Talos Vulnerability Report

TALOS-2021-1296

Accusoft ImageGear TIF IP_planar_raster_unpack improper array index validation vulnerability

JUNE 1, 2021

CVE NUMBER

CVE-2021-21833

Summary

An improper array index validation vulnerability exists in the TIF IP_planar_raster_unpack functionality of Accusoft ImageGear 19.9. A specially crafted malformed file can lead to an out-of-bounds write. An attacker can provide a malicious file to trigger this vulnerability.

Tested Versions

Accusoft ImageGear 19.9

Product URLs

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-119 - Improper Restriction of Operations within the Bounds of a Memory Buffer

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.

There is a vulnerability in the `IP_planar_raster_unpack` function which occurs with a specially crafted TIF file, leading to an out-of-bounds write which can result in code execution. Trying to load a malformed TIF file, we end up in the following situation:

```
This exception may be expected and handled.
eax=00000000 ebx=00000002 ecx=00000005 edx=00000004 esi=0b98d000 edi=00000004
eip=5c194b4c esp=0019f484 ebp=0019f4a0 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010246
igCore19d!IG_mpi_page_set+0x8dfc:
5c194b4c 8806            mov     byte ptr [esi],al          ds:002b:0b98d000=??
```

This is corresponding to the following pseudo code :

```
LINE1    undefined * __cdecl
LINE2    IP_planar_raster_unpack
LINE3              (int SamplesPerPixel,int index_up_to_sampleperpixel,dword BitsPerSample,
LINE4              int sample_per_pixel_multiply_width,void *src_buffer,void *dest_buffer,
LINE5              size_t size_memory_buffer)
LINE6
LINE7    {
           [...]
LINE31     switch(BitsPerSample) {
LINE32     case 1:
LINE33     case 2:
LINE34     case 4:
LINE35       nb_bits_BitsPerSample = (int)(8 / (longlong)(int)BitsPerSample);
LINE36       index_larger_loop = 0;
LINE37       _index_larger_loop = 0;
LINE38       _mask_bitpersample = (char)(1 << ((byte)BitsPerSample & 0x1f)) + -1;
LINE39       if (0 < sample_per_pixel_multiply_width / nb_bits_BitsPerSample) {
LINE40         do {
LINE41           bVar2 = *(byte *)(_index_larger_loop + (int)src_buffer);
LINE42           if (0 < nb_bits_BitsPerSample) {
LINE43             iVar9 = (nb_bits_BitsPerSample + -1) * BitsPerSample;
LINE44             _bit_to_process = nb_bits_BitsPerSample;
LINE45             do {
LINE46               bVar4 = (byte)iVar9;
LINE47               iVar9 = iVar9 - BitsPerSample;
LINE48               *(byte *)(index_up_to_sampleperpixel + (int)dest_buffer) =
LINE49                    bVar2 >> (bVar4 & 0x1f) & _mask_bitpersample;
LINE50               index_up_to_sampleperpixel = index_up_to_sampleperpixel + SamplesPerPixel;
LINE51               _bit_to_process = _bit_to_process + -1;
LINE52             } while (_bit_to_process != 0);
LINE53           }
LINE54           index_larger_loop = _index_larger_loop + 1;
LINE55           _index_larger_loop = index_larger_loop;
LINE56         } while (index_larger_loop < sample_per_pixel_multiply_width / nb_bits_BitsPerSample);
LINE57       }
LINE58       nb_bytes = (SamplesPerPixel * sample_per_pixel_multiply_width) / nb_bits_BitsPerSample;
LINE59       module_bytes = (SamplesPerPixel * sample_per_pixel_multiply_width) % nb_bits_BitsPerSample;
LINE60       if (0 < module_bytes) {
LINE61         nb_bytes = (uint)src_buffer & 0xffffff00 |
LINE62                    (uint)*(byte *)(index_larger_loop + (int)src_buffer);
LINE63         if (0 < module_bytes) {
LINE64           _oobw = (byte *)(index_up_to_sampleperpixel + (int)dest_buffer);
LINE65           puVar6 = (undefined *)
LINE66                    ((uint)CONCAT21((short)((ulonglong)(8 / (longlong)(int)BitsPerSample) >> 0x10),
LINE67                                    *(byte *)(index_larger_loop + (int)src_buffer)) << 8);
LINE68           nb_bits_BitsPerSample = (nb_bits_BitsPerSample + -1) * BitsPerSample;
LINE69           do {
LINE70             bVar2 = (byte)nb_bits_BitsPerSample;
LINE71             nb_bits_BitsPerSample = nb_bits_BitsPerSample - BitsPerSample;
LINE72             result = (byte)((uint)puVar6 >> 8) >> (bVar2 & 0x1f) & _mask_bitpersample;
LINE73             puVar6 = (undefined *)((uint)puVar6 & 0xffffff00 | (uint)result);
LINE74             *_oobw = result;
LINE75             _oobw = _oobw + SamplesPerPixel;
LINE76             module_bytes = module_bytes + -1;
LINE77           } while (module_bytes != 0);
LINE78           return puVar6;
LINE79         }
LINE80       }
LINE81       [...]
LINE174  }
```

