Talos Vulnerability Report

TALOS-2020-0986

Accusoft ImageGear PCX uncompress_scan_line buffer size computation code execution vulnerability

FEBRUARY 10, 2020

CVE NUMBER

CVE-2020-6063

Summary

An exploitable out-of-bounds write vulnerability exists in the `uncompress_scan_line` function of the igcore19d.dll library of Accusoft ImageGear, version 19.5.0. A specially crafted PCX 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-131: Incorrect Calculation of Buffer Size

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 `uncompress_scan_line` function. A specially crafted PCX file can lead to an out-of-bounds write which can result in remote code execution.

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

```
eax=00000104 ebx=0e6f8df8 ecx=0000000f edx=005c7000 esi=0e9e8e00 edi=00000200
eip=5de9a98c esp=006ff230 ebp=006ff2a8 iopl=0         ov up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010a06
igCore19d!IG_mpi_page_set+0xdf5fc:
5de9a98c 880c43          mov     byte ptr [ebx+eax*2],cl    ds:002b:0e6f9000=??
0:000> kb
 # ChildEBP RetAddr  Args to Child              
WARNING: Stack unwind information not available. Following frames may be wrong.
00 006ff2a8 5de9b446 006ff828 1000001b 006ff308 igCore19d!IG_mpi_page_set+0xdf5fc
01 006ff2d0 5de9a26a 006ff828 1000001b 0e18eff8 igCore19d!IG_mpi_page_set+0xe00b6
02 006ff7a0 5dd907c9 006ff828 0e18eff8 00000001 igCore19d!IG_mpi_page_set+0xdeeda
03 006ff7d8 5ddcfb97 00000000 0e18eff8 006ff828 igCore19d!IG_image_savelist_get+0xb29
04 006ffa54 5ddcf4f9 00000000 0977dfa8 00000001 igCore19d!IG_mpi_page_set+0x14807
05 006ffa74 5dd66007 00000000 0977dfa8 00000001 igCore19d!IG_mpi_page_set+0x14169
06 006ffa94 00d859ac 0977dfa8 006ffb80 006ffba4 igCore19d!IG_load_file+0x47
07 006ffb94 00d861a7 0977dfa8 006ffcc8 00000021 simple_exe_141+0x159ac
08 006ffd60 00d86cbe 00000005 0972af50 09614f40 simple_exe_141+0x161a7
09 006ffd74 00d86b27 44fe6f44 00d815e1 00d815e1 simple_exe_141+0x16cbe
0a 006ffdd0 00d869bd 006ffde0 00d86d38 006ffdf0 simple_exe_141+0x16b27
0b 006ffdd8 00d86d38 006ffdf0 76cd6359 005c4000 simple_exe_141+0x169bd
0c 006ffde0 76cd6359 005c4000 76cd6340 006ffe4c simple_exe_141+0x16d38
0d 006ffdf0 77577b74 005c4000 04605d1e 00000000 KERNEL32!BaseThreadInitThunk+0x19
0e 006ffe4c 77577b44 ffffffff 77598f06 00000000 ntdll!__RtlUserThreadStart+0x2f
0f 006ffe5c 00000000 00d815e1 005c4000 00000000 ntdll!_RtlUserThreadStart+0x1b
```

As we can see, an out-of-bounds operation occurred.
In order to reach the path to this function some conditions are required:

- `bits_per_pixel` must be set to 2
- `nplanes` must be 2

The pseudo-code of this vulnerable function looks like this:

```
LINE1   int __stdcall uncompress_scan_line(table_function *ptr_function, int a1, pcx_object *p_pcx_data, int a4, IGDIBObject *IGDIBObject)
LINE2   {
LINE3     unsigned int _buffer_size_complete_scan_line; // edi
LINE4     byte *complete_scan_line_buffer; // esi
LINE5     bool v7; // zf
LINE6     byte *buff_mem_overwritten; // ebx
LINE7     byte *v9; // eax
LINE8     int v10; // eax
LINE9     int BytesPerLine; // eax
LINE10    int v12; // edi
LINE11    byte *v13; // ecx
LINE12    byte *v14; // ecx
LINE13    char v15; // bl
LINE14    byte *v16; // ecx
LINE15    byte *v17; // esi
LINE16    byte *v18; // eax
LINE17    char v19; // bl
LINE18    int v20; // esi
LINE19    unsigned int v21; // esi
LINE20    bool v22; // cf
LINE21    void *v23; // ecx
LINE22    unsigned int i; // eax
LINE23    int v26[13]; // [esp+Ch] [ebp-6Ch]
LINE24    int v27; // [esp+40h] [ebp-38h]
LINE25    byte *v28; // [esp+48h] [ebp-30h]
LINE26    int sizeX; // [esp+4Ch] [ebp-2Ch]
LINE27    int v30; // [esp+50h] [ebp-28h]
LINE28    unsigned int buffer_size_complete_scan_line; // [esp+54h] [ebp-24h]
LINE29    unsigned int v32; // [esp+58h] [ebp-20h]
LINE30    size_t invalid_size; // [esp+5Ch] [ebp-1Ch]
LINE31    byte *v34; // [esp+60h] [ebp-18h]
LINE32    byte *_complete_scan_line_buffer; // [esp+64h] [ebp-14h]
LINE33    byte *v36; // [esp+68h] [ebp-10h]
LINE34    byte *v37; // [esp+6Ch] [ebp-Ch]
LINE35    byte *v38; // [esp+70h] [ebp-8h]
LINE36    byte *v39; // [esp+74h] [ebp-4h]
LINE37
LINE38    v34 = 0;
LINE39    sizeX = getSizeX_0(IGDIBObject);
LINE40    buffer_size_complete_scan_line = (unsigned __int16)p_pcx_data->BytesPerLine
LINE41                                   * (unsigned __int8)p_pcx_data->color_planes;// equal 512
LINE42    _buffer_size_complete_scan_line = buffer_size_complete_scan_line;
LINE43    invalid_size = compute_size_based_imagewidth_bitspersample(IGDIBObject);             [4]
LINE44    sub_77C4AF60((int)ptr_function, a1, (int)v26, 5 * buffer_size_complete_scan_line, 1);
LINE45    complete_scan_line_buffer = 0;
LINE46    v7 = p_pcx_data->encoding == 0;
LINE47    _complete_scan_line_buffer = 0;
LINE48    if ( !v7 )
LINE49    {
LINE50      complete_scan_line_buffer = AF_memm_alloc(
LINE51                                    a1,
LINE52                                    _buffer_size_complete_scan_line,
LINE53                                    (int)"..\\..\\..\\..\\Common\\Formats\\pcxread.c",
LINE54                                    835);
LINE55      _complete_scan_line_buffer = complete_scan_line_buffer;
LINE56      if ( !complete_scan_line_buffer )
LINE57        v34 = (byte *)kind_of_print_error(
LINE58                        (int)"..\\..\\..\\..\\Common\\Formats\\pcxread.c",
LINE59                        837,
LINE60                        -1000,
LINE61                        0,
LINE62                        _buffer_size_complete_scan_line,
LINE63                        a1,
LINE64                        0);
LINE65    }
LINE66    buff_mem_overwritten = AF_memm_alloc(a1, invalid_size, (int)"..\\..\\..\\..\\Common\\Formats\\pcxread.c", 839);
[3]
LINE67    v28 = buff_mem_overwritten;
LINE68    if ( buff_mem_overwritten )
LINE69      v9 = v34;
LINE70    else
LINE71      v9 = (byte *)kind_of_print_error(
LINE72                     (int)"..\\..\\..\\..\\Common\\Formats\\pcxread.c",
LINE73                     842,
LINE74                     -1000,
LINE75                     0,
LINE76                     invalid_size,
LINE77                     a1,
LINE78                     0);
LINE79    if ( !v9 )
LINE80    {
LINE81      v27 = 0;
LINE82      v30 = 0;
LINE83      if ( getSizeY_0(IGDIBObject) )
LINE84      {
LINE85        while ( !p_pcx_data->encoding )
LINE86        {
LINE87          complete_scan_line_buffer = (byte *)sub_77C4B280(v26, _buffer_size_complete_scan_line);
