Talos Vulnerability Report

TALOS-2021-1248

Accusoft ImageGear JPG format SOF marker processing out-of-bounds write vulnerability

MARCH 2, 2021

CVE NUMBER

CVE-2021-21784

Summary

An out-of-bounds write vulnerability exists in the JPG format SOF marker processing of Accusoft ImageGear 19.8. A specially crafted malformed file can lead to memory corruption. An attacker can provide a malicious file to trigger this vulnerability.

Tested Versions

Accusoft ImageGear 19.8

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.

A write access violation can happen in the `handle_color_channel_with_high_precision` function, due to a buffer overflow caused by a missing size check for a buffer memory. A specially crafted JPG file can lead to an out-of-bounds write which can result in memory corruption.

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

```
(f4c4.216c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=ffffba40 ebx=00000000 ecx=ffffd7d7 edx=0b353004 esi=00000fff edi=ffffddd5
eip=5bfe5549 esp=0019f728 ebp=0019f740 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+0xc97f9:
5bfe5549 66895afc        mov     word ptr [edx-4],bx      ds:002b:0b353000=????
```

This crash happens at LINE42 in the following pseudo-code of the function `handle_color_channel_with_high_precision`:

```
LINE1   void handle_color_channel_with_high_precision
LINE2                 (undefined4 param_1,int width,short *param_3,short *param_4,short *param_5,
LINE3                 short *raster_buffer,int param_7,int param_8)
LINE4   {
LINE5   [...]
LINE6     if (0 < width) {
LINE7       raster_buffer = raster_buffer + 2;
LINE8       do {
LINE9         short_3 = *param_3 + 0x800;
LINE10        iVar1 = *param_5 * 0x166e;
LINE11        iVar2 = *param_4 * 0x1c5a;
LINE12        short_1 = *param_4 * 0x582 + *param_5 * 0xb6d;
LINE13        short_2 = short_3 - ((int)(short_1 + (short_1 >> 0x1f & 0xfffU)) >> 0xc);
LINE14        short_1 = ((int)(iVar2 + (iVar2 >> 0x1f & 0xfffU)) >> 0xc) + short_3;
LINE15        short_3 = ((int)((iVar1 >> 0x1f & 0xfffU) + iVar1) >> 0xc) + short_3;
LINE16        if (short_3 < 0) {
LINE17          short_3 = 0;
LINE18        }
LINE19        else {
LINE20          if (0xfff < short_3) {
LINE21            short_3 = 0xfff;
LINE22          }
LINE23        }
LINE24        raster_buffer[-2] = (short)short_3;
LINE25        if (short_2 < 0) {
LINE26          short_2 = 0;
LINE27        }
LINE28        else {
LINE29          if (0xfff < short_2) {
LINE30            short_2 = 0xfff;
LINE31          }
LINE32        }
LINE33        raster_buffer[-1] = (short)short_2;
LINE34        if (short_1 < 0) {
LINE35          short_1 = 0;
LINE36        }
LINE37        else {
LINE38          if (0xfff < short_1) {
LINE39            short_1 = 0xfff;
LINE40          }
LINE41        }
LINE42        *raster_buffer = (short)short_1;
LINE43        local_8 = local_8 + *(int *)(param_7 + 0x34);
LINE44        if (local_8 == param_8) {
LINE45          param_3 = param_3 + 1;
LINE46          local_8 = 0;
LINE47        }
LINE48        local_c = local_c + *(int *)(param_7 + 0x84);
LINE49        if (local_c == param_8) {
LINE50          param_4 = param_4 + 1;
LINE51          local_c = 0;
LINE52        }
LINE53        local_10 = local_10 + *(int *)(param_7 + 0xd4);
LINE54        if (local_10 == param_8) {
LINE55          local_10 = 0;
LINE56          param_5 = param_5 + 1;
LINE57        }
LINE58        raster_buffer = raster_buffer + 3;
LINE59        width = width + -1;
LINE60      } while (width != 0);
LINE61    }
LINE62    return;
LINE63  }
```

From the pseudo code we can easily see the write into `raster_buffer` is performed by a do-while loop from LINE8 to LINE60, which is controlled by a variable named `width`. When the size of the allocated buffer `raster_buffer` is smaller than the width multiplied by `3` (used to store the three short), an out-of-bounds write can occur.

