Talos Vulnerability Report

TALOS-2020-1183

Accusoft ImageGear GIF LZW decoder heap overflow vulnerability

FEBRUARY 5, 2021

CVE NUMBER

CVE-2020-13572

Summary

A heap overflow vulnerability exists in the way the GIF parser decodes LZW compressed streams in Accusoft ImageGear 19.8. A specially crafted malformed file can trigger a heap overflow, which can result in arbitrary code execution. An attacker can provide a malicious file to trigger this vulnerability.

Tested Versions

Accusoft ImageGear Accusoft ImageGear 19.8

Product URLs

ImageGear - 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-122 - Heap-based Buffer Overflow

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.

ImageGear implements a decoder for GIF file format. Lack of bounds checking can lead to heap based buffer overflow while parsing GIF images with specially crafted image data.

GIF image is composed of a number of different headers and structures, most important of which is `DATA` block which in turn contains `ImageDescriptor` and `ImageData` blocks. For the purposes of this vulnerability, it is important to note that `ImageData` block can contain a number of different subblocks which contain LZW compressed data.

Opening the supplied proof of concept GIF image in ImageGear leads to the following crash:

```
(690.1130): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=1094d002 ebx=00004548 ecx=00000006 edx=1334aab8 esi=00000009 edi=fffffff8
eip=6ad73f5f esp=012fe884 ebp=012fea10 iopl=0         nv up ei ng nz ac po cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010293
igLZW19d+0x3f5f:
6ad73f5f 880413          mov     byte ptr [ebx+edx],al      ds:002b:1334f000=??
0:000> dd edx
1334aab8  02020202 02020202 02020202 02020202
1334aac8  02020202 02020202 02020202 02020202
1334aad8  02020202 02020202 02020202 02020202
1334aae8  02020202 02020202 02020202 02020202
1334aaf8  02020202 02020202 02020202 02020202
1334ab08  02020202 02020202 02020202 02020202
1334ab18  02020202 02020202 02020202 02020202
1334ab28  02020202 02020202 02020202 02020202
0:000> !heap -p -a edx
    address 1334aab8 found in
    _DPH_HEAP_ROOT @ 5bf1000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                             132e2dd0:         1334aab8             4545 -         1334a000             6000
          unknown!fillpattern
    6b25abb0 verifier!AVrfDebugPageHeapAllocate+0x00000240
    77cd245b ntdll!RtlDebugAllocateHeap+0x00000039
    77c36dd9 ntdll!RtlpAllocateHeap+0x000000f9
    77c35ec9 ntdll!RtlpAllocateHeapInternal+0x00000179
    77c35d3e ntdll!RtlAllocateHeap+0x0000003e
    6ad9dcff MSVCR110!malloc+0x00000049
    6af661de igCore19d!AF_memm_alloc+0x0000001e
    6ad73b1e igLZW19d+0x00003b1e
    6ad738b4 igLZW19d+0x000038b4
    6ad7254d igLZW19d+0x0000254d
    6af410d9 igCore19d!IG_image_savelist_get+0x00000b29
    6af80557 igCore19d!IG_mpi_page_set+0x00014807
    6af7feb9 igCore19d!IG_mpi_page_set+0x00014169
    6af15777 igCore19d!IG_load_file+0x00000047
```

From the above debugger output, we can observe a couple of things. First, the crash is due to access violation while writing to invalid memory pointed to by `ebx+edx`. From heap debug information we can see that the memory buffer pointed to by `edx` is of size 0x4545 and that value of 0x4548 in `ebx` makes this memory write fall outside the bounds of the buffer. If we examine the code surrounding the point of crash , we see the following:

```
.text:10003F50 loc_10003F50:                           ; CODE XREF: sub_10003A50+4F8↑j
.text:10003F50                                         ; sub_10003A50+53B↓j
.text:10003F50                 mov     eax, [ebp+var_130]
.text:10003F56                 mov     edx, [ebp+dest_buffer]                          [1]
.text:10003F5C                 mov     al, [ecx+eax]
.text:10003F5F                 mov     [ebx+edx], al                                   [2]
.text:10003F62                 mov     eax, [ebp+eax_buffer_unknown]                   [3]
.text:10003F68                 mov     edx, [ebp+var_13C]
.text:10003F6E                 movzx   ecx, word ptr [eax+ecx*2]                       [4]
.text:10003F72                 inc     ebx                                             [5]
.text:10003F73                 mov     [ebp+var_140], ebx
.text:10003F79                 cmp     ecx, 0FFFFh                                     [6]
.text:10003F7F                 jz      loc_10004172
.text:10003F85                 cmp     ecx, 0FFFEh
.text:10003F8B                 jnz     short loc_10003F50                              [7]
```

First of all, we can see that the above code constitutes a copy loop of some kind as conditional jump at [7] points to the begining of the block. At [1], address of destination buffer is retrieved into `edx` and a byte value from `al` is written into `edx` buffer at offset `ebx` at [2]. At [3], pointer to another, source, buffer is retrieved which is then used to set value of ecx at [4]. We can also observe that copy index , the value in `ebx`, is incremented by one in each iteration of the loop at [5]. Finally, the only ways to break out of the loop are at [6] and [7] where value in `ecx` is compared against 0xFFFF or 0xFFFE, if neither of these is true, the loop continues.

