Talos Vulnerability Report

TALOS-2020-0998

Accusoft ImageGear PNG store_data_buffer size computation code execution vulnerability

MAY 5, 2020

CVE NUMBER

CVE-2020-6075

Summary

An exploitable out-of-bounds write vulnerability exists in the `store_data_buffer` function of the igcore19d.dll library of Accusoft ImageGear 19.5.0. A specially crafted PNG file can cause an out-of-bounds write, resulting in a remote code execution. An attacker needs to provide a malformed file to the victim to trigger the vulnerability.

Tested Versions

Accusoft ImageGear 19.5.0

Product URLs

https://www.accusoft.com/products/imagegear/overview/

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-194: Unexpected Sign Extension

Details

The ImageGear library is a document imaging developer toolkit providing all kinds of functionality related to image conversion, creation, editing, annotation, etc. It supports more than 100 formats, including many image formats, DICOM, PDF, Microsoft Office and others.

There is a vulnerability in the `store_data_buffer` function due to an invalid cast conversion. A specially crafted PNG file can lead to an out-of-bounds, write which can result in remote code execution.

Trying to load a malformed PNG file via `IG_load_file` function, we end up in the following situation:

```
eax=00004504 ebx=0f30e5f8 ecx=000000c0 edx=0000c0c0 esi=0f306fc9 edi=7fffc503
eip=670c0cb2 esp=009dd524 ebp=009dd530 iopl=0         nv up ei ng nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010286
igCore19d!IG_mpi_page_set+0xe5922:
670c0cb2 66891443        mov     word ptr [ebx+eax*2],dx  ds:002b:0f317000=????

0:000> kb
 # ChildEBP RetAddr  Args to Child
WARNING: Stack unwind information not available. Following frames may be wrong.
00 0019d520 64c01114 0ead65c0 0e3a45f8 ffff8a07 igCore19d!IG_mpi_page_set+0xe5922
01 0019d540 64bff981 0ead65c0 0e5fe5c0 0e3a45f8 igCore19d!IG_mpi_page_set+0xe5d84
02 0019f1cc 64c00c74 0019f71c 1000001b 099acfe8 igCore19d!IG_mpi_page_set+0xe45f1
03 0019f200 64bfe32c 0019f71c 1000001b 099acfe8 igCore19d!IG_mpi_page_set+0xe58e4
04 0019f694 64af07c9 0019f71c 099acfe8 00000001 igCore19d!IG_mpi_page_set+0xe2f9c
05 0019f6cc 64b2fb97 00000000 099acfe8 0019f71c igCore19d!IG_image_savelist_get+0xb29
06 0019f948 64b2f4f9 00000000 09505fa8 00000001 igCore19d!IG_mpi_page_set+0x14807
07 0019f968 64ac6007 00000000 09505fa8 00000001 igCore19d!IG_mpi_page_set+0x14169
08 0019f988 006059ac 09505fa8 0019fa74 0019fa98 igCore19d!IG_load_file+0x47
09 0019fa88 006061a7 09505fa8 0019fbbc 00000021 simple_exe_141+0x159ac
0a 0019fc54 00606cbe 00000005 094b2f50 09397f40 simple_exe_141+0x161a7
0b 0019fc68 00606b27 d2316a2d 006015e1 006015e1 simple_exe_141+0x16cbe
0c 0019fcc4 006069bd 0019fcd4 00606d38 0019fce4 simple_exe_141+0x16b27
0d 0019fccc 00606d38 0019fce4 75286359 002a8000 simple_exe_141+0x169bd
0e 0019fcd4 75286359 002a8000 75286340 0019fd40 simple_exe_141+0x16d38
0f 0019fce4 779c7b74 002a8000 63abcc02 00000000 KERNEL32!BaseThreadInitThunk+0x19
10 0019fd40 779c7b44 ffffffff 779e8f15 00000000 ntdll!__RtlUserThreadStart+0x2f
11 0019fd50 00000000 006015e1 002a8000 00000000 ntdll!_RtlUserThreadStart+0x1b
```