In our case the buffer's size is 0x408 bytes, as we can see below:

```
0:000> !heap -p -a edx
    address 0b353004 found in
    _DPH_HEAP_ROOT @ 4d91000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                                 b2f0820:          b352bf8              408 -          b352000             2000
    5c20a8b0 verifier!AVrfDebugPageHeapAllocate+0x00000240
    7765ef8e ntdll!RtlDebugAllocateHeap+0x00000039
    775c6150 ntdll!RtlpAllocateHeap+0x000000f0
    775c57fe ntdll!RtlpAllocateHeapInternal+0x000003ee
    775c53fe ntdll!RtlAllocateHeap+0x0000003e
    5bdddcff MSVCR110!malloc+0x00000049
    5bf161de igCore19d!AF_memm_alloc+0x0000001e
    5bfe334d igCore19d!IG_mpi_page_set+0x000c75fd
    5bfd6152 igCore19d!IG_mpi_page_set+0x000ba402
    5bfd005b igCore19d!IG_mpi_page_set+0x000b430b
    5bfe7025 igCore19d!IG_mpi_page_set+0x000cb2d5
    5bfe6ef5 igCore19d!IG_mpi_page_set+0x000cb1a5
    5bfe4f91 igCore19d!IG_mpi_page_set+0x000c9241
    5bfe66ea igCore19d!IG_mpi_page_set+0x000ca99a
    5bef10d9 igCore19d!IG_image_savelist_get+0x00000b29
    5bf30557 igCore19d!IG_mpi_page_set+0x00014807
    5bf2feb9 igCore19d!IG_mpi_page_set+0x00014169
    5bec5777 igCore19d!IG_load_file+0x00000047
    004021f9 Fuzzme!fuzzme+0x00000019
    00402504 Fuzzme!fuzzme+0x00000324
    0040666d Fuzzme!fuzzme+0x0000448d
    75c8fa29 KERNEL32!BaseThreadInitThunk+0x00000019
    775e75f4 ntdll!__RtlUserThreadStart+0x0000002f
    775e75c4 ntdll!_RtlUserThreadStart+0x0000001b
```

In this case the `width` variable which is controlling the loop in LINE60 is taken directly from the `width` value of a `SOF` marker and the value was `0x150`.
Let's see how information is carried with a SOF marker:

```
2 bytes Marker Identifier (0xFFCx)
2 bytes corresponding to the `length` of data for the marker
1 byte  corresponding to the `precision` aka bits/sample and value can be 8, 12 or 16 (depending of each type of SOFx)
2 bytes corresponding to the Image `height`
2 bytes corresponding to the Image `width`
1 byte corresponding to the number of components  like 1 for grayscale, 3 for color YCbCr, 4 for color CMYK
data left for each component.
```

The values in `SOF` marker are used to compute the size of the `raster_buffer` with a formula like:

```
((((precision * component * width) + 0x1f) / 8) & 0xFFFFFFFC) + 24
```

So effectively in function `handle_color_channel_with_high_precision` we can see the loop is going to overwrite the buffer as the buffer should have been sized at least `0x150 * 6 = 0x7e0` instead of `0x408`.

