Talos Vulnerability Report

TALOS-2022-1544

Accusoft ImageGear PICT parsing pctwread_14841 out-of-bounds write vulnerability

OCTOBER 27, 2022

CVE NUMBER

CVE-2022-32588

SUMMARY

An out-of-bounds write vulnerability exists in the PICT parsing pctwread_14841 functionality of Accusoft ImageGear 20.0. A specially-crafted malformed file can lead to memory corruption. An attacker can provide a malicious file to trigger this vulnerability.

CONFIRMED VULNERABLE VERSIONS

The versions below were either tested or verified to be vulnerable by Talos or confirmed to be vulnerable by the vendor.

Accusoft ImageGear 20.0

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-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 `pctwread_14841` function, due to a buffer overflow caused by a missing buffer size check.
A specially-crafted PICT file can lead to an out-of-bounds write, which can result in memory corruption.

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

```
(1fac.7dc): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0d10cfff ebx=0d108f60 ecx=00000078 edx=0000402f esi=00000002 edi=00000020
eip=6d848a98 esp=0019f664 ebp=0019f670 iopl=0         nv up ei pl nz na pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010207
igcore20d!IG_mpi_page_set+0xdc908:
6d848a98 884801          mov     byte ptr [eax+1],cl        ds:002b:0d10d000=??
```

The issue is happening in the following code at LINE12, where we can see `dest_buffer` is written with bytes from `src_buffer`, controlled through a do-while loop at LINE10-LINE17 with the `length` passed in parameter.

```
LINE1  byte * __cdecl pict_perform_some_copy_148a60(byte *dest_buffer,byte
*src_buffer,int length)
LINE2  {
LINE3    byte *_dest_buffer;
LINE4    ushort uVar1;
LINE5    int index;
LINE6
LINE7    index = 0;
LINE8    if (0 < length) {
LINE9      _dest_buffer = dest_buffer + 1;
LINE10     do {
LINE11       uVar1 = CONCAT11(src_buffer[index * 2],src_buffer[index * 2 + 1]);
LINE12       _dest_buffer[1] = src_buffer[index * 2 + 1] << 3;
LINE13       index = index + 1;
LINE14       *_dest_buffer = (byte)(uVar1 >> 2) & 0xf8;
LINE15       _dest_buffer[-1] = (byte)(uVar1 >> 7) & 0xf8;
LINE16       _dest_buffer = _dest_buffer + 3;
LINE17     } while (index < length);
LINE18   }
LINE19   return _dest_buffer;
LINE20 }
```

When we look at the `eax` memory allocation corresponding to our buffer `dest_buffer` seen previously, we can see the buffer allocated is very small, only 1 byte:

```
0:000> !heap -p -a eax
address 0d10cfff found in
_DPH_HEAP_ROOT @ 2cb1000
in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -
VirtAddr         VirtSize)
                              d1608f0:          d10cff8                1 -
d10c000             2000
6dbca8b0 verifier!AVrfDebugPageHeapAllocate+0x00000240
77c0f10e ntdll!RtlDebugAllocateHeap+0x00000039
77b770f0 ntdll!RtlpAllocateHeap+0x000000f0
77b76e3c ntdll!RtlpAllocateHeapInternal+0x0000104c
77b75dde ntdll!RtlAllocateHeap+0x0000003e
6d50daff MSVCR110!malloc+0x00000049
6d76663e igcore20d!AF_memm_alloc+0x0000001e
6d8484f7 igcore20d!IG_mpi_page_set+0x000dc367
6d847991 igcore20d!IG_mpi_page_set+0x000db801
6d741399 igcore20d!IG_image_savelist_get+0x00000b29
6d7809e7 igcore20d!IG_mpi_page_set+0x00014857
6d780349 igcore20d!IG_mpi_page_set+0x000141b9
6d715777 igcore20d!IG_load_file+0x00000047
00402239 Fuzzme!fuzzme+0x00000019
00402544 Fuzzme!fuzzme+0x00000324
004062a0 Fuzzme!fuzzme+0x00004080
762ffa29 KERNEL32!BaseThreadInitThunk+0x00000019
77b97a9e ntdll!__RtlUserThreadStart+0x0000002f
77b97a6e ntdll!_RtlUserThreadStart+0x0000001b
```