As we can see, an out-of-bounds write operation occurred.
The pseudo-code of this vulnerable function looks like this:

```
unsigned int __cdecl store_data_buffer(int src_buffer, int dst_buffer, int size)
{
  unsigned int index; // eax
  unsigned __int8 *v4; // esi

  index = 0;
  if ( (unsigned int)(size - 1) >> 1 )
  {
    _src_buffer = (unsigned __int8 *)(src_buffer + 1);
    do
    {
      *(_WORD *)(dst_buffer + 2 * index++) = __ROL2__(*_src_buffer | (_src_buffer[1] << 8), 8);    [1]

      _src_buffer += 2;
    }
    while ( index < (unsigned int)(size - 1) >> 1 );                                               [2]
  }
  return index;
}
```

In this algorithm we can observe a function `store_data_buffer`, whose objective is to copy the content of `src_buffer` into `dst_buffer`, is crashing while filling the buffer `dst_buffer` in [1].

The copy operation is controlled by a loop [2], with a range from `0` to `size-1`.

This is happening because the `dst_buffer` is too small compared to the size argument.

Let's see how the size of the target buffer and the size argument are computed.

```
unsigned __int8 __cdecl sub_670C1030(int src_data, int a2, void *buffer_mem, int a4, int size, int a6, int a7, int a8, int a9)
{
  unsigned __int8 param_7; // al
  unsigned int i; // ecx
  _BYTE *v11; // esi
  _BYTE *v12; // ebx
  unsigned int v13; // edx
  char v14; // cl
  char v15; // cl

  switch ( *(unsigned __int8 *)src_data )
  {
    case 1u:
      sub_670BE9C0(src_data, size, a6);
      break;
    case 2u:
      sub_670BEA10(src_data, a2, size);
      break;
    case 3u:
      sub_670BEA60(src_data, a2, size, (unsigned __int8)a6);
      break;
    case 4u:
      sub_670BEAF0(src_data, a2, size, a6);
      break;
    default:
      break;
  }
  param_7 = a7;
  switch ( *(unsigned __int8 *)(a7 + 9) )
  {
    case 0u:
      if ( *(_BYTE *)(a7 + 8) == 2 )
        goto LABEL_13;
      param_7 = *(_BYTE *)(a7 + 8) - 16;
      if ( *(_BYTE *)(a7 + 8) == 16 )
      {
        param_7 = sub_670BE850(src_data, buffer_mem, size);
      }
      else
      {
        for ( i = 1; i < size; ++i )
        {
          param_7 = *(_BYTE *)(i + src_data);
          *((char *)buffer_mem + i - 1) = param_7;
        }
      }
      break;
    case 2u:
      param_7 = *(_BYTE *)(a7 + 8);
      if ( param_7 == 8 )
      {
        if ( (unsigned int)size > 1 )
        {
          v11 = buffer_mem;
          v12 = (_BYTE *)(src_data + 2);
          v13 = (size - 2) / 3u + 1;
          do
          {
            *v11 = *(v12 - 1);
            v11[1] = *v12;
            param_7 = v12[1];
            v11[2] = param_7;
            v11 += 3;
            v12 += 3;
            --v13;
          }
          while ( v13 );
        }
      }
      else if ( param_7 == 16 )
      {
        param_7 = store_data_buffer(src_data, (int)buffer_mem, size);      [3]
      }
      break;
                [...]
}
```