and the crash is happening at `LINE74`. The `oobw` variable is derived from the `dest_buffer` passed as an argument as you can see at `LINE64`

The out-of-bounds write is happening throught the loop performed between `LINE69` and `LINE77` controlled by the `module_bytes` which in our case is 4 as this is the rest of the division in `LINE59`. The issue is there is no control against the size of `_oobw` computed in `LINE75` while adding the value of `SamplesPerPixel`.

Now let's see how the buffer size is computed.

When looking at the memory buffer allocated corresponding to `oobw` we can see the size of '4' bytes in our cases:

```
0:000> !heap -p -a esi
    address 0b98d000 found in
    _DPH_HEAP_ROOT @ 3f11000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                                 9fe2104:          b98cff8                4 -          b98c000             2000
    5ce2a8b0 verifier!AVrfDebugPageHeapAllocate+0x00000240
    77dcef3e ntdll!RtlDebugAllocateHeap+0x00000039
    77d37080 ntdll!RtlpAllocateHeap+0x000000f0
    77d36ddc ntdll!RtlpAllocateHeapInternal+0x0000104c
    77d35d7e ntdll!RtlAllocateHeap+0x0000003e
    5cccdcff MSVCR110!malloc+0x00000049
    5c1861de igCore19d!AF_memm_alloc+0x0000001e
    5c297518 igCore19d!IG_mpi_page_set+0x0010b7c8
    5c29721a igCore19d!IG_mpi_page_set+0x0010b4ca
    5c29c353 igCore19d!IG_mpi_page_set+0x00110603
    5c2962cb igCore19d!IG_mpi_page_set+0x0010a57b
    5c1610d9 igCore19d!IG_image_savelist_get+0x00000b29
    5c1a0557 igCore19d!IG_mpi_page_set+0x00014807
    5c19feb9 igCore19d!IG_mpi_page_set+0x00014169
    5c135777 igCore19d!IG_load_file+0x00000047
    00498a3a Fuzzme!fuzzme+0x0000004a [C:\Users\etach\Documents\work\gitlab\ImageGear\Fuzzme\Fuzzme.cpp @ 282]
    00498e36 Fuzzme!main+0x00000376 [C:\Users\etach\Documents\work\gitlab\ImageGear\Fuzzme\Fuzzme.cpp @ 477]
    004daa53 Fuzzme!invoke_main+0x00000033 [d:\agent\_work\57\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl @ 78]
    004da8a7 Fuzzme!__scrt_common_main_seh+0x00000157 [d:\agent\_work\57\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl @ 288]
    004da73d Fuzzme!__scrt_common_main+0x0000000d [d:\agent\_work\57\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl @ 331]
    004daad8 Fuzzme!mainCRTStartup+0x00000008 [d:\agent\_work\57\s\src\vctools\crt\vcstartup\src\startup\exe_main.cpp @ 17]
    75bbfa29 KERNEL32!BaseThreadInitThunk+0x00000019
    77d57a4e ntdll!__RtlUserThreadStart+0x0000002f
    77d57a1e ntdll!_RtlUserThreadStart+0x0000001b
```

