Talos Vulnerability Report

TALOS-2021-1362

Accusoft ImageGear DecoderStream::Append heap-based buffer overflow vulnerability

FEBRUARY 23, 2022

CVE NUMBER

CVE-2021-21914

Summary

A heap-based buffer overflow vulnerability exists in the DecoderStream::Append functionality of Accusoft ImageGear 19.10. A specially-crafted file can lead to code execution. An attacker can provide a malicious file to trigger this vulnerability.

Tested Versions

Accusoft ImageGear 19.10

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.

A specially-crafted JPEG 2000 file can lead to a heap-based buffer overflow in `DecoderStream::Append`, due to a wrongly sized heap buffer caused by an integer overflow.

Trying to load a malformed JPEG 2000 file, we end up in the following situation:

```
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=41414141 ebx=ffffb007 ecx=029ebf58 edx=00000000 esi=00004ff8 edi=029ebf58
eip=6e982f83 esp=0019f820 ebp=029ecf10 iopl=0         nv up ei ng nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010282
igJPEG2K19d!CPb_JPEG2K_init+0x2e2c3:
6e982f83 ff10            call    dword ptr [eax]      ds:002b:41414141=????????
```

This write access violation is happening in the function `read_data_from_file`, the second function called by `DecoderStream::Append`:

```
undefined * __thiscall read_data_from_file(inner_struct_0x58 *this,byte *dst_buffer,uint read_size)

  {
    int iVar1;
    byte *current_file_position;
    byte *remaining_data_to_read;
    uint local_4;

    current_file_position = this->heap_mem_file_content_current_pos_ptr;
    remaining_data_to_read = this->heap_mem_file_content_end_of_file_ptr + -(int)current_file_position
    ;
    local_4 = 0;
    if (remaining_data_to_read <= read_size) {
      do {
        if (remaining_data_to_read != (byte *)0x0) {
          memcpy(dst_buffer,current_file_position,(size_t)remaining_data_to_read);                          [1]
          this->heap_mem_file_content_current_pos_ptr =
              this->heap_mem_file_content_current_pos_ptr + (int)remaining_data_to_read;
          local_4 = (uint)(remaining_data_to_read + local_4);
          dst_buffer = dst_buffer + (int)remaining_data_to_read;
          read_size = read_size + -(int)remaining_data_to_read;
        }
        if (((byte *)read_size == (byte *)0x0) ||
          (iVar1 = (*(code *)*this->PTR_FUN_ARRAY)(), iVar1 == 0)) {                  [6]
          return (byte *)local_4;
        }
        current_file_position = this->heap_mem_file_content_current_pos_ptr;
        remaining_data_to_read =
            this->heap_mem_file_content_end_of_file_ptr + -(int)current_file_position;
      } while (remaining_data_to_read <= read_size);
    }
    if ((byte *)read_size != (byte *)0x0) {
      memcpy(dst_buffer,this->heap_mem_file_content_current_pos_ptr,read_size);
      this->heap_mem_file_content_current_pos_ptr =
          this->heap_mem_file_content_current_pos_ptr + read_size;
      local_4 = read_size + local_4;
    }
    return (undefined *)local_4;
  }
```

The `read_data_from_file` is called in `DecoderStream::Append` shown here:

```
uint __thiscall
DecoderStream::Append(int this,inner_struct_0x58 *inner_0x58,uint bytes_size,ushort param3)

{
  [...]

  if (bytes_size != 0) {
    allocation_res = malloc_mem_wrap(*(int *)(this + 8),(int **)(this + 0x28),param3,bytes_size);          [2]
                    /* allocation_res[2] still reside in allocated memory */
    n_bytes_read = read_data_from_file(inner_0x58,(byte *)allocation_res[2],bytes_size);                   [4]
    if (n_bytes_read != (undefined *)bytes_size) {
      if (n_bytes_read < bytes_size) {
        memset((void *)((int)allocation_res[2] + (int)n_bytes_read),0xff,
               bytes_size - (int)n_bytes_read);                                                           [5]
      }
      local_14 = 0xfffffbff;
      local_10 = "DecoderStream::Append";
      local_c = 0x8c;
      local_8 = "..\\..\\..\\iostream\\decoderstream.cpp";
      local_4 = "unexpected EOF on pulling encoded data";
      uVar1 = FUN_73ecf6a0(*(int *)(this + 8),&local_14);
      return uVar1 & 0xffffff00;
    }
  }
  return CONCAT31((int3)((uint)n_bytes_read >> 8),1);
}
```