The `store_data_buffer` is called from the function named `sub_670C1030` in [3] but we can see that `size` and `buffer_mem` are passed as arguments so we need to go back further. This leads us to the function `process_raster_png`:

```
LINE141 int __stdcall process_raster_png(table_function *a1, void *arg4, int a3, int a4, int a5, IGDIBOject *a6, int a7, int a8)
LINE142 {
LINE143   int v8; // esi
LINE144   size_t v10; // edi
LINE145   byte *v11; // edi
LINE146   unsigned int size_buffer_mem; // edi
LINE147   byte *buffer_mem; // ebx
LINE148   int v14; // esi
LINE149   int v15; // eax
LINE150   unsigned int v16; // edi
LINE151   byte *v17; // eax
LINE152   byte *v18; // edx
LINE153   unsigned int i; // ecx
LINE154   int *v20; // ecx
LINE155   int v21; // eax
LINE156   int v22; // esi
LINE157   size_t v23; // edx
LINE158   int v24; // esi
LINE159   char v25; // cl
LINE160   int v26; // ecx
LINE161   int v27; // esi
LINE162   unsigned __int8 v28; // al
LINE163   unsigned __int8 v29; // bl
LINE164   int v30; // ecx
LINE165   int v31; // eax
LINE166   __int16 v32; // ax
LINE167   char v33; // cl
LINE168   __int16 v34; // ax
LINE169   char v35; // al
LINE170   int v36; // esi
LINE171   int v37; // eax
LINE172   int v38; // edx
LINE173   int v39; // edi
LINE174   int v40; // esi
LINE175   int j; // ecx
LINE176   unsigned __int8 v42; // al
LINE177   char v43; // al
LINE178   bool v44; // zf
LINE179   int v45; // esi
LINE180   int k; // ecx
LINE181   byte v47; // al
LINE182   byte v48; // al
LINE183   int v49; // eax
LINE184   unsigned int l; // esi
LINE185   void *v51; // eax
LINE186   void *v52; // ebx
LINE187   int v53; // ebx
LINE188   byte *v54; // edi
LINE189   int v55; // esi
LINE190   char v56; // cl
LINE191   int v57; // esi
LINE192   __int16 v58; // [esp-4h] [ebp-1C64h]
LINE193   unsigned __int8 v59; // [esp+Ch] [ebp-1C54h]
LINE194   unsigned int v60; // [esp+10h] [ebp-1C50h]
LINE195   __int16 v61; // [esp+14h] [ebp-1C4Ch]
LINE196   png_struct *table_of_size; // [esp+1Ch] [ebp-1C44h]
LINE197   unsigned int v63; // [esp+20h] [ebp-1C40h]
LINE198   int v64; // [esp+24h] [ebp-1C3Ch]
LINE199   int a6a; // [esp+28h] [ebp-1C38h]
LINE200   int v66; // [esp+2Ch] [ebp-1C34h]
LINE201   int v67; // [esp+30h] [ebp-1C30h]
LINE202   int v68; // [esp+34h] [ebp-1C2Ch]
LINE203   int v69; // [esp+38h] [ebp-1C28h]
LINE204   size_t size; // [esp+3Ch] [ebp-1C24h]
LINE205   byte *v71; // [esp+40h] [ebp-1C20h]
LINE206   int *v72; // [esp+44h] [ebp-1C1Ch]
LINE207   unsigned int ___size; // [esp+4Ch] [ebp-1C14h]
LINE208   unsigned int v74; // [esp+50h] [ebp-1C10h]
LINE209   byte *a2; // [esp+54h] [ebp-1C0Ch]
LINE210   int v76; // [esp+58h] [ebp-1C08h]
LINE211   byte *v77; // [esp+5Ch] [ebp-1C04h]
LINE212   byte *Src; // [esp+60h] [ebp-1C00h]
LINE213   byte *_buffer_mem; // [esp+64h] [ebp-1BFCh]
LINE214   unsigned int _size; // [esp+68h] [ebp-1BF8h]
LINE215   size_t v81; // [esp+74h] [ebp-1BECh]
LINE216   int v82; // [esp+78h] [ebp-1BE8h]
LINE217   int v83; // [esp+644h] [ebp-161Ch]
LINE218   int v84; // [esp+1C3Ch] [ebp-24h]
LINE219   int v85; // [esp+1C40h] [ebp-20h]
LINE220   int v86; // [esp+1C44h] [ebp-1Ch]
LINE221   int v87; // [esp+1C48h] [ebp-18h]
LINE222   int v88; // [esp+1C4Ch] [ebp-14h]
LINE223   int v89; // [esp+1C50h] [ebp-10h]
LINE224   int v90; // [esp+1C54h] [ebp-Ch]
LINE225   int v91; // [esp+1C58h] [ebp-8h]
LINE226
LINE227   v84 = 0x200000F;
LINE228   v85 = 0x1000100;
LINE229   v86 = 0x4000F;
LINE230   v87 = 0x10002;
LINE231   v90 = 0x404040F;
LINE232   v91 = 0x1010202;
LINE233   v88 = 0x408080F;
LINE234   v89 = 0x1020204;
LINE235   v76 = 0;
LINE236   v8 = 0;
LINE237   table_of_size = (png_struct *)AF_memm_alloc((int)arg4, 48u, (int)"..\\..\\..\\..\\Common\\Formats\\pngread.c", 2934);
LINE238   if ( !table_of_size )
LINE239     return kind_of_print_error((int)"..\\..\\..\\..\\Common\\Formats\\pngread.c", 2938, -1000, 0, 48, (int)arg4, 0);
LINE240   _size = compute_raster_size(a4);
LINE241   v10 = _size - (_size & 0x3F) + 64;
LINE242   Src = AF_memm_alloc((int)arg4, v10, (int)"..\\..\\..\\..\\Common\\Formats\\pngread.c", 2948);
LINE243   if ( !Src )
LINE244   {
LINE245     sub_66FD60E0((int)arg4, (int)"..\\..\\..\\..\\Common\\Formats\\pngread.c", 2951);
LINE246     return kind_of_print_error((int)"..\\..\\..\\..\\Common\\Formats\\pngread.c", 2952, -1000, 0, _size, (int)arg4, 0);
LINE247   }
LINE248   v11 = AF_memm_alloc((int)arg4, v10, (int)"..\\..\\..\\..\\Common\\Formats\\pngread.c", 2958);