The buffer corresponding to `dest_buffer` is allocated in a very-large previous function named `allocate_table_buffer_related_sample_per_pixel` with the following pseudo-code in `LINE378`:

```
LINE175  int allocate_table_buffer_related_sample_per_pixel
LINE176                (mys_table_function *mys_table_function,uint kind_of_heap,mys_tags_data *mys_tags_data
LINE177                ,undefined4 some_buffer,HIGDIBINFO higdibinfo,short *ID_TIF_YCBCR_SUBSAMPLING_value)
LINE178
LINE179  {
LINE180    dest_buffer = (byte *)0x0;
LINE181    _raster_size = 0;
LINE182    local_c = 0;
LINE183    local_24 = (byte *)0x0;
LINE184    table_ptr = (int *)0x0;
LINE185    if (*(ushort *)&mys_tags_data->ID_TIF_SAMPLES_PER_PIXEL == 0) {
LINE186      AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\tifread.c",0x178d,-0x80d,0,0,0,(LPCHAR)0x0);
LINE187      loop_index_temp = AF_error_check();
LINE188      return loop_index_temp;
LINE189    }
LINE190    io_buff = (io_buffer *)
LINE191              AF_memm_alloc(kind_of_heap,
LINE192                            (uint)*(ushort *)&mys_tags_data->ID_TIF_SAMPLES_PER_PIXEL * 0x34);
LINE193    if (io_buff == (io_buffer *)0x0) {
LINE194      AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\tifread.c",0x1793,-1000,0,0,0,(LPCHAR)0x0);
LINE195      loop_index_temp = AF_error_check();
LINE196      return loop_index_temp;
LINE197    }
LINE198    table_ptr_byte_from_file =
LINE199         (byte **)AF_memm_alloc(kind_of_heap,
LINE200                                (uint)*(ushort *)&mys_tags_data->ID_TIF_SAMPLES_PER_PIXEL << 2);
LINE201    if (table_ptr_byte_from_file == (byte **)0x0) {
LINE202      _look_index = 0x179b;
LINE203      p_size_memory_buffer_00 = table_ptr_byte_from_file;
LINE204    }
LINE205    else {
LINE206      OS_memset(table_ptr_byte_from_file,0,
LINE207                (uint)*(ushort *)&mys_tags_data->ID_TIF_SAMPLES_PER_PIXEL << 2);
LINE208      p_size_memory_buffer_00 =
LINE209           (byte **)AF_memm_alloc(kind_of_heap,
LINE210                                  (uint)*(ushort *)&mys_tags_data->ID_TIF_SAMPLES_PER_PIXEL << 2);
LINE211      if (p_size_memory_buffer_00 != (byte **)0x0) {
LINE212        if (mys_tags_data->ID_TIF_PHOTO_INTERP == IG_TIF_PHOTO_YCBCR) {
LINE213          _look_index = (int)*ID_TIF_YCBCR_SUBSAMPLING_value;
LINE214          local_24 = (byte *)((int)((mys_tags_data->ID_TIF_IMAGE_WIDTH - 1) + _look_index) /
LINE215                              _look_index);
LINE216          *p_size_memory_buffer_00 =
LINE217               (byte *)((ID_TIF_YCBCR_SUBSAMPLING_value[1] * _look_index + 2) * (int)local_24);
LINE218          _raster_size = IO_raster_size_get(higdibinfo);
LINE219          table_ptr = (int *)AF_memm_alloc(kind_of_heap,(int)ID_TIF_YCBCR_SUBSAMPLING_value[1] << 2);
LINE220          if (table_ptr != (int *)0x0) {
LINE221            num_to_alloc = 0;
LINE222            if (0 < ID_TIF_YCBCR_SUBSAMPLING_value[1]) {
LINE223              size_malloc = _raster_size + 0x80;
LINE224              do {
LINE225                _buff_mem = AF_memm_alloc(kind_of_heap,size_malloc);
LINE226                table_ptr[num_to_alloc] = (int)_buff_mem;
LINE227                if (_buff_mem == (void *)0x0) {
LINE228                  table_ptr = (int *)0x0;
LINE229                  _look_index = 0x17d5;
LINE230                  goto error_allocate_mem;
LINE231                }
LINE232                num_to_alloc = num_to_alloc + 1;
LINE233              } while (num_to_alloc < ID_TIF_YCBCR_SUBSAMPLING_value[1]);
LINE234            }
LINE235            goto continue_process;
LINE236          }
LINE237          size_malloc = (int)ID_TIF_YCBCR_SUBSAMPLING_value[1] << 2;
LINE238          _look_index = 0x17cb;
LINE239  error_allocate_mem:
LINE240          AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\tifread.c",_look_index,-1000,0,
LINE241                            kind_of_heap,size_malloc,(LPCHAR)table_ptr);
LINE242          AF_memm_free(kind_of_heap,io_buff);
LINE243          AF_memm_free(kind_of_heap,table_ptr_byte_from_file);
LINE244        }
LINE245        else {
LINE246  continue_process:
LINE247                      /* Planar Config = 1 */
LINE248          if (mys_tags_data->ID_TIF_PLANAR_CONFIG == 1) {
                [...]
LINE340          }
LINE341          else {
LINE342                      /* Planar Config = 2 */
LINE343            if (mys_tags_data->ID_TIF_PLANAR_CONFIG == 2) {
LINE344              if (mys_tags_data->ID_TIF_PHOTO_INTERP == IG_TIF_PHOTO_YCBCR) {
                     [...]
LINE362              }
LINE363              else {
LINE364                _look_index = 0;
LINE365                if (*(short *)&mys_tags_data->ID_TIF_SAMPLES_PER_PIXEL != 0) {
LINE366                  do {
LINE367                    look_index = _look_index + 1;
LINE368                    value_to_store =
LINE369                         (int)*(short *)((mys_tags_data->ID_TIF_BITS_PER_SAMPLE - 2) + look_index * 2)
LINE370                         * mys_tags_data->ID_TIF_IMAGE_WIDTH + 7;
LINE371                    p_size_memory_buffer_00[_look_index] =
LINE372                         (byte *)((int)(value_to_store + (value_to_store >> 0x1f & 7U)) >> 3);
LINE373                    _look_index = look_index;
LINE374                  } while (look_index < (int)(uint)*(ushort *)&mys_tags_data->ID_TIF_SAMPLES_PER_PIXEL
LINE375                           );
LINE376                }
LINE377                _raster_size = IO_raster_size_get(higdibinfo);
LINE378                dest_buffer = (byte *)AF_memm_alloc(kind_of_heap,_raster_size);