Investigating the callstack for the memory allocation and especially the `igcore20d!IG_mpi_page_set+0x000dc367` leads us into a function named `pctwread_148410` at LINE95 with the following pseudo-code:

```
LINE21  AT_ERRCOUNT
LINE22  pctwread_148410(mys_table_function *mys_table_fun,uint heap,undefined
param_3,
LINE23                  pict_header *pict_header,HIGDIBINFO higdibinfo)
LINE24  {
        [...]
LINE73    raster_size = IO_raster_size_get(higdibinfo);
LINE74    bloc_of_pixel? = pict_header->bloc_of_pixels?;
LINE75    buffer_from_file.size_buffer =
LINE76         (int)*(short *)((int)&bloc_of_pixel?->bounds_bottom + 2) -
LINE77         (int)*(short *)((int)&bloc_of_pixel?->bounds_top + 2);
LINE78    size_buffer_2 = buffer_from_file.size_buffer * 5;
LINE79    if ((int)size_buffer_2 < (int)raster_size) {
LINE80
AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\pctwread.c",0x2f5,-0x194,0,raste
r_size,
LINE81                        buffer_from_file.size_buffer,(LPCHAR)0x0);
LINE82    }
LINE83    else {
        [...]
LINE95      buffer_raster = (byte *)AF_memm_alloc(saved_heap,raster_size);
LINE96      size_to_read = saved_heap;
LINE97      src_buffer = (byte *)AF_memm_alloc(saved_heap,size_buffer_2);
LINE98      raster_size_by_heigth = dup_raster_size * heigth;
LINE99      local_2c = (byte *)AF_memm_alloc(size_to_read,raster_size_by_heigth);
LINE100     local_68 = AF_memm_alloc(size_to_read,raster_size_by_heigth);
LINE101     dup_buffer_raster = buffer_raster;
LINE102     if (((buffer_raster == (byte *)0x0) || (src_buffer == (byte *)0x0)) ||
(local_2c == (byte *)0x0)
LINE103        ) {
LINE104       err_status_iodibcreate_iob_init =
LINE105
AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\pctwread.c",0x308,-1000,0,0,0,
LINE106                                (LPCHAR)0x0);
LINE107     }
LINE108     else if ((pixelSize#1 == 0x20) && (cmpCount == 4)) {
LINE109       local_38 = True;
LINE110     }
LINE111     dup_src_buffer = local_2c;
LINE112     OS_memset(local_2c,0,raster_size_by_heigth);
LINE113     err_status_iob_init =
IOb_init(mys_table_fun,saved_heap,&buffer_from_file,0x5000,1);
LINE114     ppVar3 = saved_pict_header;
LINE115     err_status_iodibcreate_iob_init = err_status_iodibcreate_iob_init +
err_status_iob_init;
LINE116     if (err_status_iodibcreate_iob_init == 0) {
LINE117       IO_attribute_set(mys_table_fun,4,
LINE118                        (AT_RESOLUTION *)&saved_pict_header-
>horizontal_pixel_per_inch);
LINE119       cmpCount = 0;
LINE120       _index_bloc_data = 0;
LINE121       do {
LINE122         if ((int)ppVar3->field15_size_buffer <= (int)cmpCount) break;
LINE123         bloc_of_pixel? = ppVar3->bloc_of_pixels?;
LINE124         pbVar1 = (bits_per_channel_table *)
LINE125                  (uint)*(ushort *)((int)&bloc_of_pixel?->cmpCount +
_index_bloc_data);
LINE126         if ((bits_per_channel_table *)0x2 < pbVar1) {
```

```
LINE127           pbVar1 = (bits_per_channel_table *)0x3;
LINE128         }
LINE129         iVar2 = 0;
LINE130         if (pbVar1 != (bits_per_channel_table *)0x0) {
LINE131           do {
LINE132             local_14[iVar2] = (uint)*(ushort *)((int)&bloc_of_pixel?-
>cmpSize + _index_bloc_data);
LINE133             iVar2 = iVar2 + 1;
LINE134           } while (iVar2 < (int)pbVar1);
LINE135         }
LINE136         size_to_read = (int)*(short *)((int)&(bloc_of_pixel?-
>dstRect.bottom).srcRect.right +
LINE137                                        _index_bloc_data) -
LINE138                         (int)*(short *)((int)&(bloc_of_pixel?-
>dstRect.top).srcRect.left +
LINE139                                        _index_bloc_data);
LINE140         local_70 = (int)*(short *)((int)&(bloc_of_pixel?-
>dstRect.bottom).srcRect.bottom +
LINE141                                      _index_bloc_data) -
LINE142                       (int)*(short *)((int)&(bloc_of_pixel?-
>dstRect.top).srcRect.top +
LINE143                                      _index_bloc_data);
LINE144         dup#2_size_to_read = size_to_read;
LINE145         IO_raster_size_calc(size_to_read,pbVar1,(int *)local_14);
LINE146         if (pixelSize#1 == 1) {
LINE147           size_to_read = DIB1bit_packed_raster_size_calc(size_to_read);
LINE148         }
LINE149         else {
LINE150           size_to_read = DIBStd_raster_size_calc_simple
LINE151                                    (size_to_read,pbVar1,
LINE152                                     (uint)*(ushort *)
LINE153                                            ((int)&saved_pict_header-
>bloc_of_pixels?->cmpSize +