The `SOFx` marker values are read from the file with the function `parse_SOF` with pseudo code below :

```
LINE64   int parse_SOF(jpeg_dec *jpeg_dec,read_buffer *read_buffer,SOF_object *SOF_object)
LINE65   {
        [...]
LINE73
LINE74     kind_of_heap = jpeg_dec->kind_of_heap;
LINE75     data_from_file = (byte *)read_buffer->read_buffer_data;
LINE76     _index = 0;
LINE77     if (SOF_object == NULL) {
LINE78       _index = AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\jpeg_dec.c",0x476,-0xd02,0,0,0,NULL
LINE79                                  );
LINE80       return _index;
LINE81     }
LINE82     wrapper_memset(SOF_object,0,0x14);
LINE83     (SOF_object->SOF).precision = (uint)*data_from_file;
LINE84     (SOF_object->SOF).height =
LINE85          (uint)(ushort)(*(ushort *)(data_from_file + 1) << 8 | *(ushort *)(data_from_file + 1) >> 8);
LINE86     (SOF_object->SOF).width =
LINE87          (uint)(ushort)(*(ushort *)(data_from_file + 3) << 8 | *(ushort *)(data_from_file + 3) >> 8);
LINE88     (SOF_object->SOF).component = (uint)data_from_file[5];
LINE89     if ((jpeg_dec->type_of_SOF == 0) &&
LINE90        ((SOF_object->SOF).precision == 16
LINE91                        /* Enforce 8bit per channel */)) {
LINE92       AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\jpeg_dec.c",0x488,-0x128e,1,0x10,0x10,
LINE93                         "Precision of 16 is not allowed in JPEG Lossy, reading as 8");
LINE94       (SOF_object->SOF).precision = 8;
LINE95     }
LINE96     num_component = (SOF_object->SOF).component;
LINE97     pCVar1 = (ColorComponent *)
LINE98              AF_memm_alloc(kind_of_heap,num_component << 4,
LINE99                            (dword)"..\\..\\..\\..\\Common\\Formats\\jpeg_dec.c");
LINE100    (SOF_object->SOF).colorComponents = pCVar1;
            [...]
LINE139    return _index;
LINE140  }
```

We can see the storage of `precision`, `height`, `width` and `component` are read from the file respectively in LINE83, LINE84, LINE86 and LINE88.

