Talos Vulnerability Report

TALOS-2021-1373

Accusoft ImageGear XWD parser::xwdread_pixmapformat_0_or_1 heap-based buffer overflow vulnerability

FEBRUARY 23, 2022

CVE NUMBER

CVE-2021-21943

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-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 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:

```
(1b14.213c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=016304c8 ebx=000000f5 ecx=0000003b edx=00000001 esi=0aa34b38 edi=0bc05000
eip=6e9fe13d esp=0019f45c ebp=0019f474 iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010206
MSVCR110!memcpy+0x21e:
6e9fe13d f3a5           rep movs dword ptr es:[edi],dword ptr [esi]
```

The access violation is originated at `[3]` in the `xwdread_pixmapformat_0_or_1` function:

```
void xwdread_pixmapformat_0_or_1
            (mys_table_function *param_1,uint param_2,XWD_read *xwd,undefined4 param_4,
            HIGDIBINFO param_5)

{
[...]

local_8 = DAT_102bcea8 ^ (uint)&stack0xfffffffc;
local_1d4 = param_2;
bytePerLine = (xwd->header_data).BytesPerLine;
error_code = 0;
local_1dc = xwd;
local_1e0 = param_5;
local_1ac = 0;
bytePerLine_ = bytePerLine;
dVar2 = IGDIBStd::DIB_bit_depth_get(param_5);
if (dVar2 == 1) {
    dst_buff_size = IO_raster_size_get(param_5);                                    [1]
}
else {
    dst_buff_size = DIBStd_raster_size_get(param_5);
}
[...]

dst_buff = (byte *)AF_memm_alloc(local_1d4,dst_buff_size);                          [2]

[...]
if ((error_code == 0) && (local_1d0 = error_code, height = DIB_height_get(local_1e0), 0 < height)) {
    do {
    depth_done = 0;
    if (pixmapDepth_ != 0) {
        io_buffer = local_1a8;
        do {
        src_buff = (byte *)get_data_from_file(io_buffer,bytePerLine_);
        array_of_source_buff[depth_done] = src_buff;
        if (src_buff == (byte *)0x0) {
            local_1ac = AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\xwdread.c",0x2fc,-0x803,
                                    0,bytePerLine_,local_1d4,(LPCHAR)0x0);
            uVar11 = local_1d4;
            if (local_1ac != 0) goto LAB_101a44d0;
        }
        depth_done = depth_done + 1;
        piVar8 = (io_buffer *)&piVar8->size_buffer;
        } while (depth_done < pixmapDepth_);
        uVar11 = local_1d4;
        if (local_1ac != 0) break;
    }
    bytePerLine__ = bytePerLine_;
    [...]
    bit_depth = IGDIBStd::DIB_bit_depth_get(local_1e0);
    if (bit_depth == 1) {
        OS_memcpy(dst_buff,array_of_source_buff[0],bytePerLine__);                  [3]
    }
```

The `OS_memcpy` function at `[3]` is a `memcpy` wrapper, so `BytesPerLine` bytes from `array_of_source_buff[0]` are copied to `dst_buff`. The `BytesPerLine` value and `array_of_source_buff[0]`'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 return value of the function `IO_raster_size_get`, that computes the `dst_buff_size` value at `[1]`, in this specific case, is essentially: `(PixmapWidth + 0x1f) >> 3) & 0xfffffffc`.

It is evident that the size of `dst_buff_size` is only dependent on the `PixmapWidth`. The returned value of `IO_raster_size_get` 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: 3046

    Key  : Analysis.DebugAnalysisManager
    Value: Create

    Key  : Analysis.Elapsed.mSec
    Value: 9918

    Key  : Analysis.Init.CPU.mSec
    Value: 499

    Key  : Analysis.Init.Elapsed.mSec
    Value: 59454

    Key  : Analysis.Memory.CommitPeak.Mb
    Value: 141

    Key  : Timeline.OS.Boot.DeltaSec
    Value: 38712

    Key  : Timeline.Process.Start.DeltaSec
    Value: 59

    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: 6e9fe13d (MSVCR110!memcpy+0x0000021e)
ExceptionCode: c0000005 (Access violation)
ExceptionFlags: 00000000
NumberParameters: 2
Parameter[0]: 00000001
Parameter[1]: 0bc05000
Attempt to write to address 0bc05000

FAULTING_THREAD:  0000213c

PROCESS_NAME:  Fuzzme.exe

WRITE_ADDRESS:  0bc05000

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:  0bc05000

STACK_TEXT:
0019f460 6ebff9a6     0bc04ff8 0aa34b30 000000f5 MSVCR110!memcpy+0x21e
WARNING: Stack unwind information not available. Following frames may be wrong.
0019f474 6ed94489     0bc04ff8 0aa34b30 000000f5 igCore19d+0xf9a6
0019f6bc 6ed9517f     0019fc3c 1000001e 0019f718 igCore19d!IG_mpi_page_set+0x138459
0019f6e4 6ed93dfb     0019fc3c 1000001e 0ad9cff8 igCore19d!IG_mpi_page_set+0x13914f
0019fbb4 6ec313d9     0019fc3c 0ad9cff8 00000001 igCore19d!IG_mpi_page_set+0x137dcb
0019fbec 6ec708d7     00000000 0ad9cff8 0019fc3c igCore19d!IG_image_savelist_get+0xb29
0019fe68 6ec70239     00000000 05287fd0 00000001 igCore19d!IG_mpi_page_set+0x148a7
0019fe88 6ec05757     00000000 05287fd0 00000001 igCore19d!IG_mpi_page_set+0x14209
0019fea8 00402219     05287fd0 0019febc 00000001 igCore19d!IG_load_file+0x47
0019fec0 00402524     05287fd0 05289fe0 051edf50 Fuzzme!fuzzme+0x19
0019ff28 0040668d     00000005 051e6f50 051edf50 Fuzzme!fuzzme+0x324
0019ff70 765f0419     0030c000 765f0400 0019ffdc Fuzzme!fuzzme+0x448d
0019ff80 777a72ed     0030c000 ccadecfe 00000000 KERNEL32!BaseThreadInitThunk+0x19
0019ffdc 777a72bd     ffffffff 777c65e6 00000000 ntdll!__RtlUserThreadStart+0x2f
0019ffec 00000000     00406715 0030c000 00000000 ntdll!_RtlUserThreadStart+0x1b


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  MSVCR110!memcpy+21e

MODULE_NAME: MSVCR110

IMAGE_NAME:  MSVCR110.dll

FAILURE_BUCKET_ID:  INVALID_POINTER_WRITE_STRING_DEREFERENCE_NXCODE_FILL_PATTERN_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:  {4dd4b8b0-04bb-4a7d-d82a-fdf37d88888e}

Followup:     MachineOwner
```

Timeline

2021-09-10 - Initial contact
2021-09-14 - Vendor acknowledged and created support ticket
2021-09-21 - Vendor closed support ticket and confirmed under review with engineering team
2021-11-30 - 60 day follow up
2021-12-02 - 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-2021-1371

NEXT REPORT TALOS-2021-1374