LINE154                                            _index_bloc_data));
LINE155         }
LINE156         bloc_of_pixel? = saved_pict_header->bloc_of_pixels?;
LINE157         local_5c = (int)*(short *)((int)&bloc_of_pixel?->bounds_bottom +
_index_bloc_data + 2) -
LINE158                       (int)*(short *)((int)&bloc_of_pixel?->bounds_top +
_index_bloc_data + 2);
LINE159         local_40 = size_to_read;
LINE160         IOb_seek(&buffer_from_file,*(int *)(&bloc_of_pixel?->field_0x3c +
_index_bloc_data),0);
LINE161         dup_buffer_raster = buffer_raster;
LINE162         OS_memset(buffer_raster,0,raster_size);
LINE163         if (pixelSize#1 != 0x18) {
LINE164           size_to_read = (uint)*(ushort *)
LINE165                                    ((int)&saved_pict_header->bloc_of_pixels?->
LINE166                                           rowBytes_value_without_pixmapflag +
_index_bloc_data);
LINE167         }
LINE168         local_64 = (uint)*(ushort *)
LINE169                             ((int)&saved_pict_header->bloc_of_pixels?-
>packType + _index_bloc_data);
LINE170         local_54 = 0;
LINE171         dup_size_to_read = size_to_read;
LINE172         if (err_status_iodibcreate_iob_init == 0) {
LINE173           for (; size_to_read = dup_size_to_read, local_54 < local_70;
local_54 = local_54 + 1) {
```

```
LINE174             if (((7 < (int)dup_size_to_read) && (local_64 != 1)) &&
LINE175                ((local_64 != 2 || ((int)pixelSize#1 < 0x18)))) {
LINE176               if ((int)dup_size_to_read < 0xfb) {
LINE177                 iVar2 = IOb_byte_read(&buffer_from_file,&local_25);
LINE178                 if (iVar2 != 0) {
LINE179                   size_to_read = (uint)local_25;
LINE180                   goto LAB_1014872b;
LINE181                 }
LINE182                 iVar2 = 900;
LINE183               }
LINE184               else {
LINE185                 iVar2 = read_short_05c350(&buffer_from_file,local_58);
LINE186                 if (iVar2 == 0) {
LINE187                   iVar2 = 0x37b;
LINE188                 }
LINE189                 else {
LINE190                   size_to_read = (uint)local_58[0];
LINE191 LAB_1014872b:
LINE192                   data_from_file = (byte
*)get_data_from_file(&buffer_from_file,size_to_read);
LINE193                   dup_src_buffer = src_buffer;
LINE194                   if (data_from_file != (byte *)0x0) {
LINE195                     switch(pixelSize#1) {
LINE196                     case 1:
LINE197                     case 8:
LINE198
FUN_10149d50(dup_buffer_raster,data_from_file,size_to_read,local_40);
LINE199                       break;
LINE200                     case 4:
LINE201                       iVar2 = (int)raster_size / 2;
LINE202                       dup_buffer_raster = buffer_raster + iVar2;
LINE203
FUN_10149d50(dup_buffer_raster,data_from_file,size_to_read,local_40);
LINE204                       pixelSize#1 = dup_pixelSize#1;
LINE205
FUN_10022e70(1,dup_pixelSize#1,buffer_from_file.size_buffer,dup_buffer_raster,
LINE206                                       buffer_raster,iVar2);
LINE207                       break;
LINE208                     case 0x10:
LINE209                       pict_perform_some_copy_149C70
LINE210
(src_buffer,data_from_file,size_to_read,dup_size_to_read);
LINE211                       pict_perform_some_copy_148a60
LINE212
(dup_buffer_raster,dup_src_buffer,dup#2_size_to_read);
LINE213                       break;
LINE214                     case 0x18:
LINE215                     case 0x20:
LINE216
FUN_10149d50(src_buffer,data_from_file,size_to_read,dup_size_to_read);
LINE217                       FUN_10148ac0(dup_buffer_raster,
(int)dup_src_buffer,dup#2_size_to_read,local_5c
LINE218                                       ,local_38);
LINE219                     }
LINE220                     goto switchD_1014880a_caseD_2;
LINE221                   }
LINE222                   iVar2 = 0x38c;
LINE223                 }
LINE224               }
```

```
LINE225 LAB_1014891b:
LINE226               err_status_iodibcreate_iob_init =
LINE227
AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\pctwread.c",iVar2,-0x834,0,0,
LINE228                                        0,(LPCHAR)0x0);
LINE229               break;
LINE230             }
[...]
LINE270           }
LINE271         }
LINE272         cmpCount = cmpCount + 1;
LINE273         _index_bloc_data = _index_bloc_data + 0x440;
LINE274         ppVar3 = saved_pict_header;
LINE275       } while (err_status_iodibcreate_iob_init == 0);
[...]
LINE290     }
[...]
LINE305   }
[...]
LINE309 }
```

