esi]

0:000> !heap -p -a edi
    address 124b7000 found in
    _DPH_HEAP_ROOT @ 871000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                                173e3af8:         124b5100             1eff -         124b5000             3000
    7a37ab70 verifier!AVrfDebugPageHeapAllocate+0x00000240
    7701909b ntdll!RtlDebugAllocateHeap+0x00000039
    76f6bbad ntdll!RtlpAllocateHeap+0x000000ed
    76f6b0cf ntdll!RtlpAllocateHeapInternal+0x0000022f
    76f6ae8e ntdll!RtlAllocateHeap+0x0000003e
    7a40dcff MSVCR110!malloc+0x00000049
    7a6f61de igCore19d!AF_memm_alloc+0x0000001e
    7a7fa423 igCore19d!IG_mpi_page_set+0x000fe6d3
    7a7fafc7 igCore19d!IG_mpi_page_set+0x000ff277
    7a7f9d95 igCore19d!IG_mpi_page_set+0x000fe045
    7a6d10d9 igCore19d!IG_image_savelist_get+0x00000b29
    7a710557 igCore19d!IG_mpi_page_set+0x00014807
    7a70feb9 igCore19d!IG_mpi_page_set+0x00014169
    7a6a5777 igCore19d!IG_load_file+0x00000047
    00e920d0 Fuzzme!fuzzme+0x000000b0
    00e92502 Fuzzme!fuzzme+0x000004e2
    00e97a5a Fuzzme!fuzzme+0x00005a3a
    76146359 KERNEL32!BaseThreadInitThunk+0x00000019
    76f97c24 ntdll!__RtlUserThreadStart+0x0000002f
    76f97bf4 ntdll!_RtlUserThreadStart+0x0000001b
```

An out-of-bounds write operation occurred during the `memcpy` above, which is called by `IO_read`.

Of course we need go back along the stack to find the root cause and we'd land in the `sgiread` function with the following pseudo code:

```
LINE 15   void sgiread(mys_table_function *mys_table_func_obj,uint kind_heap,SGI_FILE *sgi_header_from_file,
LINE 16               int param_4,IGDIBStd *IGDIBStd_obj)
LINE 17
LINE 18   {
LINE 19     void *mem_to_free;
LINE 20     int SamplePerPixel;
LINE 21     dword _ylength;
LINE 22     uint bit_depth;
LINE 23     byte *buffer_1;
LINE 24     BYTE **table_scanline_1;
LINE 25     BYTE **table_double_xsize_buffers;
LINE 26     io_buffer *not_rle_buffer;
LINE 27     void *buff_double_xsize;
LINE 28     BYTE *buff_xsize;
LINE 29     int size_read;
LINE 30     byte *pbVar1;
LINE 31     byte *pbVar2;
LINE 32     io_buffer *io_buff;
LINE 33     byte **dest_buffer;
LINE 34     uint xsize;
LINE 35     uint uVar3;
LINE 36     BYTE **ylength;
LINE 37     bool notRLE;
LINE 38     BYTE **__xsize;
LINE 39     uint num_channel;
LINE 40     BYTE **table_xsize_buffers;
LINE 41     int status;
LINE 42
LINE 43     SamplePerPixel = get_SamplesPerPixel((HDIB)IGDIBStd_obj);
LINE 44     notRLE = sgi_header_from_file->sig_Storage_format != 1;
LINE 45     status = 0;
LINE 46     _ylength = getLength(IGDIBStd_obj);
LINE 47     xsize = (uint)sgi_header_from_file->sig_xsize;
LINE 48     ylength = (BYTE **)(_ylength & 0xffff);
LINE 49     bit_depth = compute_raster_size_if_bit_depth_more_than_1(IGDIBStd_obj);
LINE 50     buffer_1 = AF_memm_alloc(kind_heap,bit_depth);
LINE 51     if (buffer_1 == (byte *)0x0) {
LINE 52       status = AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\sgiread.c",0x34e,-1000,0,bit_depth,
LINE 53                                  kind_heap,(char *)0x0);
LINE 54     }
LINE 55     table_scanline_1 = (BYTE **)AF_memm_alloc(kind_heap,(uint)sgi_header_from_file->sig_zsize << 2);
LINE 56     table_double_xsize_buffers =
LINE 57          (BYTE **)AF_memm_alloc(kind_heap,(uint)sgi_header_from_file->sig_zsize << 2);
LINE 58     not_rle_buffer =
LINE 59          (io_buffer *)AF_memm_alloc(kind_heap,(uint)sgi_header_from_file->sig_zsize * 0x34);
LINE 60     if (((table_scanline_1 == (BYTE **)0x0) || (table_double_xsize_buffers == (BYTE **)0x0)) ||
LINE 61        (not_rle_buffer == (io_buffer *)0x0)) {
LINE 62       status = AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\sgiread.c",0x355,-1000,0,0,0,
LINE 63                                  (char *)0x0);
LINE 64     }
LINE 65     else {
LINE 66       wrapper_memset(table_scanline_1,0,(uint)sgi_header_from_file->sig_zsize << 2);
LINE 67       wrapper_memset(table_double_xsize_buffers,0,(uint)sgi_header_from_file->sig_zsize << 2);
LINE 68       wrapper_memset(not_rle_buffer,0,(uint)sgi_header_from_file->sig_zsize * 0x34);
LINE 69     }
LINE 70     if (status == 0) {
LINE 71       num_channel = 0;
LINE 72       if (notRLE) {
LINE 73         [...]