From analysis of the attached proof of concept file, we can conclude that heap overflow happens while processing compressed LZW stream because as we can observe the source buffer pointed to by `eax` being populated while decoding.

The vulnerability stems from the fact that above loop makes no checks to make sure that the copy index in `ebx` is smaller than the size of the buffer. Once the loop is entered, unless it's terminated by 0xFFFF or 0xFFFE in `ecx`, the index in `ebx` will continue to grow past the bounds of the buffer which leads to heap buffer overflow as observed in the above crash.

Additionally, from PoC file analysis, we can show that overflown buffer size is under direct control and in fact comes from `ImageWidth` value in `ImageDescriptor` field. With precise control over size of destination buffer and precise control over results of LZW stream decoding it is possible to overwrite adjacent heap memory causing futher memory corruption which can ultimately lead to arbitrary code execution.

Crash Information

```
    0:000> !analyze -v
*******************************************************************************
*                                                                             *
*                        Exception Analysis                                   *
*                                                                             *
*******************************************************************************
DUMP_CLASS: 2
DUMP_QUALIFIER: 0
FAULTING_IP:
igLZW19d+3f5f
6ad73f5f 880413          mov     byte ptr [ebx+edx],al
EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 6ad73f5f (igLZW19d+0x00003f5f)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 00000001
   Parameter[1]: 1334f000
Attempt to write to address 1334f000
FAULTING_THREAD:  00001130
FOLLOWUP_IP:
igLZW19d+3f5f
6ad73f5f 880413          mov     byte ptr [ebx+edx],al
WRITE_ADDRESS:  1334f000
ERROR_CODE: (NTSTATUS) 0xc0000005 - The instruction at 0x%p referenced memory at 0x%p. The memory could not be %s.
EXCEPTION_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:  1334f000
WATSON_BKT_PROCSTAMP:  5f732f32
WATSON_BKT_PROCVER:  1.0.0.2
PROCESS_VER_PRODUCT:  Fuzzme
WATSON_BKT_MODULE:  igLZW19d.dll
WATSON_BKT_MODSTAMP:  5f3ec43d
WATSON_BKT_MODOFFSET:  3f5f
WATSON_BKT_MODVER:  19.8.0.0
MODULE_VER_PRODUCT:  Accusoft ImageGear
BUILD_VERSION_STRING:  17134.1.x86fre.rs4_release.180410-1804
MODLIST_WITH_TSCHKSUM_HASH:  e753365c5948f36a0d0dda7434e6452388200aaa
MODLIST_SHA1_HASH:  80c4e65037a99371ebd47e3f748fe58ab869701f
NTGLOBALFLAG:  2100000
APPLICATION_VERIFIER_FLAGS:  0
PRODUCT_TYPE:  1
SUITE_MASK:  272
DUMP_TYPE:  fe
APPLICATION_VERIFIER_LOADED: 1
PROCESS_NAME:  unknown
ANALYSIS_SESSION_TIME:  10-23-2020 18:10:21.0042
ANALYSIS_VERSION: 10.0.17763.1 x86fre
THREAD_ATTRIBUTES:
OS_LOCALE:  ENU
BUGCHECK_STR:  APPLICATION_FAULT_INVALID_POINTER_WRITE_AVRF
DEFAULT_BUCKET_ID:  INVALID_POINTER_WRITE_AVRF
PRIMARY_PROBLEM_CLASS:  APPLICATION_FAULT
PROBLEM_CLASSES:
    ID:     [0n313]
    Type:   [@ACCESS_VIOLATION]
    Class:  Addendum
    Scope:  BUCKET_ID
    Name:   Omit
    Data:   Omit