The computation of the size for our buffer `buffer_raster` is identified by the variable `raster_size` at LINE73 and made through a call to `IO_raster_size_get`. In our case we can see `raster_size` set to 0, making the buffer 1 byte only.

We have to investigate through several nested functions to identify where `raster_size` is computed, starting from function `IO_raster_size_get` :

```
LINE310 uint IO_raster_size_get(HIGDIBINFO higdibinfo)
LINE311 {
        [...]
   /* indirect call to some computer_raster_size */
LINE322   _raster_size = DIBStd_raster_size_get(higdibinfo);
LINE323   return _raster_size;
LINE324 }
```

The function `IO_raster_size_get`at LINE322 is calling the function `DIBStd_raster_size_get`:

```
LINE325 AT_INT DIBStd_raster_size_get(HIGDIBINFO hdib)
LINE326 {
    [...]
LINE344   uVar1 = (*hdib->igdibstd_vftable->IGDIBStd::compute_raster_size)(hdib);
LINE345   *in_FS_OFFSET = local_10;
LINE346   return uVar1;
LINE347 }
```

This function is calling a function named `compute_raster_size` at LINE344 with the following pseudo-code :

```
LINE348 uint __thiscall IGDIBStd::compute_raster_size(IGDIBRunEnds *this)
LINE349 {
LINE350   uint _raster_size;
LINE351   ulonglong uVar1;
LINE352   longlong _computed_raster_size;
LINE353   longlong _bits_channel;
LINE354
LINE355   _bits_channel =
LINE356        (longlong)(int)this->depth_round_value *
LINE357        (longlong)(int)(this->table_color).ptr_bits_per_channel_table;
LINE358   uVar1 = __allmul((uint)_bits_channel,(uint)((ulonglong)_bits_channel >>
0x20),this->biHeigth,
LINE359                    (int)this->biHeigth >> 0x1f);
LINE360   _computed_raster_size = (longlong)(uVar1 + 0x1f) >> 3;
LINE361   _raster_size = (uint)_computed_raster_size & 0xfffffffc;
LINE362   if ((-1 < _computed_raster_size) &&
LINE363      ((0 < (int)((longlong)(uVar1 + 0x1f) >> 0x23) || (0x7ffffffe <
_raster_size)))) {
LINE364     wrapper_thow_exception
LINE365               ((undefined *)0xfffffe6f,(char *)0x0,(undefined *)0x0,
(undefined *)0x0,
LINE366                (undefined **)0x10230a38,(undefined *)0x29);
LINE367   }
LINE368   return _raster_size;
LINE369 }
```

Finally we can observe that the returned value `_raster_size` at LINE368 is derived from `_computed_raster_size` on LINE361, which is itself computed from a multiplication between `_bits_channel` and `this->biHeigth` at LINE358.
To get a null product, one of the two multipliers must be null, which in our case is `this->biHeigth`. Now in order to identify where is `this->biHeigth` is coming from, you may put a breakpoint on access memory using a debugger with a recorded trace, for example, or put some breakpoints backward and so on.