LINE379                if (dest_buffer == (byte *)0x0) {
LINE380                  AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\tifread.c",0x183d,-1000,0,
LINE381                                    kind_of_heap,_raster_size,(LPCHAR)0x0);
LINE382                  goto LAB_10177a96;
LINE383                }
LINE384              }
LINE385              loop_index_temp = 0;
LINE386              _look_index = 0;
LINE387              _io_buff = io_buff;
LINE388              if (*(short *)&mys_tags_data->ID_TIF_SAMPLES_PER_PIXEL != 0) {
LINE389                do {
LINE390                  dVar1 = IOb_init(mys_table_function,kind_of_heap,_io_buff,
LINE391                                   (int)p_size_memory_buffer_00[_look_index] * 5,1);
LINE392                  if (0 < (int)dVar1) {
LINE393                    loop_index_temp = 1;
LINE394                    local_c = 1;
LINE395                    break;
LINE396                  }
LINE397                  _look_index = _look_index + 1;
```

```
LINE398                 _io_buff = (io_buffer *)&_io_buff->size_buffer;
LINE399               } while (_look_index < (int)(uint)*(ushort *)&mys_tags_data->ID_TIF_SAMPLES_PER_PIXEL)
LINE400               ;
LINE401             }
LINE402             pIVar3 = (HIGDIBINFO)0x0;
LINE403             higdibinfo = (HIGDIBINFO)0x0;
LINE404             local_18 = 0;
LINE405             while ((loop_index_temp == 0 &&
LINE406                    (local_18 < (int)mys_tags_data->from_ID_TIF_STRIP_OFFSET_or_ID_TIF_TILE_OFFSETS))
LINE407                   ) {
LINE408               if ((mys_tags_data->ID_TIF_TILE_OFFSETS != 0) &&
LINE409                  (*(short *)&mys_tags_data->ID_TIF_SAMPLES_PER_PIXEL != 0)) {
LINE410                 _look_index = 0;
LINE411                 _io_buff = io_buff;
LINE412                 do {
LINE413                   perform_some_read_or_write_intofile
LINE414                             (_io_buff,*(int *)(mys_tags_data->ID_TIF_TILE_OFFSETS +
LINE415                                                (mys_tags_data->
LINE416                                                 result_strip_tile_offset_divided_sample_per_pixel *
LINE417                                                 _look_index + local_18) * 4) +
LINE418                                         mys_tags_data->IFD_Offset,0,0);
LINE419                   _look_index = _look_index + 1;
LINE420                   _io_buff = (io_buffer *)&_io_buff->size_buffer;
LINE421                   loop_index_temp = local_c;
LINE422                   pIVar3 = higdibinfo;
LINE423                 } while (_look_index <
LINE424                         (int)(uint)*(ushort *)&mys_tags_data->ID_TIF_SAMPLES_PER_PIXEL);
LINE425               }
LINE426               strip = 0;
LINE427               if (0 < (int)mys_tags_data->ID_TIF_ROWS_PER_STRIP) {
LINE428                 do {
LINE429                   local_c = loop_index_temp;
LINE430                   if ((int)mys_tags_data->ID_TIF_IMAGE_HEIGHT <= (int)pIVar3) break;
LINE431                   _look_index = 0;
LINE432                   pIVar3 = higdibinfo;
LINE433                   if (mys_tags_data->ID_TIF_PHOTO_INTERP == IG_TIF_PHOTO_YCBCR) {
                           [...]
LINE473                   }
LINE474                   else {
LINE475                     _io_buff = io_buff;
LINE476                     if (*(short *)&mys_tags_data->ID_TIF_SAMPLES_PER_PIXEL != 0) {
LINE477                       do {
LINE478                         src_buff = (byte *)get_data_from_file(_io_buff,(uint)*
LINE479                                                               p_size_memory_buffer_00);
LINE480                         table_ptr_byte_from_file[_look_index] = src_buff;
LINE481                         if (src_buff == (byte *)0x0) {
LINE482                           AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\tifread.c",0x1881,
LINE483                                             -0x803,0,(AT_INT)*p_size_memory_buffer_00,0,(LPCHAR)0x0)
LINE484                           ;
LINE485                           loop_index_temp = loop_index_temp + 1;
LINE486                           break;
LINE487                         }
LINE488                         _look_index = _look_index + 1;
LINE489                         _io_buff = (io_buffer *)&_io_buff->size_buffer;
LINE490                       } while (_look_index <
LINE491                                (int)(uint)*(ushort *)&mys_tags_data->ID_TIF_SAMPLES_PER_PIXEL);
LINE492                     }
LINE493                     local_c = loop_index_temp;
LINE494                     if (loop_index_temp != 0) break;
LINE495                     will_iIG_IP_planar_raster_unpack
LINE496                               (mys_tags_data,some_buffer,table_ptr_byte_from_file,
LINE497                                (uint *)p_size_memory_buffer_00,dest_buffer);
LINE498                     loop_index_temp =
LINE499                          IO_raster_set(mys_table_function,dest_buffer,higdibinfo,_raster_size);
LINE500                     higdibinfo = (HIGDIBINFO)((int)&higdibinfo->igdibstd_vftable + 1);
LINE501   LAB_101777e4:
LINE502                     pIVar3 = higdibinfo;
LINE503                     local_c = loop_index_temp;
LINE504                     if (loop_index_temp != 0) break;
LINE505                   }
LINE506                   strip = strip + 1;
LINE507                   pIVar3 = higdibinfo;
LINE508                   local_c = loop_index_temp;
LINE509                 } while (strip < (int)mys_tags_data->ID_TIF_ROWS_PER_STRIP);
LINE510               }
            [...]
LINE563   }
```