LINE249   v77 = v11;
LINE250   if ( !v11 )
LINE251   {
LINE252     sub_66FD60E0((int)arg4, (int)"..\\..\\..\\..\\Common\\Formats\\pngread.c", 2961);
LINE253     return kind_of_print_error((int)"..\\..\\..\\..\\Common\\Formats\\pngread.c", 2962, -1000, 0, _size, (int)arg4, 0);
LINE254   }
LINE255   v71 = AF_memm_alloc((int)arg4, _size, (int)"..\\..\\..\\..\\Common\\Formats\\pngread.c", 2967);
LINE256   if ( !v71 )
LINE257   {
  sub_66FD60E0((int)arg4, (int)"..\\..\\..\\..\\Common\\Formats\\pngread.c", 2970);
  return kind_of_print_error((int)"..\\..\\..\\..\\Common\\Formats\\pngread.c", 2971, -1000, 0, _size, (int)arg4, 0);
}
if ( _size )
  memset(v11, 0, _size);
size_buffer_mem = lead_to_compute_size_based_width_bits(a6);     [6]
_size = size_buffer_mem;
buffer_mem = AF_memm_alloc((int)arg4, size_buffer_mem, (int)"..\\..\\..\\..\\Common\\Formats\\pngread.c", 2981);       [5]
_buffer_mem = buffer_mem;
if ( !buffer_mem )
{
  sub_66FD60E0((int)arg4, (int)"..\\..\\..\\..\\Common\\Formats\\pngread.c", 2984);
  return kind_of_print_error(
           (int)"..\\..\\..\\..\\Common\\Formats\\pngread.c",
           2985,
           -1000,
           0,
           size_buffer_mem,
           (int)arg4,
           0);
}
wrapper_memset(&v81, 0, 0x1BC8u);
v82 = 2;
if ( *(_BYTE *)(a4 + 12) == 1 )
{
  v14 = 4 * ((getSizeY_0(a6) + 7) / 8);
  if ( v14 > 0xFFFF )
  {
    v15 = getSizeY_0(a6);
    return kind_of_print_error(
             (int)"..\\..\\..\\..\\Common\\Formats\\pngread.c",
             3023,
             -1005,
             0,
             v15,
             0,
             "Interlaced png image has too big heght. Can't load image.");
  }
  if ( size_buffer_mem > 0xFFFF )
    return kind_of_print_error(
             (int)"..\\..\\..\\..\\Common\\Formats\\pngread.c",
             3029,
             -1005,
             0,
             size_buffer_mem,
             0,
             "Interlaced png image has too big raster size. Can't load image.");
  a2 = AF_memm_alloc((int)arg4, 4 * v14, (int)"..\\..\\..\\..\\Common\\Formats\\pngread.c", 3034);
  if ( !a2 )
    return kind_of_print_error(
             (int)"..\\..\\..\\..\\Common\\Formats\\pngread.c",
             3038,
             -1000,
             0,
             4 * v14,
             (int)arg4,
             0);
  v16 = 0;
  v63 = (unsigned __int16)v14;
  if ( (_WORD)v14 )
  {
    do
    {
      v17 = AF_memm_alloc((int)arg4, ___size, (int)"..\\..\\..\\..\\Common\\Formats\\pngread.c", 3044);
      *(_DWORD *)&a2[4 * v16] = v17;
      if ( !v17 )
        return kind_of_print_error(
                 (int)"..\\..\\..\\..\\Common\\Formats\\pngread.c",
                 3048,
                 -1000,
                 0,
                 ___size,
                 (int)arg4,
                 0);
      ++v16;
    }
    while ( v16 < (unsigned __int16)v14 );
    if ( (_WORD)v14 )
    {
      v18 = a2;
      v14 = (unsigned __int16)v14;
      do
      {
        for ( i = 0; i < ___size; *(_BYTE *)(i + *(_DWORD *)v18 - 1) = -1 )
          ++i;
        v18 += 4;
        --v14;
      }
      while ( v14 );
    }
  }
  sub_670C0A50(table_of_size, (__int16 *)a4);
  v20 = &table_of_size->field_A;
  v21 = 1;
  v22 = 0;
  v66 = 1;
  v72 = &table_of_size->field_A;
  v69 = 0;
  do
  {
    if ( *(_WORD *)v20 )
    {
      if ( _size )
      {
        memset(v77, 0, _size);
        v20 = v72;
      }
      v23 = *((__int16 *)v20 - 2);
      v64 = *((__int16 *)v20 - 1);
      v67 = *((unsigned __int8 *)&v84 + v22 + 1);
      v16 = 0;
      v24 = *(unsigned __int16 *)(a4 + 4);
      size = *((__int16 *)v20 - 2);                       [8]
      v74 = 0;
      v76 = *(unsigned __int16 *)(a4 + 4);
      if ( v64 > 0 )
      {
        v60 = v23 - 1;
```

```
          do
          {
            sub_670C0090((int)a1, (size_t)&v81, Src, v23);
            v25 = v60 / *(__int16 *)v72;
            if ( !v25 )
              v25 = 1;
            LOBYTE(a6a) = v25;
            v59 = v25;
            sub_670C1030((int)Src, (int)v77, buffer_mem, 0, size, a6a, a5, a8, *(_DWORD *)(a3 + 16));         [4]
                                [...]
          }
          while ( v64 > 0 );
          v20 = v72;
        }
        v21 = v66;
        v22 = v69;
      }
      ++v21;
      ++v22;
      v20 = (int *)((char *)v20 + 6);
      v66 = v21;
      v69 = v22;
      v72 = v20;
    }
    while ( (__int16)v21 <= 7 );
                [...]
  return 0;
}
```