While this is a very long process we can summarize it by saying that the size for the buffer allocation is taken from an `HDIB` object. This object is created while parsing the SOF marker in function `create_HDIB_from_SOF` through the call to `CreateLPHDIB` in LINE335.
The `precision`, derived from `precision_bit_depth`, is read from the file at LINE135. The `width` and `component` are respectively read from the file at LINE149 and LINE151.

```
LINE64   void create_HDIB_from_SOF(jpeg_related *table_fields_name,uint jpeg_marker,undefined4 _store_result,
LINE65                     byte *buffer_data_from_file,jpeg_dec *jpeg_dec,AT_MODE FORMAT_ID)
LINE66   {
            [...]
LINE100
LINE101    _jpeg_dec = jpeg_dec;
LINE102    _SOF_Marker_type = NULL;
LINE103    _kind_of_heap = jpeg_dec->kind_of_heap;
LINE104    SOF_DATASIZE = NULL;
LINE105    ___num_component = NULL;
LINE106    iVar1 = FUN_100422f0(0,0x15,&___num_component);
LINE107    iVar2 = FUN_10042460(0,(uint *)0x15,(int)"ACTUAL_BIT_DEPTH",0,NULL,NULL,&___buffer_data_from_file,
LINE108                         4,(uint)___num_component);
LINE109    uVar8 = iVar1 + iVar2;
LINE110    local_30 = uVar8;
LINE111    if (___num_component != NULL) {
LINE112      FUN_10042420(___num_component);
LINE113    }
LINE114    switch(jpeg_marker & 0xffff) {
LINE115    case 0xffc0:
LINE116      _SOF_Marker_type = "SOF0";
LINE117      table_fields_name->SOF_TYPE = 0;
LINE118      SOF_DATASIZE = "SOF0_DATASIZE";
LINE119      break;
LINE120    case 0xffc1:
LINE121      _SOF_Marker_type = "SOF1";
LINE122      table_fields_name->SOF_TYPE = 0;
LINE123      SOF_DATASIZE = "SOF1_DATASIZE";
LINE124      break;
LINE125    case 0xffc2:
LINE126      _SOF_Marker_type = "SOF2";
LINE127      table_fields_name->SOF_TYPE = 2;
LINE128      SOF_DATASIZE = "SOF2_DATASIZE";
LINE129      break;
LINE130    case 0xffc3:
LINE131      _SOF_Marker_type = "SOF3";
LINE132      table_fields_name->SOF_TYPE = 1;
LINE133      SOF_DATASIZE = "SOF3_DATASIZE";
LINE134    }
LINE135    precision_bit_depth = (byte *)(uint)*buffer_data_from_file;
LINE136    if (___buffer_data_from_file != NULL) {
LINE137      precision_bit_depth = (byte *)((uint)___buffer_data_from_file & 0xff);
LINE138    }
LINE139    precision = (byte)precision_bit_depth;
LINE140    if (((precision != 8) && (precision != 12)) && (precision != 16)) {
LINE141      buffer_data_from_file = precision_bit_depth;
LINE142      AF_err_record_set("..\\..\\..\\..\\Common\\Formats\\jpgread.c",0x671,-0x12e3,0,
LINE143                        precision_bit_depth,8,"Unsupported bit depth for channel.");
LINE144      AF_error_check();
LINE145      return;
LINE146    }
LINE147    heigth = (uint)(ushort)(*(ushort *)(buffer_data_from_file + 1) << 8 |
LINE148                            *(ushort *)(buffer_data_from_file + 1) >> 8);
LINE149    width = (uint)(ushort)(*(short *)(buffer_data_from_file + 3) << 8 |
LINE150                           (ushort)buffer_data_from_file[4]);
LINE151    component = buffer_data_from_file[5];
           [...]
LINE332      LPHDIB = (LPHIGDIBINFO)&table_fields_name->HDIB;
LINE333                      /* Will create color table related data
LINE334                         */
LINE335      CreateLPHDIB(LPHDIB,(uint)(ushort)width,heigth & 0xffff,uVar8,(uint)component,(uint)precision);
LINE336      copy_resolution_to_dib((HDIB)*LPHDIB,&table_fields_name->at_resolution);
LINE337      iVar1 = DIB_colorspace_get((HDIB)*LPHDIB);
LINE338      if ((char)iVar1 == '\x03') {
LINE339        call_IGDIB::AllocatePaletteForDib((HDIB)*LPHDIB);
LINE340      }
LINE341    }
LINE342    AF_error_check();
LINE343    return;
LINE344  }
```

Please notice in this function in LINE140 that if the precision read from a SOF marker is not corresponding to the value of ‘8’, ‘12’ or ‘16’, the function ends here without creating the HDIB object. The need to create an HDIB object is from my understanding to support multiple file formats, so this a common structure that is reused for several purposes, in this specific case it’s used for the allocating `raster_buffer`.