LINE88          _complete_scan_line_buffer = complete_scan_line_buffer;
LINE89          if ( !complete_scan_line_buffer )
LINE90          {
LINE91            v10 = kind_of_print_error(
LINE92                    (int)"..\\..\\..\\..\\Common\\Formats\\pcxread.c",
LINE93                    856,
LINE94                    -2051,
LINE95                    0,
LINE96                    _buffer_size_complete_scan_line,
LINE97                    a1,
LINE98                    0);
LINE99  LABEL_13:
LINE100           if ( v10 )
LINE101             goto LABEL_25;
LINE102         }
LINE103         if ( p_pcx_data->bits_per_pixel == 1 )     [2]
LINE104         {
LINE105           BytesPerLine = (unsigned __int16)p_pcx_data->BytesPerLine;
LINE106           v12 = 0;
LINE107           v34 = &complete_scan_line_buffer[BytesPerLine];
LINE108           v13 = &complete_scan_line_buffer[BytesPerLine + BytesPerLine];
LINE109           v38 = v13;
LINE110           v14 = &v13[BytesPerLine];
LINE111           v39 = complete_scan_line_buffer;
LINE112           v37 = v14;
LINE113           v36 = buff_mem_overwritten;
LINE114           if ( sizeX )
LINE115           {
LINE116             v32 = ((unsigned int)(sizeX - 1) >> 1) + 1;
```

```
            do
            {
              v15 = 2
                  * (((unsigned __int8)(*v34 & byte_77E9E184[v12]) >> (7 - v12)) | (2
                                                                                  * (((unsigned __int8)(*v38 & byte_77E9E184[v12])
>> (7 - v12)) | (2 * ((unsigned __int8)(*v14 & byte_77E9E184[v12]) >> (7 - v12))))));
              v16 = v37;
              v17 = v34;
              v18 = v36 + 1;
              *v36 = ((unsigned __int8)(*v39 & byte_77E9E184[v12]) >> (7 - v12)) | v15;
              v36 = v18;
              v19 = (unsigned __int8)(*v16 & byte_77E9E185[v12]) >> (6 - v12);
              v14 = v37;
              *v18 = ((unsigned __int8)(*v39 & byte_77E9E185[v12]) >> (6 - v12)) | (2
                                                                                  * (((unsigned __int8)(*v17 & byte_77E9E185[v12])
>> (6 - v12)) | (2 * (((unsigned __int8)(*v38 & byte_77E9E185[v12]) >> (6 - v12)) | (2 * v19)))));
              if ( v12 == 6 )
              {
                ++v39;
                ++v38;
                v12 = 0;
                ++v14;
                v34 = v17 + 1;
                v37 = v14;
              }
              else
              {
                v12 += 2;
              }
              v7 = v32-- == 1;
              v36 = v18 + 1;
            }
            while ( !v7 );
            buff_mem_overwritten = v28;
          }
          _buffer_size_complete_scan_line = buffer_size_complete_scan_line;
        }
        else
        {
          for ( i = 0; i < _buffer_size_complete_scan_line; ++i )
          {
            buff_mem_overwritten[2 * i] = complete_scan_line_buffer[i] >> 4;       [1]
            buff_mem_overwritten[2 * i + 1] = complete_scan_line_buffer[i] & 0xF;
          }
        }
        v20 = v30;
        if ( !sub_77C494C0((int)ptr_function, (int)buff_mem_overwritten, v30, invalid_size) )
        {
          v21 = v20 + 1;
          v30 = v21;
          v22 = v21 < getSizeY_0(IGDIBObject);
          complete_scan_line_buffer = _complete_scan_line_buffer;
          if ( v22 )
            continue;
        }
        goto LABEL_25;
      }
      v10 = sub_77D3A2C0((int)v26, complete_scan_line_buffer, _buffer_size_complete_scan_line, (int)&v27);
      goto LABEL_13;
    }
  }
LABEL_25:
  if ( _complete_scan_line_buffer && p_pcx_data->encoding )
    sub_77C55F40((void *)a1, _complete_scan_line_buffer, (int)"..\\..\\..\\..\\Common\\Formats\\pcxread.c", 910);
  if ( buff_mem_overwritten )
    sub_77C55F40((void *)a1, buff_mem_overwritten, (int)"..\\..\\..\\..\\Common\\Formats\\pcxread.c", 912);
  sub_77C4AC30(v26);
  return sub_77C2AA00(v23);
}
```