In [4] we can identify our function call with our parameters named here `buffer_mem` and `size` respectively. The `buffer_mem` is allocated in [5] and the size for his allocation is computed in [6] through a call to the function `lead_to_compute_size_based_width_bits` returning an unsigned value as we can see in the following pseudo code where the indirect call lands to the function `compute_size`

```
unsigned int __thiscall compute_size(IGDIBOject *this)
{
  return ((this->width * this->colorspace_related * this->depth + 31) >> 3) & 0xFFFFFFFC;      [7]
}
```

We can see the final size for the `buffer_mem` is computed from a field directly taken to from file, like `width` and other valued derived from `bits` and `colorspace` computation in [7].
Now if take a look back to the `size` parameter we can observe it’s computed differently, getting its value from `v20` at [8]. This is a pointer to signed integers, where `size` is 32-bits unsigned integer.

When looking further into how this table of integer is filled, we land to function `compute_raster_size`, which is computing a size using `bits` and `width` through a test case of `PNG_COLOR_SPACE_TYPE_color_type`.

```
unsigned int __cdecl compute_raster_size(int a1)
{
  unsigned int v1; // eax
  unsigned int raster_size; // eax

  v1 = 0;
  switch ( PNG_COLOR_SPACE_TYPE_color_type )
  {
    case 0u:
    case 3u:
      raster_size = ((bits * width) + 7) >> 3) + 1;
      break;
    case 2u:
      raster_size = ((3 * bits * width) >> 3) + 1;
      break;
    case 4u:
      raster_size = (2 * ((bits * width) >> 3)) + 1;
      break;
    case 6u:
      v1 = (4 * bits * width) >> 3;
      goto LABEL_6;
    default:
LABEL_6:
      raster_size = v1 + 1;
      break;
  }
  return raster_size;
}
```

The cast conversion to `int16` at [8], of the value computed from `compute_raster_size`, causes a sign extension when transforming the value into a larger data type (from int16 to size_t) at [8]. This in turn increases the loop count via the `size` variable, allowing an attacker to cause an out-of-bounds write leading to memory corruption, which could result in remote code execution.