LINE 74       }
LINE 75       /* Decode RLE */
LINE 76       else {
LINE 77         if (sgi_header_from_file->sig_zsize != 0) {
LINE 78
LINE 79         /* Create a pseudo index with buff_double_xsize & _table_scanline_1 */
LINE 80           table_xsize_buffers = table_scanline_1;
LINE 81           do {
LINE 82
LINE 83           /* Allocate buffers size controlled from file */
LINE 84             buff_double_xsize = AF_memm_alloc(kind_heap,xsize * 2 + 1);                                                    [4]
LINE 85
LINE 86           /* In other words: *table_double_xsize_buffers[index] = buff_double_xsize */                                     [3]
LINE 87             *(void **)((int)((int)table_double_xsize_buffers - (int)table_scanline_1) +
LINE 88                        (int)table_xsize_buffers) = buff_double_xsize;
LINE 89
LINE 90             if (buff_double_xsize == (void *)0x0) {
LINE 91               status = AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\sgiread.c",0x365,-1000,0,
LINE 92                                          xsize,kind_heap,(char *)0x0);
LINE 93             }
LINE 94
LINE 95             buff_xsize = AF_memm_alloc(kind_heap,xsize);
LINE 96
LINE 97           /* In the same time fill in table of pointer table_xsize_buffers too */
LINE 98             *table_xsize_buffers = buff_xsize;
LINE 99
LINE 100          if (buff_xsize == (BYTE *)0x0) {
LINE 101              status = AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\sgiread.c",0x369,-1000,0,
LINE 102                                         xsize,kind_heap,(char *)0x0);
LINE 103            }
LINE 104
LINE 105          /* in the same time index += 4  */
LINE 106          table_xsize_buffers = table_xsize_buffers + 1;
LINE 107
LINE 108            num_channel = num_channel + 1;
LINE 109          } while (num_channel < sgi_header_from_file->sig_zsize);
LINE 110
LINE 111          if (status != 0) goto LAB_1016a6ed;
LINE 112        }
LINE 113      }
LINE 114      rowno = (BYTE **)0x0;
LINE 115      if (ylength != (BYTE **)0x0) {
LINE 116        __xsize = ylength;
LINE 117        while( true ) {
LINE 118          __xsize = (BYTE **)((int)__xsize - 1);
LINE 119
LINE 120        if (notRLE) {
LINE 121            [...]
LINE 122          }
LINE 123
LINE 124        /* RLE Parsing starttab and lengthtab of sgi file */
LINE 125          else {
LINE 126            num_channel = 0;
LINE 127            if (sgi_header_from_file->sig_zsize != 0) {
LINE 128              dest_buffer = table_double_xsize_buffers;
LINE 129              do {
LINE 130                  /*
LINE 131                    rleoffset = starttab[rowno+channo*YSIZE]
```

```
LINE 132                    IO_seek(mys_table_func_obj,rleoffset, 0)                                                    [5]
LINE 133                */
LINE 134               IO_seek(mys_table_func_obj,
LINE 135                       sgi_header_from_file->starttab
LINE 136                       [(int)(sgi_header_from_file->sig_ysize * num_channel +
LINE 137                             rowno)],0);
LINE 138
LINE 139               /*
LINE 140                   rlelength = lengthtab[rowno+channo*YSIZE]
LINE 141                   size_read = IO_read(mys_table_func_obj,table_double_xsize_buffers[rowno],rlelength)          [2]
LINE 142               */
LINE 143               size_read = IO_read(mys_table_func_obj,*dest_buffer,                                            [1]
LINE 144                                   sgi_header_from_file->lengthtab
LINE 145                                   [(int)(sgi_header_from_file->sig_ysize * num_channel +
LINE 146                                         rowno)]);
LINE 147
LINE 148
LINE 149               status = kind_of_memcpy(*dest_buffer,
LINE 150                                       *(byte **)((int)((int)table_scanline_1 -
LINE 151                                                        (int)table_double_xsize_buffers) +
LINE 152                                                  (int)dest_buffer),size_read,xsize);
LINE 153               num_channel = num_channel + 1;
LINE 154               dest_buffer = dest_buffer + 1;
LINE 155             } while (num_channel < sgi_header_from_file->sig_zsize);
LINE 156           }
LINE 157         }
LINE 158       [...]