The size of `dest_buffer` which is represented by `_raster_size` is computed through the function `IO_raster_size_get` as you can see at `LINE377`

```
LINE565   uint IO_raster_size_get(HIGDIBINFO higdibinfo)
LINE566   {
LINE567     dword bit_depth;
LINE568     uint _raster_size;
LINE569
LINE570     bit_depth = IGDIBStd::DIB_bit_depth_get(higdibinfo);
LINE571     if (bit_depth == 1) {
LINE572                       /* return size_X + 0x1f >> 3 & 0xfffffffc; */
LINE573       _raster_size = DIB1bit_packed_raster_size_get(higdibinfo);
LINE574       return _raster_size;
LINE575     }
LINE576                       /* indirect call to some computer_raster_size */
LINE577     _raster_size = DIBStd_raster_size_get(higdibinfo);
LINE578     return _raster_size;
LINE579   }
```

The `_raster_size` is computed differently according to the `bit_depth` of the image. In our case the `bit_depth` is greater than '1' and so is computed through the function call `DIBStd_raster_size_get` at `LINE577`. This function is a wrapper to land finally into a function named `IGDIBStd::compute_raster_size` with the following pseudo-code:

```
LINE580  uint __thiscall IGDIBStd::compute_raster_size(HIGDIBINFO this)
LINE581  {
LINE582    longlong lVar1;
LINE583    uint uVar2;
LINE584    ulonglong uVar3;
LINE585
LINE586    lVar1 = (longlong)(int)(this->mys_struct_table_color).ptr_bits_per_channel_table *
LINE587            (longlong)(int)(this->mys_struct_table_color).ptr_channel_count;
LINE588    uVar3 = __allmul((uint)lVar1,(uint)((ulonglong)lVar1 >> 0x20),this->size_X,
LINE589                     (int)this->size_X >> 0x1f);
LINE590    lVar1 = (longlong)(uVar3 + 0x1f) >> 3;
LINE591    uVar2 = (uint)lVar1 & 0xfffffffc;
LINE592    if ((-1 < lVar1) && ((0 < (int)((longlong)(uVar3 + 0x1f) >> 0x23) || (0x7ffffffe < uVar2)))) {
LINE593      wrapper_thow_exception
LINE594                 ((undefined *)0xfffffe6f,(char *)0x0,(undefined *)0x0,(undefined *)0x0,
LINE595                  (undefined **)0x1022fa38,(undefined *)0x29);
LINE596    }
LINE597    return uVar2;
LINE598  }
```