Crash Information

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


KEY_VALUES_STRING: 1

        Key  : AV.Fault
        Value: Write

        Key  : Analysis.CPU.Sec
        Value: 0

        Key  : Analysis.DebugAnalysisProvider.CPP
        Value: Create: 8007007e on DESKTOP-PJK7PVH

        Key  : Analysis.DebugData
        Value: CreateObject

        Key  : Analysis.DebugModel
        Value: CreateObject

        Key  : Analysis.Elapsed.Sec
        Value: 4

        Key  : Analysis.Memory.CommitPeak.Mb
        Value: 78

        Key  : Analysis.System
        Value: CreateObject

        Key  : Timeline.OS.Boot.DeltaSec
        Value: 91061

        Key  : Timeline.Process.Start.DeltaSec
        Value: 8


ADDITIONAL_XML: 1

APPLICATION_VERIFIER_LOADED: 1

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 670c0cb2 (igCore19d!IG_mpi_page_set+0x000e5922)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 00000001
   Parameter[1]: 0f317000
Attempt to write to address 0f317000

FAULTING_THREAD:  000010cc

PROCESS_NAME:  simple.exe_141.exe

WRITE_ADDRESS:  0f317000

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:  0f317000

STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
009dd530 670c1114 0f2fe5c0 0f30e5f8 ffff8a07 igCore19d!IG_mpi_page_set+0xe5922
009dd550 670bf981 0f2fe5c0 0ee305c0 0f30e5f8 igCore19d!IG_mpi_page_set+0xe5d84
009df1dc 670c0c74 009df72c 1000001b 0e64afe8 igCore19d!IG_mpi_page_set+0xe45f1
009df210 670be32c 009df72c 1000001b 0e64afe8 igCore19d!IG_mpi_page_set+0xe58e4
009df6a4 66fb07c9 009df72c 0e64afe8 00000001 igCore19d!IG_mpi_page_set+0xe2f9c
009df6dc 66fefb97 00000000 0e64afe8 009df72c igCore19d!IG_image_savelist_get+0xb29
009df958 66fef4f9 00000000 09e01fa8 00000001 igCore19d!IG_mpi_page_set+0x14807
009df978 66f86007 00000000 09e01fa8 00000001 igCore19d!IG_mpi_page_set+0x14169
009df998 006059ac 09e01fa8 009dfa84 009dfaa8 igCore19d!IG_load_file+0x47
009dfa98 006061a7 09e01fa8 009dfbcc 00000021 simple_exe_141+0x159ac
009dfc64 00606cbe 00000005 09daef50 09c93f40 simple_exe_141+0x161a7
009dfc78 00606b27 f7329ef4 006015e1 006015e1 simple_exe_141+0x16cbe
009dfcd4 006069bd 009dfce4 00606d38 009dfcf4 simple_exe_141+0x16b27
009dfcdc 00606d38 009dfcf4 75286359 00bc4000 simple_exe_141+0x169bd
009dfce4 75286359 00bc4000 75286340 009dfd50 simple_exe_141+0x16d38
009dfcf4 779c7b74 00bc4000 469eabd0 00000000 KERNEL32!BaseThreadInitThunk+0x19
009dfd50 779c7b44 ffffffff 779e8f0f 00000000 ntdll!__RtlUserThreadStart+0x2f
009dfd60 00000000 006015e1 00bc4000 00000000 ntdll!_RtlUserThreadStart+0x1b


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  igCore19d!IG_mpi_page_set+e5922

MODULE_NAME: igCore19d

IMAGE_NAME:  igCore19d.dll

FAILURE_BUCKET_ID:  INVALID_POINTER_WRITE_AVRF_c0000005_igCore19d.dll!IG_mpi_page_set

OS_VERSION:  10.0.18362.239

BUILDLAB_STR:  19h1_release_svc_prod1

OSPLATFORM_TYPE:  x86

OSNAME:  Windows 10

FAILURE_ID_HASH:  {39ff52ad-9054-81fd-3e4d-ef5d82e4b2c1}

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

Timeline

2020-01-30 - Vendor Disclosure
2020-04-30 - Vendor Patched

2020-05-05 - Public Release

CREDIT

Discovered by Emmanuel Tacheau of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT
TALOS-2020-0988

NEXT REPORT
TALOS-2020-0999