LINE 159       }
LINE 160     }
LINE 161   }
LINE 162  [...]
LINE 163 }
```

The memory corruption is happening while reading the content of the file in [1] through the call to the function `IO_read` which at the end will lead to `ReadFile` `winapi`. As the decompilation sometimes is not always trivial to get, we can rewrite the `IO_read` line differently, see the comment in [2]. This part of code is reading into buffers which are located into the table named `table_double_xsize_buffers` of size `rlelength`. The `rlelength` value is controlled from the input file.
The buffers sizes for `table_double_xsize_buffers` [3] are also controlled from the input file, using the `xsize` value from the SGI header [4], well known as `scanline` size.

But what is the condition that makes the overwrite happen? In fact we need to understand a fact about the `ReadFile` `winapi`, which is the following: in a situation where a call to read a file is performed with a size larger than the bytes left, `ReadFile` will return the bytes from the current offset to the end of the file. So for example after several reads, depending of the seek method call of course, it is possible to return the entire file if the offset from the start of the file is 0 and the requested size is larger than the file size.

In [5] we can see that before the call to `IO_read` we have a call to `IO_seek`. Keep in mind that `rleoffset` is taken from the input file, which is the standard mechanism of RLE compression in an SGI file. If we take a closer look at `IO_seek` we can see the following pseudo code:

```
LINE 169  dword IO_seek(mys_table_function *obj_mys_table_function,int lDistanceToMove,int dwMoveMethod)
LINE 170  {
LINE 171    dword dVar1;
LINE 172    dword _nextoffset;
LINE 173    [...]
LINE 174    _nextoffset = perform_set_file_pointer_operations_related                                                   [6]
LINE 175                            (obj_mys_table_function,lDistanceToMove,dwMoveMethod);
LINE 176    return _nextoffset;
LINE 177  }
```

`IO_seek` is in fact a kind of wrapper to another function named here `perform_set_file_pointer_operations_related` [6] with the following pseudo code:

```
LINE 179  int perform_set_file_pointer_operations_related
LINE 180                (mys_table_function *param_1,int seek_offset,int dwMoveMethod)
LINE 181  {
LINE 182    dword dVar1;
LINE 183    int file_size;
LINE 184    dword size_of_bloc;
LINE 185    int iVar2;
LINE 186    int new_offset;
LINE 187    bool bVar3;
LINE 188    bool bVar4;
LINE 189    [...]
LINE 190    if (dwMoveMethod == 0) {
LINE 191        if (seek_offset < 0) {                                                                                  [7]
LINE 192            seek_offset = 0;
LINE 193        }
LINE 194        if (((size_of_bloc == 0) || (seek_offset <= (int)(file_size - size_of_bloc))) || (file_size <= seek_offset)) {
LINE 196            size_of_bloc = set_file_pointer_related(param_1,seek_offset,0);                                     [8]
LINE 197            return size_of_bloc;
LINE 198          }
LINE 199        }
LINE 200    [...]
LINE 201    return seek_offset;
LINE 202   }
```

Obviously what is happening is that if the seek offset represented by the `seek_offset` variable is negative, it's set to 0 [7] enforcing to seek to the start of the file [8].

In summary, if the `rleoffset` is negative, the code is seeking to the start of the file and then reading the data pointed by `rleoffset` with a size of `rlelength`. If the `rlelength` requested is greater than `xsize * 2 + 1` (the `dest_buffer` size) and the `file_size` is also greater than `xsize * 2 + 1`, an out-of-bound write will occur in [1], leading to memory corruption and possibly code execution.