The result which lead to ‘4’ is based on values taken from directly from the file and computed along the program and finally are associated into the following variables `ptr_bits_per_channel_table`, `ptr_channel_count` and `size_X`. The following TIFF tags must be present: - A planar configuration with a value of ‘2’. - A photometric tag with some specifics values

The value written into the memory are indirectly controlled from the bytes we can have from the malformed file and the size of the allocation buffer is also controlled through the values of the files.

Crash Information

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


KEY_VALUES_STRING: 1

    Key  : AV.Fault
    Value: Write

    Key  : Analysis.CPU.mSec
    Value: 2405

    Key  : Analysis.DebugAnalysisManager
    Value: Create

    Key  : Analysis.Elapsed.mSec
    Value: 29571

    Key  : Analysis.Init.CPU.mSec
    Value: 3624

    Key  : Analysis.Init.Elapsed.mSec
    Value: 32534

    Key  : Analysis.Memory.CommitPeak.Mb
    Value: 171

    Key  : Timeline.OS.Boot.DeltaSec
    Value: 279515

    Key  : Timeline.Process.Start.DeltaSec
    Value: 31

    Key  : WER.OS.Branch
    Value: vb_release

    Key  : WER.OS.Timestamp
    Value: 2019-12-06T14:06:00Z

    Key  : WER.OS.Version
    Value: 10.0.19041.1

    Key  : WER.Process.Version
    Value: 1.0.1.1


NTGLOBALFLAG:  2100000

APPLICATION_VERIFIER_FLAGS:  0

APPLICATION_VERIFIER_LOADED: 1

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 5c194b4c (igCore19d!IG_mpi_page_set+0x00008dfc)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 00000001
   Parameter[1]: 0b98d000
Attempt to write to address 0b98d000

