Talos Vulnerability Report

TALOS-2020-1096

Accusoft ImageGear DICOM parse_dicom_meta_info code execution vulnerability

SEPTEMBER 1, 2020

CVE NUMBER

CVE-2020-6152

SUMMARY

A code execution vulnerability exists in the DICOM parse_dicom_meta_info functionality of Accusoft ImageGear 19.7. A specially crafted malformed file can cause an out-of-bounds write. An attacker can trigger this vulnerability by providing a victim with a malicious DICOM file.

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 19.7

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-252 - Unchecked Return Value

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, including many image formats, DICOM, PDF, Microsoft Office and others.

There is a vulnerability in the `parse_dicom_meta_info` function which occurs with a specially crafted DICOM file leading to an out-of-bounds write which can result in remote code execution.

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

```
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=deadfad0 ecx=00000012 edx=010c5000 esi=00020003 edi=deadface
eip=0a5c838f esp=00fbf23c ebp=00fbf39c iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
igMED19d!CPb_MED_init+0x152ef:
0a5c838f c6040700        mov     byte ptr [edi+eax],0       ds:0023:deadface=??
```

The pseudo code responsible to parse the dicom meta information is described below.

```
void __cdecl parse_dicom_meta_info(mys_table_function *file_operations,mys_dicom_to_determine *param_2,undefined8 *param_3)
{
  bool bVar1;
  uint uVar2;
  int iVar3;
  undefined4 offset_in_file;
  int variable;
  int status_get_dicom_tag;
  uint *puVar4;
  uint allocation_size;
  int status;
  mys_table_function *_file_operations;
  int iVar5;
  byte *pbVar6;
  uint _size_from_tag_data;
  int local_150;
  uint size_from_tag;
  uint index_vr_code;
  int local_144;
  int local_140;
  uint dicom_tag_id;
  int local_138;
  uint local_134;
  int local_130;
  int local_12c;
  byte *ptr_buffer_allocated;
  int local_120;
  undefined8 local_11c;
  undefined8 local_114;
  undefined4 local_10c;
  byte preambule_buffer [256];
  uint local_8;

  local_8 = DAT_10050fe0 ^ (uint)&stack0xfffffffc;
  local_12c = 0;
  ptr_buffer_allocated = (byte *)0x0;
  local_140 = 0;
  local_120 = 0;
  bVar1 = false;
  local_138 = 0;
  local_11c = 0;
  local_114 = 0;
  local_10c = 0;
  memset(preambule_buffer,0,0x100);
  (*_igcore19d!set_endian)(file_operations,0);
  (*_igcore19d!set_file_pointer_related)(file_operations,0,0);
  (*_igcore19d!read_data_into_buffer)(file_operations,preambule_buffer,0x80);
  copy_preambule_buffer(param_2,preambule_buffer,0x80);
  (*_igcore19d!set_file_pointer_related)(file_operations,4,1);
  local_134 = 0;
  offset_in_file = (*_igcore19d!possible_get_current_offset)(file_operations);
  variable = FUN_1000ab00(file_operations,1,3,&dicom_tag_id,0);
  (*_igcore19d!set_file_pointer_related)(file_operations,offset_in_file,0);
  local_144 = 0;
  status = 0;
  do {                                                                                                                    [1]
    _file_operations = file_operations;
    iVar5 = variable;
    status_get_dicom_tag =
         get_dicom_tag_info(file_operations,variable,&dicom_tag_id,&index_vr_code,&size_from_tag,                         [4]
                            &local_130);
    iVar3 = local_130;
    uVar2 = dicom_tag_id;
    _size_from_tag_data = size_from_tag;                                                                                  [5]
    if (status_get_dicom_tag == 0) {
      iVar5 = 0x5622;
      _file_operations = (mys_table_function *)0x1225;
      goto display_error;
    }
    if (status != 0) {
      local_120 = local_120 + local_130;
    }
    puVar4 = &DAT_10044008;
    do {
      if (*puVar4 == dicom_tag_id) {
        local_138 = local_138 + 1;
        break;
      }
      puVar4 = puVar4 + 1;
    } while ((int)puVar4 < 0x10044020);
    if ((short)(dicom_tag_id >> 0x10) == 2) {                                                                             [6]
      if ((ushort)dicom_tag_id < (ushort)local_134) {
        iVar5 = 0x5625;
        _file_operations = (mys_table_function *)0x124c;
display_error:
        (*(code *)igcore19d!display_error_message)
                  ("..\\..\\..\\..\\Common\\Components\\MED\\Dicom\\dcmread.c",_file_operations,
                   iVar5);
        goto LAB_1001846a;
      }
      allocation_size = size_from_tag + 2;

      if (0x100 < allocation_size) {
[7]
        status = perform_checking(dicom_tag_id,(short)index_vr_code,size_from_tag);
        if (status == 0) {
          iVar5 = 0x5614;
          _file_operations = (mys_table_function *)0x1264;
        }
        else {
          allocation_size = allocate_mem(param_2,allocation_size,&ptr_buffer_allocated);                                  [8]
          if (allocation_size == 0) goto read_operations;
        }
        goto display_error;
      }
      ptr_buffer_allocated = preambule_buffer;                                                                            [9]
read_operations:
      status = perform_some_read_operations(file_operations,uVar2,_size_from_tag_data,index_vr_code,ptr_buffer_allocated,
&local_134,&local_130); [14]
      if (status == 0) {
        iVar5 = 0x5624;
        _file_operations = (mys_table_function *)0x126e;
        goto display_error;
      }
      ptr_buffer_allocated[_size_from_tag_data] = 0;                                                                      [3]
      [...]
    }
```

```
                [...]
  } while( true );                                                                                                    [2]
}
```