The heap buffer in which the heap overflow takes place is allocated in `[2]`. The function called at `[2]` allocates a buffer with a size related to `bytes_size`, which is directly read from the file. This buffer is than populated, with data taken from the file, in `[4]` . If the buffer is not full after `[4]`, the remaining available space would be filled up with `0xff` in `[5]`.

The first instructions of `malloc_mem_wrap`, the function called in `[2]`, are shown here:

```
PUSH       ECX
PUSH       EBX
PUSH       EBP
MOV        EBP,dword ptr [ESP + bytes_size]
PUSH       ESI
PUSH       EDI
PUSH       ECX
LEA        EDI,[EBP + 0x1c]                                                                               [3]
PUSH       EDI
MOV        ESI,param_2
MOV        EBX,ECX
CALL       Environ::AllocMem
[...]
```

An integer oveflow could occur in `[3]` because of the sum of `bytes_size` and `0x1c`, leading to allocate a wrongly sized buffer. This could lead to a heap-based buffer oveflow in `[1]` and/or `[5]`, which can result in remote code execution.

The exception, shown previosly, is the consequence of exploiting the wrongly sized allocated memory due to the integer overflow in `[1]`. In this specific case this led to overwriting an object's field, living in the heap, that is used to fetch a function array pointer, then used in `[6]`.

Crash Information

crash output:

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


KEY_VALUES_STRING: 1

    Key  : AV.Dereference
    Value: String

    Key  : AV.Fault
    Value: Read

    Key  : Analysis.CPU.mSec
    Value: 2764

    Key  : Analysis.DebugAnalysisManager
    Value: Create

    Key  : Analysis.Elapsed.mSec
    Value: 14845

    Key  : Analysis.Init.CPU.mSec
    Value: 687

    Key  : Analysis.Init.Elapsed.mSec
    Value: 1088838

    Key  : Analysis.Memory.CommitPeak.Mb
    Value: 133

    Key  : Timeline.OS.Boot.DeltaSec
    Value: 24729

    Key  : Timeline.Process.Start.DeltaSec
    Value: 1088

    Key  : WER.OS.Branch
    Value: rs5_release

    Key  : WER.OS.Timestamp
    Value: 2018-09-14T14:34:00Z

    Key  : WER.OS.Version
    Value: 10.0.17763.1

    Key  : WER.Process.Version
    Value: 1.0.1.1


NTGLOBALFLAG:  470

APPLICATION_VERIFIER_FLAGS:  0

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 6e982f83 (igJPEG2K19d!CPb_JPEG2K_init+0x0002e2c3)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 00000000
   Parameter[1]: 41414141
Attempt to read from address 41414141