    PID:    [Unspecified]
    TID:    [0x1130]
    Frame:  [0] : igLZW19d
    ID:     [0n286]
    Type:   [INVALID_POINTER_WRITE]
    Class:  Primary
    Scope:  DEFAULT_BUCKET_ID (Failure Bucket ID prefix)
            BUCKET_ID
    Name:   Add
    Data:   Omit
    PID:    [Unspecified]
    TID:    [0x1130]
    Frame:  [0] : igLZW19d
    ID:     [0n98]
    Type:   [AVRF]
    Class:  Addendum
    Scope:  DEFAULT_BUCKET_ID (Failure Bucket ID prefix)
            BUCKET_ID
    Name:   Add
    Data:   Omit
    PID:    [0x690]
    TID:    [0x1130]
    Frame:  [0] : igLZW19d
LAST_CONTROL_TRANSFER:  from 6ad738b4 to 6ad73f5f
STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
012fea10 6ad738b4 012ff55c 10000026 0bed9ff0 igLZW19d+0x3f5f
012fea60 6ad7254d 012ff55c 10000026 0bed9ff0 igLZW19d+0x38b4
012ff4d4 6af410d9 012ff55c 0bed9ff0 00000001 igLZW19d+0x254d
012ff50c 6af80557 00000000 0bed9ff0 012ff55c igCore19d!IG_image_savelist_get+0xb29
012ff788 6af7feb9 00000000 0a086fe0 00000001 igCore19d!IG_mpi_page_set+0x14807
012ff7a8 6af15777 00000000 0a086fe0 00000001 igCore19d!IG_mpi_page_set+0x14169
012ff7c8 00bc20d0 0a086fe0 012ff7dc 09fd4fc0 igCore19d!IG_load_file+0x47


STACK_COMMAND:  ~0s ; .cxr ; kb
THREAD_SHA1_HASH_MOD_FUNC:  2ce14bea4e604d248b5b15a2510c032aa5d1921b
THREAD_SHA1_HASH_MOD_FUNC_OFFSET:  38dd752a58275002ed88ae16bc5ae5f26c0f650a
THREAD_SHA1_HASH_MOD:  76476422c95de9d51201ea0d5862c350bd728e86
FAULT_INSTR_CODE:  8b130488
SYMBOL_STACK_INDEX:  0
SYMBOL_NAME:  igLZW19d+3f5f
FOLLOWUP_NAME:  MachineOwner
MODULE_NAME: igLZW19d
IMAGE_NAME:  igLZW19d.dll
DEBUG_FLR_IMAGE_TIMESTAMP:  5f3ec43d
FAILURE_BUCKET_ID:  INVALID_POINTER_WRITE_AVRF_c0000005_igLZW19d.dll!Unknown
BUCKET_ID:  APPLICATION_FAULT_INVALID_POINTER_WRITE_AVRF_igLZW19d+3f5f
FAILURE_EXCEPTION_CODE:  c0000005
FAILURE_IMAGE_NAME:  igLZW19d.dll
BUCKET_ID_IMAGE_STR:  igLZW19d.dll
FAILURE_MODULE_NAME:  igLZW19d
```

```
BUCKET_ID_MODULE_STR:  igLZW19d
FAILURE_FUNCTION_NAME:  Unknown
BUCKET_ID_FUNCTION_STR:  Unknown
BUCKET_ID_OFFSET:  3f5f
BUCKET_ID_MODTIMEDATESTAMP:  5f3ec43d
BUCKET_ID_MODCHECKSUM:  18bd0
BUCKET_ID_MODVER_STR:  19.8.0.0
BUCKET_ID_PREFIX_STR:  APPLICATION_FAULT_INVALID_POINTER_WRITE_AVRF_
FAILURE_PROBLEM_CLASS:  APPLICATION_FAULT
FAILURE_SYMBOL_NAME:  igLZW19d.dll!Unknown
TARGET_TIME:  2020-10-23T16:10:25.000Z
OSBUILD:  17134
OSSERVICEPACK:  753
SERVICEPACK_NUMBER: 0
OS_REVISION: 0
OSPLATFORM_TYPE:  x86
OSNAME:  Windows 10
OSEDITION:  Windows 10 WinNt SingleUserTS
USER_LCID:  0
OSBUILD_TIMESTAMP:  1998-02-05 12:31:21
BUILDDATESTAMP_STR:  180410-1804
BUILDLAB_STR:  rs4_release
BUILDOSVER_STR:  10.0.17134.1.x86fre.rs4_release.180410-1804
ANALYSIS_SESSION_ELAPSED_TIME:  63bc
ANALYSIS_SOURCE:  UM
FAILURE_ID_HASH_STRING:  um:invalid_pointer_write_avrf_c0000005_iglzw19d.dll!unknown
FAILURE_ID_HASH:  {287bbd66-8a9f-e2d7-91d6-15aac92a66ff}
Followup:     MachineOwner
---------
```

Timeline

2020-10-29 - Vendor Disclosure

2021-02-05 - Vendor Patched

2021-02-09 - Public Release

CREDIT

Discovered by Emmanuel Tacheau and Marcin Towalski of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2020-1008

NEXT REPORT TALOS-2020-1196