The function `parse_dicom_meta_info` is responsible for processing the dicom file through a do-while loop [1] & [2] to parse the meta info. In our case it's crashing in [3] because `ptr_buffer_allocated` is null. We can control the `_size_from_tag_data` directly from the crafted file. Now let's understand how we can reach this state in detail.

First, a call to the function `get_dicom_tag_info` in [4] fills several important variables : `size_from_tag` , `dicom_tag_id` and `index_vr_code`.
The `size_from_tag` and `dicom_tag_id` are directly read from the file data itself, when `index_vr_code` is corresponding to an index of table of different Value Representation (VR). In our case the interesting Value Representation is 'SQ' for Sequence and associated to an index of '12'. The 'SQ' Value Representation is interesting to us because it doesn't have an enforced length value defined by the DICOM specification.

In [5] `_size_from_tag_data` is assigned the content of `size_from_tag` directly. The `dicom_tag_id` is used to verify the processing of tag IDs corresponding to the 0x0002 group in [6].

According to the `allocation_size`, which is just `size_from_tag + 2`, a test [7] is performed to decide if `ptr_buffer_allocated` will be assigned the result of a memory allocation in [8] or if it will be assigned the local buffer `preambule_buffer` in [9]. Because of this, `ptr_buffer_allocated` can become null only through the `allocate_mem` function.

We need to go deeper into the function `allocate_mem` described below, which is calling in [10] `AF_memm_alloc`, described just after.

```
uint allocate_mem(mys_dicom_to_determine *param_1,size_t size,byte **ptr_ptr_buff)
{
  byte *ptr_buff;
  uint uVar1;

  if ((param_1 == (mys_dicom_to_determine *)0x0) || (param_1->constant_0xdeadadde != -0x21525222)) {
    (*(code *)igcore19d!display_error_message)
              ("..\\..\\..\\..\\Common\\Components\\MED\\Dicom\\DataSet.c",0x580,0x5613,0,0,param_1,
               0);
  }
  else {
    *ptr_ptr_buff = (byte *)0x0;                                                                                      [14]
    if (param_1 == (mys_dicom_to_determine *)0xfffffffb4) {
      *ptr_ptr_buff = (byte *)0x0;
    }
    else {
      ptr_buff = (byte *)(*_igCore19d!AF_memm_alloc)                                                                  [10]
                                (*(undefined4 *)&param_1->kind_of_heap,size,
                                 "..\\..\\..\\..\\Common\\Components\\MED\\Dicom\\DataSet.c",
                                 0x567);
      if (ptr_buff != (byte *)0x0) {                                                                                  [13]
        *ptr_ptr_buff = ptr_buff;
        memset(ptr_buff,0,size);
      }
    }
  }
                    /* WARNING: Could not recover jumptable at 0x1000e963. Too many branches */
                    /* WARNING: Treating indirect jump as call */
  uVar1 = (*_igcore19d!perform_some_memory_operations)();
  return uVar1;
}


void * __thiscall AF_memm_alloc(undefined4 this,uint param_2,size_t size)
{
  int iVar1;
  uint *puVar2;
  void *_Memory;
  uint *puVar3;
  undefined4 *_Dst;
  int iVar4;
  uint uVar5;
  uint *_Dst_00;
    wrapper_EnterCriticalSection(*(LPCRITICAL_SECTION *)(Count_CriticalSectionUse + 0x1684));
  _Memory = MSVCR110.DLL::malloc(size);                                                                               [12]
  if (_Memory == (void *)0x0) {
    wrapper_LeaveCriticalSection(*(LPCRITICAL_SECTION *)(Count_CriticalSectionUse + 0x1684));
    return (void *)0x0;                                                                                               [11]
  }
  [...]
}
```

