Talos Vulnerability Report

TALOS-2021-1368

Accusoft ImageGear XWD parser heap-based buffer overflow vulnerability

FEBRUARY 23, 2022

CVE NUMBER

CVE-2021-21939

Summary

A heap-based buffer overflow vulnerability exists in the XWD parser 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-120 - Buffer Copy without Checking Size of Input (‘Classic 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 XWD file can lead to a heap-based buffer overflow in the XWD parser, due to a missing size check.

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

```
(27a8.2444): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=0bb36b30 ecx=00000075 edx=00000001 esi=0bb36b30 edi=0a90aff0
eip=6ef7e280 esp=0019f608 ebp=0019f624 iopl=0         nv up ei pl nz na po cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010203
MSVCR110!memcpy+0x567:
6ef7e280 660f7f4f10      movdqa  xmmword ptr [edi+10h],xmm1 ds:002b:0a90b000=????????????????????????????????
```

The access violation is originated at `[3]` in the `xwdread_read_bitmap` function:

```
void xwdread_read_bitmap(mys_table_function *param_1,uint param_2,XWD_read *XWDcontent,
                        undefined4 param_4,HIGDIBINFO param_5)

{


  BytesPerLine = (XWDcontent->header_data).BytesPerLine;
  local_38 = 0;
  local_3c = 0;
  local_44 = 0;
  local_24 = 0;
  local_14 = 0;
  local_20 = 0;
  local_c = BytesPerLine;
  bit_depth = IGDIBStd::DIB_bit_depth_get(param_5);
  if (bit_depth == 1) {
    dst_buff_size = IO_raster_size_get(param_5);
  }
  else {
    dst_buff_size = DIBStd_raster_size_get(param_5);                                            [1]
  }
  bitsPerPixel = (XWDcontent->header_data).BitsPerPixel;
  PixmapWidth = DIB_width_get(param_5);
  local_7c.size_buffer = DIB_height_get(param_5);

  [...]

  IOb_init(param_1,param_2,&local_7c,BytesPerLine * 5,1);
  lplValue1 = dst_buff_size;
  dst_buff = (byte *)AF_memm_alloc(param_2,dst_buff_size);                                       [2]
  if ((dst_buff != (byte *)0x0) ||
     (AVar3 = AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\xwdread.c",0x3ad,-1000,0,lplValue1
                                ,param_2,(LPCHAR)0x0), AVar3 == 0)) {
    local_8 = 0;
    if (0 < (int)local_7c.size_buffer) {
      while ((iVar6 = local_8, puVar4 = (uint *)get_data_from_file(&local_7c,BytesPerLine),
             local_1c = puVar4, puVar4 != (uint *)0x0 ||
             (AVar3 = AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\xwdread.c",0x3b6,-0x803,0,
                                        BytesPerLine,param_2,(LPCHAR)0x0), AVar3 == 0))) {
        buff = puVar4;

        [...]

        bit_depth = IGDIBStd::DIB_bit_depth_get(param_5);
        if (true) {
          switch(bit_depth) {
          case 1:
          case 4:
          case 8:
            OS_memcpy(dst_buff,buff,BytesPerLine);                                               [3]
            break;

          [...]
}
```

The `OS_memcpy` function at `[3]` is a `memcpy` wrapper, so `BytesPerLine` bytes from `buff` are copied to `dst_buff`. The `BytesPerLine` value and `buff`'s content are taken directly from the XWD file. The destination buffer `dst_buff` is allocated at `[2]` using `dst_buff_size` as size.

The function `DIBStd_raster_size_get`, that computes the `dst_buff_size` value at `[1]`, is shown here:

```
AT_INT DIBStd_raster_size_get(HIGDIBINFO hdib)

    {
    [...]
    uVar1 = (*hdib->igdibstd_vftable->IGDIBStd::compute_raster_size)((IGDIBStd *)hdib);
    [...]
    return uVar1;
    }
```

The function call `compute_raster_size`:

```
uint __fastcall IGDIBStd::compute_raster_size(IGDIBStd *param_1)

{
longlong lVar1;
uint uVar2;
ulonglong uVar3;

lVar1 = (longlong)(int)(param_1->mys_struct_table_color).ptr_bits_per_channel_table *
        (longlong)(int)(param_1->mys_struct_table_color).ptr_channel_count;                     [4]
uVar3 = __allmul((uint)lVar1,(uint)((ulonglong)lVar1 >> 0x20),param_1->size_X,
                 (int)param_1->size_X >> 0x1f);                                                  [5]
lVar1 = (longlong)(uVar3 + 0x1f) >> 3;
                     /* if __all_mul's params2/4 are 0 the results is simply param1*param3 */
uVar2 = (uint)lVar1 & 0xfffffffc;
if ((-1 < lVar1) && ((0 < (int)((longlong)(uVar3 + 0x1f) >> 0x23) || (0x7ffffffe < uVar2)))) {
    wrapper_thow_exception
            ((undefined *)0xfffffe6f,(char *)0x0,(undefined *)0x0,(undefined *)0x0,
            (undefined **)0x1022fa38,(undefined *)0x29);
}
return uVar2;
}
```

The calculation of `dst_buff_size` is based on the variable used in `[4]` and `[5]`, where the first is dependent on the `PixmapDepth` and the latter is dependent on the `PixmapWidth`. The returned value is then used at `[2]` to allocate the `dst_buff` that is used as a temporary buffer to store, one at the time, the content of the bitmap's "rows". The problem is that neither `PixmapWidth` nor the calculated size are ever compared with the `BytesPerLine` variable. This allows the allocation of less space than required, leading to a heap-based buffer overflow.