To understand why we land in the `handle_color_channel_with_high_precision` function, we need to get into another function named `jpeg_raster_set`. We can see in LINE623 the function `handle_color_channel_with_high_precision` is called passing `width` and `raster_buffer` as parameters, that in turn were passed as parameters to `jpeg_raster_set` (LINE344). Note that the function `handle_color_channel_with_high_precision` is only called if `precision` is not `8` (LINE568).

```
LINE344   void jpeg_raster_set(jpeg_dec *jpeg_dec,SOF_object *SOF_Object,jpeg_related *jpeg_related,
LINE345                       undefined4 param_4,int width,int height,byte *min_height,short *raster_buffer,
LINE346                       uint size_raster_buffer,int SOF_type,int param_11,int value_to_8,void *param_13,
LINE347                       int param_14,int param_15)
LINE348   {
                [...]
LINE393     if (SOF_type == 2) {
LINE394       _nr_comp = *(byte *)&(SOF_Object->SOF).component;
LINE395     }
LINE396     else {
LINE397       _nr_comp = *(byte *)&SOF_Object->possible_num_component_or_color_channel;
LINE398     }
LINE399     pbVar4 = (byte *)SOF_Object->nr_component_buffer_data;
LINE400     if (SOF_type == 0) {
LINE401   SOF_type_2:
LINE402       _num_component = (uint)_nr_comp;
LINE403       if (_num_component != 0) {
LINE404         puVar15 = (undefined4 *)(pbVar4 + 0x20);
LINE405         uVar8 = _num_component;
LINE406         piVar14 = local_18;
LINE407         while (uVar12 = _num_component, uVar8 != 0) {
LINE408           uVar8 = uVar8 - 1;
LINE409           *piVar14 = 0;
LINE410           piVar14 = piVar14 + 1;
LINE411         }
LINE412         do {
LINE413           *puVar15 = puVar15[-7];
LINE414           puVar15 = puVar15 + 0x14;
LINE415           uVar12 = uVar12 - 1;
LINE416         } while (uVar12 != 0);
LINE417       }
LINE418                       /* --------------------------------
LINE419                           Num of component = 1 e.g grayscale
LINE420                          -------------------------------- */
LINE421       if (_num_component == 1) {
                [...]
LINE480       }
LINE481       else {
LINE482         if (_num_component == 3) {
LINE483   YCbCr:
LINE484                       /* ----------------------------------------
LINE485                          3 components e.g YCbCr
LINE486                          ---------------------------------------- */
                [...]
LINE560               index = 0;
LINE561                       /* max_loop is 16 */
LINE562               if (0 < max_loop) {
LINE563                 do {
LINE564                   _some_buffer = *(short **)(pbVar4 + 0x20);
LINE565                   _some_buffer_2 = *(short **)(pbVar4 + 0x70);
LINE566                   _some_buffer_3 = *(short **)(pbVar4 + 0xc0);
LINE567                         /* if precision is 8 but data is read from 2nd SOF not first one */
LINE568                   if ((SOF_Object->SOF).precision == 8) {
                          [...]
LINE620                   }
LINE621                   else {
LINE622                         /* precision is not 8 */
LINE623                     handle_color_channel_with_high_precision
LINE624                               (&BYTE_10270680,width,_some_buffer,_some_buffer_2,_some_buffer_3,
LINE625                                raster_buffer,(int)pbVar4,param_11);
        [...]
LINE1040    return;
LINE1041  }
```

To invoke the described functions in the order that we described, you need to have a file containing two `SOF` markers, one totally valid to compute size and raster buffer and another `SOF` marker with some invalid `precision` value (like '0xD', for example) to hit LINE140 in `create_HDIB_from_SOF`.

First, the first SOF marker is parsed and a valid HDIB object will be created (together with the `raster_buffer`), based on the SOF marker values (width, precision, number of components). Next, the second SOF marker is parsed, but the HDIB object will fail to be allocated because of the invalid `precision` value in the second SOF marker, hence the `raster_buffer` is not getting updated. Because the second SOF marker has a `precision` higher than `8`, the function `handle_color_channel_with_high_precision` will be called via `jpeg_raster_set`, where the values from the first SOF marker (width, precision, number of components) will be used, however this function is not supposed to be called with a `precision` value of 8. This makes the code run under wrong sizes assumptions for the `raster_buffer` and the do-while loop will run past the end of the buffer, triggering the out-of-bounds write condition in the heap, which can lead to arbitrary code execution.