A null return value can occur in `AF_memm_alloc` [11] only if `MSVCR110.DLL::malloc` is failing [12] . When this happens, the test in [13] will not succeed and `*ptr_ptr_buf` is left untouched, meaning that its value is 0, as it was initialized in [14].
Now we get back into `parse_dicom_meta_info` with a null value for `ptr_buffer_allocated`, the `index_vr_code` will help to avoid an exception in the function `perform_some_read_operations` in [14] before leading to the crash.
Because an attacker controls the value for `size_from_tag` (and thus also `allocation_size`), the `malloc` at [12] can fail by supplying an arbitrary large value. Later, the 0-write at [3] can be controlled using `allocation_size`, because `ptr_buffer_allocated` will be null.

Crash Information

Crash output:

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

KEY_VALUES_STRING: 1

    Key  : AV.Fault
    Value: Write

    Key  : Analysis.CPU.Sec
    Value: 1

    Key  : Analysis.DebugAnalysisProvider.CPP
    Value: Create: 8007007e on DESKTOP-451082P

    Key  : Analysis.DebugData
    Value: CreateObject

    Key  : Analysis.DebugModel
    Value: CreateObject

    Key  : Analysis.Elapsed.Sec
    Value: 5

    Key  : Analysis.Memory.CommitPeak.Mb
    Value: 85

    Key  : Analysis.System
    Value: CreateObject

    Key  : Timeline.OS.Boot.DeltaSec
    Value: 151001

    Key  : Timeline.Process.Start.DeltaSec
    Value: 132


ADDITIONAL_XML: 1

NTGLOBALFLAG:  2100000

APPLICATION_VERIFIER_FLAGS:  0

APPLICATION_VERIFIER_LOADED: 1

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 0a5c838f (igMED19d!CPb_MED_init+0x000152ef)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 00000001
   Parameter[1]: deadface
Attempt to write to address deadface

FAULTING_THREAD:  00001510

PROCESS_NAME:  Fuzzme.exe

WRITE_ADDRESS:  deadface

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:  deadface

STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
00fbf39c 0a5cb44f 00fbf4f4 07b62fa0 00fbf428 igMED19d!CPb_MED_init+0x152ef
00fbf3c4 0a5c6747 00fbf4f4 07b62fa0 10000023 igMED19d!CPb_MED_init+0x183af
00fbf46c 027410d9 00fbf4f4 0776cfb8 00000001 igMED19d!CPb_MED_init+0x136a7
00fbf4a4 02780557 00000000 0776cfb8 00fbf4f4 igCore19d!IG_image_savelist_get+0xb29
00fbf720 0277feb9 00000000 05fb3fe0 00000001 igCore19d!IG_mpi_page_set+0x14807
00fbf740 02715777 00000000 05fb3fe0 00000001 igCore19d!IG_mpi_page_set+0x14169
00fbf760 00961372 05fb3fe0 00fbf77c 05facf80 igCore19d!IG_load_file+0x47
00fbf788 00961778 05fb3fe0 00fbf7f0 00000021 Fuzzme!fuzzme+0x162
00fbf818 00961f7d 00000005 05facf80 02f44f48 Fuzzme!fuzzme+0x568
00fbf860 7744e2f9 010c4000 7744e2e0 00fbf8cc Fuzzme!fuzzme+0xd6d
00fbf870 77c727c7 010c4000 c1489ff1 00000000 KERNEL32!BaseThreadInitThunk+0x19
00fbf8cc 77c7279b ffffffff 77cb2d62 00000000 ntdll!__RtlUserThreadStart+0x2b
00fbf8dc 00000000 00962005 010c4000 00000000 ntdll!_RtlUserThreadStart+0x1b


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  igMED19d!CPb_MED_init+152ef

MODULE_NAME: igMED19d

IMAGE_NAME:  igMED19d.dll

FAILURE_BUCKET_ID:  NULL_POINTER_WRITE_AVRF_c0000005_igMED19d.dll!CPb_MED_init

OS_VERSION:  10.0.17763.1

BUILDLAB_STR:  rs5_release

OSPLATFORM_TYPE:  x86

OSNAME:  Windows 10

FAILURE_ID_HASH:  {45a639e4-203c-d375-5c67-d4bd7ae4ad96}

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

TIMELINE

2020-06-23 - Vendor Disclosure
2020-09-01 - Public Release

CREDIT

Discovered by Emmanuel Tacheau of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2020-1036

NEXT REPORT TALOS-2020-1095