Crash Information

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


KEY_VALUES_STRING: 1

    Key  : AV.Fault
    Value: Write

    Key  : Analysis.CPU.mSec
    Value: 2687

    Key  : Analysis.DebugAnalysisManager
    Value: Create

    Key  : Analysis.Elapsed.mSec
    Value: 10478

    Key  : Analysis.Init.CPU.mSec
    Value: 1015

    Key  : Analysis.Init.Elapsed.mSec
    Value: 63464

    Key  : Analysis.Memory.CommitPeak.Mb
    Value: 139

    Key  : Timeline.OS.Boot.DeltaSec
    Value: 206950

    Key  : Timeline.Process.Start.DeltaSec
    Value: 62

    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:  2000000

APPLICATION_VERIFIER_FLAGS:  0

APPLICATION_VERIFIER_LOADED: 1

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 6ef7e280 (MSVCR110!memcpy+0x00000567)
ExceptionCode: c0000005 (Access violation)
ExceptionFlags: 00000000
NumberParameters: 2
Parameter[0]: 00000001
Parameter[1]: 0a90b000
Attempt to write to address 0a90b000

FAULTING_THREAD:  00002444

PROCESS_NAME:  Fuzzme.exe

WRITE_ADDRESS:  0a90b000

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:  0a90b000

STACK_TEXT:
0019f610 6e18f9a6     0a90aff0 0bb36b30 000000f5 MSVCR110!memcpy+0x567
WARNING: Stack unwind information not available. Following frames may be wrong.
0019f624 6e3247d2     0a90aff0 0bb36b30 000000f5 igCore19d+0xf9a6
0019f6bc 6e325165     0019fc3c 1000001e 0019f718 igCore19d!IG_mpi_page_set+0x1387a2
0019f6e4 6e323dfb     0019fc3c 1000001e 0a338ff8 igCore19d!IG_mpi_page_set+0x139135
0019fbb4 6e1c13d9     0019fc3c 0a338ff8 00000001 igCore19d!IG_mpi_page_set+0x137dcb
0019fbec 6e2008d7     00000000 0a338ff8 0019fc3c igCore19d!IG_image_savelist_get+0xb29
0019fe68 6e200239     00000000 052d7fc0 00000001 igCore19d!IG_mpi_page_set+0x148a7
0019fe88 6e195757     00000000 052d7fc0 00000001 igCore19d!IG_mpi_page_set+0x14209
0019fea8 00402219     052d7fc0 0019febc 00000001 igCore19d!IG_load_file+0x47
0019fec0 00402524     052d7fc0 052d9fe0 0523df50 Fuzzme!fuzzme+0x19
0019ff28 0040668d     00000005 05236f38 0523df50 Fuzzme!fuzzme+0x324
0019ff70 75330419     0029e000 75330400 0019ffdc Fuzzme!fuzzme+0x448d
0019ff80 778b72ed     0029e000 58b504f3 00000000 KERNEL32!BaseThreadInitThunk+0x19
0019ffdc 778b72bd     ffffffff 778d65b3 00000000 ntdll!__RtlUserThreadStart+0x2f
0019ffec 00000000     00406715 0029e000 00000000 ntdll!_RtlUserThreadStart+0x1b


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  MSVCR110!memcpy+567

MODULE_NAME: MSVCR110

IMAGE_NAME:  MSVCR110.dll

FAILURE_BUCKET_ID:  INVALID_POINTER_WRITE_AVRF_c0000005_MSVCR110.dll!memcpy

OS_VERSION:  10.0.17763.1

BUILDLAB_STR:  rs5_release

OSPLATFORM_TYPE:  x86
```

```
OSNAME:  Windows 10

IMAGE_VERSION:  11.0.50727.1

FAILURE_ID_HASH:  {80e9803c-2e1f-2683-6c9b-fae163af54bc}

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

Timeline

2021-08-30 - Initial contact
2021-08-31 - Vendor acknowledged and created support ticket
2021-09-10 - Vendor closed support ticket and confirmed under review with engineering team
2021-11-30 - 60 day follow up
2021-12-01 - Vendor advised release planned for Q1 2022
2021-12-07 - 30 day disclosure extension granted
2022-01-06 - Final disclosure notification
2022-02-23 - Public disclosure

CREDIT

Discovered by Francesco Benvenuto of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT

TALOS-2021-1367

NEXT REPORT

TALOS-2021-1371