Crash Information

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


KEY_VALUES_STRING: 1

    Key  : AV.Fault
    Value: Write

    Key  : Analysis.CPU.mSec
    Value: 1280

    Key  : Analysis.DebugAnalysisProvider.CPP
    Value: Create: 8007007e on DESKTOP-4DAOCFH

    Key  : Analysis.DebugData
    Value: CreateObject

    Key  : Analysis.DebugModel
    Value: CreateObject

    Key  : Analysis.Elapsed.mSec
    Value: 10671

    Key  : Analysis.Memory.CommitPeak.Mb
    Value: 71

    Key  : Analysis.System
    Value: CreateObject

    Key  : Timeline.OS.Boot.DeltaSec
    Value: 172399

    Key  : Timeline.Process.Start.DeltaSec
    Value: 149

    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


ADDITIONAL_XML: 1

OS_BUILD_LAYERS: 1

NTGLOBALFLAG:  2100000

APPLICATION_VERIFIER_FLAGS:  0

APPLICATION_VERIFIER_LOADED: 1

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 5bfe5549 (igCore19d!IG_mpi_page_set+0x000c97f9)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 00000001
   Parameter[1]: 0b353000
Attempt to write to address 0b353000

FAULTING_THREAD:  0000216c

PROCESS_NAME:  Fuzzme.exe

WRITE_ADDRESS:  0b353000

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:  0b353000

STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
0019f740 5bfe7c6b     5c120680 000000a4 0b344b50 igCore19d!IG_mpi_page_set+0xc97f9
0019f7f8 5bfe3471     0ad41f60 0019fab4 0ad3f720 igCore19d!IG_mpi_page_set+0xcbf1b
0019f87c 5bfd6438     000001e6 0ad3f720 0ad41f60 igCore19d!IG_mpi_page_set+0xc7721
0019fa94 5bfd005b     0ad41f60 0019fab4 5bfe2fa0 igCore19d!IG_mpi_page_set+0xba6e8
0019fb14 5bfe7025     0000ffd9 5bfe2fa0 0ad3f720 igCore19d!IG_mpi_page_set+0xb430b
0019fb30 5bfe6ef5     0ad3f720 0ad41f60 0000ffda igCore19d!IG_mpi_page_set+0xcb2d5
0019fb54 5bfe4f91     0ad3f720 0ad41f60 0019fb7c igCore19d!IG_mpi_page_set+0xcb1a5
0019fb74 5bfe66ea     0019ffc2 1000001d 0ad3df70 igCore19d!IG_mpi_page_set+0xc9241
0019fbb4 5bef10d9     1000001d 0ad3df70 00000001 igCore19d!IG_mpi_page_set+0xca99a
0019fbec 5bf30557     00000000 0ad3df70 0019fc3c igCore19d!IG_image_savelist_get+0xb29
0019fe68 5bf2feb9     00000000 05325f90 00000001 igCore19d!IG_mpi_page_set+0x14807
0019fe88 5bec5777     00000000 05325f90 00000001 igCore19d!IG_mpi_page_set+0x14169
0019fea8 004021f9     05325f90 0019febc 00000001 igCore19d!IG_load_file+0x47
0019fec0 00402504     05325f90 05323fc0 05273f28 Fuzzme!fuzzme+0x19
0019ff28 0040666d     00000005 0526aee8 05273f28 Fuzzme!fuzzme+0x324
0019ff70 75c8fa29     00201000 75c8fa10 0019ffdc Fuzzme!fuzzme+0x448d
0019ff80 775e75f4     00201000 3dbd1269 00000000 KERNEL32!BaseThreadInitThunk+0x19
0019ffdc 775e75c4     ffffffff 77607356 00000000 ntdll!__RtlUserThreadStart+0x2f
0019ffec 00000000     004066f5 00201000 00000000 ntdll!_RtlUserThreadStart+0x1b


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  igCore19d!IG_mpi_page_set+c97f9
```

```
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:  19.8.0.0

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

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

Timeline

2021-02-09 - Vendor Disclosure
2021-03-02 - Public Release

CREDIT

Discovered by Emmanuel Tacheau of Cisco Talos.