Using one of the two methods, we'll end into a function I named `pctwread_copypalette_1482a0` with the following pseudo-code:

```
 LINE370 AT_ERRCOUNT pctwread_copypalette_1482a0(pict_header *pict_header,LPHDIB
lphDib)
LINE371 {
[...]
LINE385   biBitCount = 0;
LINE386   if (pict_header->possible_v2_type == 0) {
LINE387     pict_header->possible_v2_type = 3;
LINE388     pict_header->field10_0x20 = 1;
LINE389     pict_header->field12_0x28 = 1;
LINE390     uVar3 = pict_header->bloc_of_pixels?->hRes;
LINE391     uVar1 = uVar3 >> 0x10;
LINE392     if (uVar1 != 0) {
LINE393       uVar3 = uVar1;
LINE394     }
LINE395     pict_header->horizontal_pixel_per_inch = uVar3;
LINE396     uVar3 = pict_header->bloc_of_pixels?->vRes;
LINE397     uVar1 = uVar3 >> 0x10;
LINE398     if (uVar1 != 0) {
LINE399       uVar3 = uVar1;
LINE400     }
LINE401     pict_header->vertical_pixel_per_inch = uVar3;
LINE402   }
LINE403   if (pict_header->v2type == 0xfffe0000) {
LINE404     lower_right_x = (pict_header-
>optimal_src_rectangle_72dpi).frame_lower_right_x;
LINE405     biHeigth = (int)(short)(pict_header-
>optimal_src_rectangle_72dpi).frame_lower_right_y -
LINE406                (int)(short)(pict_header-
>optimal_src_rectangle_72dpi).frame_top_left_y;
LINE407     top_left_x = (pict_header-
>optimal_src_rectangle_72dpi).frame_top_left_x;
LINE408   }
LINE409   else {
LINE410     lower_right_x = (pict_header->frame_info).frame_lower_right_x;
LINE411     biHeigth = (int)(short)(pict_header->frame_info).frame_lower_right_y -
LINE412                (int)(short)(pict_header->frame_info).frame_top_left_y;
LINE413     top_left_x = (pict_header->frame_info).frame_top_left_x;
LINE414   }
LINE415   _biWidth = (int)(short)lower_right_x - (int)(short)top_left_x;
LINE416   if ((int)biHeigth < 0) {
LINE417     biHeigth = -biHeigth;
LINE418   }
[...]
LINE438   _status = DIB_info_create_cnvt_res
LINE439                       (lphDib,biHeigth,_biWidth,biBitCount,0,
LINE440                        *(undefined8 *)&pict_header-
>horizontal_pixel_per_inch,
LINE441                        *(undefined8 *)&pict_header->vertical_pixel_per_inch,
LINE442                        pict_header->possible_v2_type);
LINE443   if ((_status == 0) && (_biWidth = DIB_colorspace_get(*lphDib), _biWidth ==
3)) {
LINE444     call_IGDIB::DIB_palette_alloc(*lphDib);
LINE445     _Size = DIB_palette_size_get(*lphDib);
LINE446     _Src = &pict_header->bloc_of_pixels?->field25_0x40;
LINE447     _Dst = (void *)DIB_palette_pointer_get(*lphDib);
LINE448     memcpy(_Dst,_Src,_Size);
LINE449   }
LINE450   AVar2 = AF_error_check();
```

```
LINE451   return AVar2;
LINE452 }
```

In our case, the interesting `biHeigth` is computed at LINE405, which is the result of a subtraction of two variables: `frame_lower_right_y` and `frame_top_left_y`. In all cases where theses two variables are equal or null, the subtraction is null, thus `biHeigth`.
Earlier, we saw in `pict_perform_some_copy_148a60` at LINE17 the do-while loop is controlled by the `length` variable passed in parameter, corresponding to `dup#2_size_to_read` at LINE212 in the function `pctwread_148410`. This variable is a duplicate of `size_to_read` at LINE144, which is also read from the pict file.