FAULTING_THREAD:  000027cc

PROCESS_NAME:  Fuzzme.exe

READ_ADDRESS:  41414141

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:  00000000

EXCEPTION_PARAMETER2:  41414141

STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
0019f830 6e9c82c6     029e7f18 ffffffff 029f1ea8 igJPEG2K19d!CPb_JPEG2K_init+0x2e2c3
0019f848 6e8e2ebb     0019f960 6e8e12b4 00000048 igJPEG2K19d!CPb_JPEG2K_init+0x73606
0019f850 6e8e12b4     00000048 02a2b308 6e99cd17 igJPEG2K19d+0x2ebb
0019f960 6e97d73e     029e0f78 02a2afc8 00000055 igJPEG2K19d+0x12b4
0019f980 6e974109     029e0f78 02a2afc8 02a2c458 igJPEG2K19d!CPb_JPEG2K_init+0x28a7e
0019fa58 6e96f55c     029e0f78 02a2afc8 007ebd78 igJPEG2K19d!CPb_JPEG2K_init+0x1f449
0019faac 6e96cccd     02a2afc8 02a2aec8 02a2b044 igJPEG2K19d!CPb_JPEG2K_init+0x1a89c
0019fb10 6e95dbba     02a2afc8 8c153897 007ebd78 igJPEG2K19d!CPb_JPEG2K_init+0x1800d
0019fb54 6e961059     00000000 02a2aec8 00000000 igJPEG2K19d!CPb_JPEG2K_init+0x8efa
0019fb70 6e9570a6     007ebd78 02a2aec8 8c15385f igJPEG2K19d!CPb_JPEG2K_init+0xc399
0019fb9c 6e95711e     0019fc3c 007ebd00 00000000 igJPEG2K19d!CPb_JPEG2K_init+0x23e6
0019fbb4 6ef013d9     0019fc3c 007ebd00 00000001 igJPEG2K19d!CPb_JPEG2K_init+0x245e
0019fbec 6ef408d7     00000000 007ebd00 0019fc3c igCore19d!IG_image_savelist_get+0xb29
0019fe68 6ef40239     00000000 007177a0 00000001 igCore19d!IG_mpi_page_set+0x148a7
0019fe88 6eed5757     00000000 007177a0 00000001 igCore19d!IG_mpi_page_set+0x14209
0019fea8 00402219     007177a0 0019febc 00000001 igCore19d!IG_load_file+0x47
0019fec0 00402524     007177a0 007177e8 00717ad0 Fuzzme!fuzzme+0x19
0019ff28 0040668d     00000005 00716200 00717ad0 Fuzzme!fuzzme+0x324
0019ff70 75b90419     002fd000 75b90400 0019ffdc Fuzzme!fuzzme+0x448d
0019ff80 77a072ed     002fd000 fead64a1 00000000 KERNEL32!BaseThreadInitThunk+0x19
0019ffdc 77a072bd     ffffffff 77a265b0 00000000 ntdll!__RtlUserThreadStart+0x2f
0019ffec 00000000     00406715 002fd000 00000000 ntdll!_RtlUserThreadStart+0x1b


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  igJPEG2K19d!CPb_JPEG2K_init+2e2c3

MODULE_NAME: igJPEG2K19d

IMAGE_NAME:  igJPEG2K19d.dll

FAILURE_BUCKET_ID:  INVALID_POINTER_READ_FILL_PATTERN_41414141_c0000005_igJPEG2K19d.dll!CPb_JPEG2K_init

OS_VERSION:  10.0.17763.1
```

```
BUILDLAB_STR:  rs5_release

OSPLATFORM_TYPE:  x86

OSNAME:  Windows 10

IMAGE_VERSION:  25.1.0.0

FAILURE_ID_HASH:  {79891e7d-f1f3-59e0-064a-4dbbd8234ed4}

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

Timeline

2021-08-23 - Initial contact
2021-08-24 - Vendor acknowledged and created support ticket
2021-10-29 - 60 day follow up
2021-11-30 - Vendor investigating status
2021-12-02 - Vendor advised release planned for Q1 2022
2021-12-07 - 30 day disclosure extension granted (2022-01-24)
2022-01-06 - Final disclosure notification
2022-02-23 - Public disclosure

CREDIT

Discovered by Francesco Benvenuto of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2021-1364

NEXT REPORT TALOS-2021-1367

© 2022 Cisco Systems, Inc. and/or its affiliates. All rights reserved. View our Privacy Policy.