To make it happen several preconditions are required: - RLE format compression should be used. - the `zsize` corresponding to the channel must be equal or greater than the value of '4' - a negative value must be present into any entry of the `starttab` table for all `rleoffset`. - `rlelength` must be superior to `scanline` size - `file_size` must be superior to `scanline` size

Crash Information

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


KEY_VALUES_STRING: 1

    Key  : AV.Fault
    Value: Write

    Key  : Analysis.CPU.mSec
    Value: 3031

    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: 16715

    Key  : Analysis.Memory.CommitPeak.Mb
    Value: 170

    Key  : Analysis.System
    Value: CreateObject

    Key  : Timeline.OS.Boot.DeltaSec
    Value: 166559

    Key  : Timeline.Process.Start.DeltaSec
    Value: 57

    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: 7a40df22 (MSVCR110!memcpy+0x0000002a)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 00000001
   Parameter[1]: 124b7000
Attempt to write to address 124b7000

FAULTING_THREAD:  00005b64

PROCESS_NAME:  Fuzzme.exe

WRITE_ADDRESS:  124b7000

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:  124b7000

STACK_TEXT:
006ff394 7a69f9c6     124b5100 26221000 00003e4f MSVCR110!memcpy+0x2a
WARNING: Stack unwind information not available. Following frames may be wrong.
006ff3a8 7a6ea6fd     124b5100 26221000 00003e4f igCore19d+0xf9c6
006ff3cc 7a6e9e42     00000000 124b5100 67676767 igCore19d!IG_cpm_profiles_reset+0xf8ed
006ff3e4 7a7fa58b     006ffb54 124b5100 67676767 igCore19d!IG_cpm_profiles_reset+0xf032
006ff440 7a7fafc7     006ffb54 1000002a 174cbf80 igCore19d!IG_mpi_page_set+0xfe83b
006ff474 7a7f9d95     006ffb54 1000002a 12cf6ff8 igCore19d!IG_mpi_page_set+0xff277
006ffacc 7a6d10d9     006ffb54 12cf6ff8 00000001 igCore19d!IG_mpi_page_set+0xfe045
006ffb04 7a710557     00000000 12cf6ff8 006ffb54 igCore19d!IG_image_savelist_get+0xb29
006ffd80 7a70feb9     00000000 097c1f30 00000001 igCore19d!IG_mpi_page_set+0x14807
006ffda0 7a6a5777     00000000 097c1f30 00000001 igCore19d!IG_mpi_page_set+0x14169
006ffdc0 00e920d0     097c1f30 006ffdd4 09700ec0 igCore19d!IG_load_file+0x47
006ffde0 00e92502     097c1f30 097bffe0 00000021 Fuzzme!fuzzme+0xb0
006ffe70 00e97a5a     00000005 09700ec0 09707f28 Fuzzme!fuzzme+0x4e2
006ffeb8 76146359     004e1000 76146340 006fff24 Fuzzme!fuzzme+0x5a3a
006ffec8 76f97c24     004e1000 6c3fd3a1 00000000 KERNEL32!BaseThreadInitThunk+0x19
006fff24 76f97bf4     ffffffff 76fb8ff5 00000000 ntdll!__RtlUserThreadStart+0x2f
006fff34 00000000     00e97ae2 004e1000 00000000 ntdll!_RtlUserThreadStart+0x1b


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  MSVCR110!memcpy+2a

MODULE_NAME: MSVCR110
```

```
IMAGE_NAME:  MSVCR110.dll

FAILURE_BUCKET_ID:  INVALID_POINTER_WRITE_STRING_DEREFERENCE_AVRF_c0000005_MSVCR110.dll!memcpy

OS_VERSION:  10.0.18362.1

BUILDLAB_STR:  19h1_release

OSPLATFORM_TYPE:  x86

OSNAME:  Windows 10

IMAGE_VERSION:  11.0.51106.1

FAILURE_ID_HASH:  {77975e19-9d4d-daf1-6c0e-6a3a4c334a80}

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

Timeline

2020-10-27 - Vendor Disclosure
2021-02-05 - Vendor Patched
2021-02-09 - Public Release

CREDIT

Discovered by Emmanuel Tacheau of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2020-1176

NEXT REPORT TALOS-2020-1196