FAULTING_THREAD:  0000254c

PROCESS_NAME:  Fuzzme.exe

WRITE_ADDRESS:  0b98d000

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:  0b98d000

STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
0019f4a0 5c142d4d     00000004 00000000 00000001 igCore19d!IG_mpi_page_set+0x8dfc
0019f4c4 5c29c17f     00000004 00000000 00000001 igCore19d!IG_thread_image_unlock+0x4b5d
0019f4f8 5c297712     0019f5c4 0b418d68 099e4ff0 igCore19d!IG_mpi_page_set+0x11042f
0019f544 5c29721a     0019fb30 10000021 12688ff0 igCore19d!IG_mpi_page_set+0x10b9c2
0019f56c 5c29c353     0019fb30 10000021 0019f5c4 igCore19d!IG_mpi_page_set+0x10b4ca
0019f594 5c2962cb     0019fb30 10000021 0b418d68 igCore19d!IG_mpi_page_set+0x110603
0019faa8 5c1610d9     0019fb30 0b418d68 00000001 igCore19d!IG_mpi_page_set+0x10a57b
0019fae0 5c1a0557     00000000 0b418d68 0019fb30 igCore19d!IG_image_savelist_get+0xb29
0019fd5c 5c19feb9     00000000 05414f88 00000001 igCore19d!IG_mpi_page_set+0x14807
0019fd7c 5c135777     00000000 05414f88 00000001 igCore19d!IG_mpi_page_set+0x14169
0019fd9c 00498a3a     05414f88 0019fe0c 004801a4 igCore19d!IG_load_file+0x47
0019fe14 00498e36     05414f88 0019fe8c 004801a4 Fuzzme!fuzzme+0x4a
0019fee4 004daa53     00000005 05354f20 0535df20 Fuzzme!main+0x376
0019ff04 004da8a7     b039b859 004801a4 004801a4 Fuzzme!invoke_main+0x33
0019ff60 004da73d     0019ff70 004daad8 0019ff80 Fuzzme!__scrt_common_main_seh+0x157
0019ff68 004daad8     0019ff80 75bbfa29 0035e000 Fuzzme!__scrt_common_main+0xd
0019ff70 75bbfa29     0035e000 75bbfa10 0019ffdc Fuzzme!mainCRTStartup+0x8
0019ff80 77d57a4e     0035e000 a98e2324 00000000 KERNEL32!BaseThreadInitThunk+0x19
0019ffdc 77d57a1e     ffffffff 77d788fd 00000000 ntdll!__RtlUserThreadStart+0x2f
0019ffec 00000000     004801a4 0035e000 00000000 ntdll!_RtlUserThreadStart+0x1b


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  igCore19d!IG_mpi_page_set+8dfc

MODULE_NAME: igCore19d

IMAGE_NAME:  igCore19d.dll

FAILURE_BUCKET_ID:  INVALID_POINTER_WRITE_AVRF_c0000005_igCore19d.dll!IG_mpi_page_set
```

```
OS_VERSION:  10.0.19041.1

BUILDLAB_STR:  vb_release

OSPLATFORM_TYPE:  x86

OSNAME:  Windows 10

IMAGE_VERSION:  19.9.0.0

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

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

Timeline

2021-05-10 - Vendor Disclosure
2021-05-31 - Vendor Patched

2021-06-01 - Public Release

CREDIT

Discovered by Emmanuel Tacheau of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2021-1276

NEXT REPORT TALOS-2021-1308