In this algorithm we can observe a function `uncompress_scan_line`, whose objective is to decompress the pcx data, is crashing while filling the buffer `buff_mem_overwritten` in [1]. The path taken depends of the value from the pcx header `bits_per_pixel` not equal to `1` as we can see in [2]. The out-of-bounds occurs because the computed size of the target memory `buff_mem_overwritten` allocated in [3] is computed from an indirect function call through the function `compute_size_based_imagewidth_bitspersample` at [4]. The size is computed in the final destination function named `compute_size_for_pcx`.

The pseudo code for the function `compute_size_for_pcx` is the following:

```
unsigned int __thiscall compute_size_for_pcx(IGDIBOject *this)
{
  return ((this->sizeX * this->size_of_table_round * this->depth + 31) >> 3) & 0xFFFFFFFC;   [5]
  //
  // round_valued_bitperpixel_plane = round_max_bits_per_sample(bitspersample);
  //
  // sizeX = pcx_object->Xmax - pcx_object->Xmin + 1;
  // size_table_round = 1 is for up to 256 colors
  // return_1_if_for_colors_less_than_256
  // product_bit_per_pixel_nplanes = 1 or product_bit_per_pixel_nplanes = 4 or product_bit_per_pixel_nplanes = 8
  //
  //
  // depth is calculated from int __cdecl round_depth(int depth)
}
```

We can see the returned size is multiplication of three different values where one of them, `sizeX` at [5], is calculated with the formula `sizeX = pcx_object->Xmax - pcx_object->Xmin + 1` [6], in a function named `build_IGDIBObject_pcxrelatedobject`:

```
int __stdcall build_IGDIBObject_pcxrelatedobject(int a1, pcx_object *pcx_object, ColorMapTable *pColorMapTable, IGDIBOject *IGDIBOject)
{
  pcx_object *_pcx_object; // edx
  int total_bits_per_pixel; // edi
  __int16 sizeX; // bx
  __int16 v7; // ax
  int v8; // eax
  int v9; // eax
  void *v10; // ecx
  void *dest_buffer; // eax
  int _size; // [esp-4h] [ebp-10h]
  __int16 sizeY; // [esp+18h] [ebp+Ch]

  _pcx_object = pcx_object;
  total_bits_per_pixel = (unsigned __int8)pcx_object->bits_per_pixel * (unsigned __int8)pcx_object->color_planes;
  sizeX = pcx_object->Xmax - pcx_object->Xmin + 1;    [6]
  sizeY = pcx_object->Ymax - pcx_object->Ymin + 1;
  [...]
}
```

This formula using signed integer may cause the issue if the value for `Xmax` is less than the value of `Xmin`.

If we take a look at the origin of `Xmax` and `Xmin` we'll need to look at the `PCX_parse_header` responsible for filling directly from the file the values of the pcx header.