Thus a missing check for a minimum size in the code is enabling the vulnerability to trigger. The assignments happening inside that function are out-of-bounds heap writes which lead to memory corruption and possibly code execution.
The pict file after research must have the following constraint: it must be a pict v2 extended file type. The processing of opcode `DirectBitsRect` or `PackBitsRect` is prone to the vulnerability.

Crash Information

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


KEY_VALUES_STRING: 1

    Key  : AV.Fault
    Value: Write

    Key  : Analysis.CPU.mSec
    Value: 5562

    Key  : Analysis.DebugAnalysisManager
    Value: Create

    Key  : Analysis.Elapsed.mSec
    Value: 14140

    Key  : Analysis.Init.CPU.mSec
    Value: 7530

    Key  : Analysis.Init.Elapsed.mSec
    Value: 90474

    Key  : Analysis.Memory.CommitPeak.Mb
    Value: 163

    Key  : Timeline.OS.Boot.DeltaSec
    Value: 382110

    Key  : Timeline.Process.Start.DeltaSec
    Value: 32

    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: 1.0.2.0


NTGLOBALFLAG:  2000000

PROCESS_BAM_CURRENT_THROTTLED: 0

PROCESS_BAM_PREVIOUS_THROTTLED: 0

APPLICATION_VERIFIER_FLAGS:  0
```

```
APPLICATION_VERIFIER_LOADED: 1

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 6d848a98 (igcore20d!IG_mpi_page_set+0x000dc908)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 00000001
   Parameter[1]: 0d10d000
Attempt to write to address 0d10d000

FAULTING_THREAD:  000007dc

PROCESS_NAME:  Fuzzme.exe

WRITE_ADDRESS:  0d10d000

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:  0d10d000

STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
0019f670 6d8487b9     0d10cff8 0d108f60 00000020 igcore20d!IG_mpi_page_set+0xdc908
0019f734 6d847991     0019fc3c 1000001e 0ab58ff8 igcore20d!IG_mpi_page_set+0xdc629
0019fbb4 6d741399     0019fc3c 0ab58ff8 00000001 igcore20d!IG_mpi_page_set+0xdb801
0019fbec 6d7809e7     00000000 0ab58ff8 0019fc3c
igcore20d!IG_image_savelist_get+0xb29
0019fe68 6d780349     00000000 052acfd0 00000001 igcore20d!IG_mpi_page_set+0x14857
0019fe88 6d715777     00000000 052acfd0 00000001 igcore20d!IG_mpi_page_set+0x141b9
0019fea8 00402239     052acfd0 0019febc 762ff550 igcore20d!IG_load_file+0x47
0019fec0 00402544     052acfd0 0019fef8 0520cf48 Fuzzme!fuzzme+0x19
0019ff28 004062a0     00000005 05206f88 0520cf48 Fuzzme!fuzzme+0x324
0019ff70 762ffa29     003f2000 762ffa10 0019ffdc Fuzzme!fuzzme+0x4080
0019ff80 77b97a9e     003f2000 8b0919bf 00000000 KERNEL32!BaseThreadInitThunk+0x19
0019ffdc 77b97a6e     ffffffff 77bb8a56 00000000 ntdll!__RtlUserThreadStart+0x2f
0019ffec 00000000     00406328 003f2000 00000000 ntdll!_RtlUserThreadStart+0x1b


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  igcore20d!IG_mpi_page_set+dc908

MODULE_NAME: igcore20d

IMAGE_NAME:  igcore20d.dll

FAILURE_BUCKET_ID:
INVALID_POINTER_WRITE_AVRF_c0000005_igcore20d.dll!IG_mpi_page_set

OS_VERSION:  10.0.19041.1

BUILDLAB_STR:  vb_release
```

```
OSPLATFORM_TYPE:  x86

OSNAME:  Windows 10

IMAGE_VERSION:  20.0.0.0

FAILURE_ID_HASH:  {fe8f80f8-683f-d41f-7c33-712a409d5fb5}

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

TIMELINE

2022-06-22 - Vendor Disclosure

2022-10-25 - Vendor Patch Release

2022-10-27 - Public Release

CREDIT

Discovered by Emmanuel Tacheau of Cisco Talos.