The pseudo code of the function named `PCX_parse_header`:

```
int __stdcall PCX_parse_header(void *this, int a1, pcx_object *pcx_object, DWORD *pcolorMapTable, int encoding_related, int
current_offset, int a6)
{
  size_t v7; // ecx
  table_function *v8; // esi
  int offset_in_file; // ecx
  __int16 XMin; // ax           [8]
  __int16 YMin; // ax
  ColorMapTable *colorMapTable; // ebx
  size_t TotalBits; // ecx
  int v14; // eax
  pcx_object *_pcx_object_2; // edx
  bool v16; // zf
  char v17; // al
  int v18; // edi
  char *v19; // edx
  char *v20; // edx
  int v21; // edi
  byte *v22; // ebx
  int v23; // edi
  int v24; // edi
  unsigned int v25; // eax
  _BYTE *v26; // ecx
  void *v27; // ecx
  char *bits_per_pixel; // [esp+30h] [ebp-60h]
  char *palette; // [esp+34h] [ebp-5Ch]
  char *color_planes; // [esp+38h] [ebp-58h]
  __int16 *H_res; // [esp+3Ch] [ebp-54h]
  __int16 *V_res; // [esp+40h] [ebp-50h]
  byte buffer; // [esp+47h] [ebp-49h]
  int pcx_type; // [esp+48h] [ebp-48h]
  int Filler; // [esp+4Ch] [ebp-44h]
  DWORD *_pcolorMapTable; // [esp+50h] [ebp-40h]
  pcx_object *_pcx_object; // [esp+54h] [ebp-3Ch]
  int _current_offset; // [esp+58h] [ebp-38h]
  int v40; // [esp+5Ch] [ebp-34h]
  int v41; // [esp+60h] [ebp-30h]
  int v42; // [esp+64h] [ebp-2Ch]
  int v43; // [esp+68h] [ebp-28h]
  int v44; // [esp+6Ch] [ebp-24h]
  int v45; // [esp+70h] [ebp-20h]
  int v46; // [esp+74h] [ebp-1Ch]
  int v47; // [esp+78h] [ebp-18h]
  int v48; // [esp+7Ch] [ebp-14h]
  int v49; // [esp+80h] [ebp-10h]
  int v50; // [esp+84h] [ebp-Ch]
  int v51; // [esp+88h] [ebp-8h]

  v8 = (table_function *)this;
  _pcx_object = pcx_object;
  _pcolorMapTable = pcolorMapTable;
  v40 = 0;
  v41 = 0x80008000;
  v42 = 0x80800000;
  v43 = 0x80000080;
  v44 = 0x80808000;
  v45 = 0x80808000;
  v46 = 0xC0C0C0;
  v47 = 0xFF00FF00;
  v48 = 0xFFFF0000;
  v49 = 0xFF0000FF;
  v50 = 0xFFFFFF00;
  v51 = 0xFFFFFF00;
  sub_77C491E0(v7, (table_function *)this, 31, (int)"PCX", 31, 1, 0, 0, 0, 1);
  set_endian_type((table_function *)this, 0);
  _current_offset = get_one_byte((table_function *)this, (byte *)pcx_object);
  pcx_type = (int)&pcx_object->version;
  _current_offset += get_one_byte((table_function *)this, (byte *)&pcx_object->version);
  _current_offset += get_one_byte((table_function *)this, (byte *)&pcx_object->encoding);
  bits_per_pixel = &pcx_object->bits_per_pixel;
  _current_offset += get_one_byte((table_function *)this, (byte *)&pcx_object->bits_per_pixel);
  _current_offset += read_short((table_function *)this, (byte *)&pcx_object->Xmin);
  _current_offset += read_short((table_function *)this, (byte *)&pcx_object->Ymin);
  _current_offset += read_short((table_function *)this, (byte *)&pcx_object->Xmax);
  _current_offset += read_short((table_function *)this, (byte *)&pcx_object->Ymax);
  H_res = &pcx_object->hres;
  _current_offset += read_short((table_function *)this, (byte *)&pcx_object->hres);
  V_res = &pcx_object->vres;
  _current_offset += read_short((table_function *)this, (byte *)&pcx_object->vres);
  palette = pcx_object->palette;
  _current_offset += get_bytes_into_buffer((table_function *)this, (byte *)pcx_object->palette, 48u);
  _current_offset += get_one_byte((table_function *)this, (byte *)&pcx_object->Reserved);
  color_planes = &pcx_object->color_planes;
  _current_offset += get_one_byte((table_function *)this, (byte *)&pcx_object->color_planes);
  _current_offset += read_short((table_function *)this, (byte *)&pcx_object->BytesPerLine);
  _current_offset += read_short((table_function *)this, (byte *)&pcx_object->PaletteInfo);
  _current_offset += read_short((table_function *)this, (byte *)&pcx_object->HScreenSize);
  _current_offset += read_short((table_function *)this, (byte *)&pcx_object->VScreenSize);
  Filler = (int)pcx_object->Filler;
  offset_in_file = get_bytes_into_buffer((table_function *)this, (byte *)pcx_object->Filler, 54u) + _current_offset;
  XMin = pcx_object->Xmin;
  if ( XMin > pcx_object->Xmax )         [7]
  {
    pcx_object->Xmin = pcx_object->Xmax;
    v8 = (table_function *)this;
    pcx_object->Xmax = XMin;
  }
  YMin = pcx_object->Ymin;
  colorMapTable = (ColorMapTable *)_pcolorMapTable;
  if ( YMin > pcx_object->Ymax )
  {
    pcx_object->Ymin = pcx_object->Ymax;
    v8 = (table_function *)this;
    pcx_object->Ymax = YMin;
  }
  [...]
}
```

In [7] we can see a signed comparison causing the issue since both `Xmin` [8] and `pcx_object->Xmax` are signed, preventing the exchange of the two variables, which was supposed to prevent the error of the subtraction operation performed in [6]. This leads to some smaller value than planned, causing the invalid size computation of the memory allocation and thus leading to the out of band write.

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: 8

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

    Key  : Analysis.System
    Value: CreateObject

    Key  : Timeline.OS.Boot.DeltaSec
    Value: 297991

    Key  : Timeline.Process.Start.DeltaSec
    Value: 80


ADDITIONAL_XML: 1

APPLICATION_VERIFIER_LOADED: 1

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 5de9a98c (igCore19d!IG_mpi_page_set+0x000df5fc)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 00000001
   Parameter[1]: 0e6f9000
Attempt to write to address 0e6f9000

FAULTING_THREAD:  00005a54

PROCESS_NAME:  simple.exe_141.exe

WRITE_ADDRESS:  0e6f9000

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:  0e6f9000

STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
006ff2a8 5de9b446 006ff828 1000001b 006ff308 igCore19d!IG_mpi_page_set+0xdf5fc
006ff2d0 5de9a26a 006ff828 1000001b 0e18eff8 igCore19d!IG_mpi_page_set+0xe00b6
006ff7a0 5dd907c9 006ff828 0e18eff8 00000001 igCore19d!IG_mpi_page_set+0xdeeda
006ff7d8 5ddcfb97 00000000 0e18eff8 006ff828 igCore19d!IG_image_savelist_get+0xb29
006ffa54 5ddcf4f9 00000000 0977dfa8 00000001 igCore19d!IG_mpi_page_set+0x14807
006ffa74 5dd66007 00000000 0977dfa8 00000001 igCore19d!IG_mpi_page_set+0x14169
006ffa94 00d859ac 0977dfa8 006ffb80 006ffba4 igCore19d!IG_load_file+0x47
006ffb94 00d861a7 0977dfa8 006ffcc8 00000021 simple_exe_141+0x159ac
006ffd60 00d86cbe 00000005 0972af50 09614f40 simple_exe_141+0x161a7
006ffd74 00d86b27 44fe6f44 00d815e1 00d815e1 simple_exe_141+0x16cbe
006ffdd0 00d869bd 006ffde0 00d86d38 006ffdf0 simple_exe_141+0x16b27
006ffdd8 00d86d38 006ffdf0 76cd6359 005c4000 simple_exe_141+0x169bd
006ffde0 76cd6359 005c4000 76cd6340 006ffe4c simple_exe_141+0x16d38
006ffdf0 77577b74 005c4000 04605d1e 00000000 KERNEL32!BaseThreadInitThunk+0x19
006ffe4c 77577b44 ffffffff 77598f06 00000000 ntdll!__RtlUserThreadStart+0x2f
006ffe5c 00000000 00d815e1 005c4000 00000000 ntdll!_RtlUserThreadStart+0x1b


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  igCore19d!IG_mpi_page_set+df5fc

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

2020-01-27 - Vendor Disclosure

2020-02-10 - Public Release

CREDIT

Discovered by Emmanuel Tacheau of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT

TALOS-2019-0972

NEXT REPORT

TALOS-2020-0987