Talos Vulnerability Report

TALOS-2021-1275

# Accusoft ImageGear DICOM parse_dicom_meta_info integer overflow vulnerability

JUNE 1, 2021

### CVE NUMBER

CVE-2021-21807

### Summary

An integer overflow vulnerability exists in the DICOM parse_dicom_meta_info functionality of Accusoft ImageGear 19.9. A specially crafted malformed file can lead to a stack-based buffer overflow. 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-190 - Integer Overflow or Wraparound

### 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 `parse_dicom_meta_info` function which occurs with a specially crafted DICOM file, leading to a stack-based buffer overflow which can result in remote code execution.

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

```
(b7c0.b52c): Security check failure or stack buffer overrun - code c0000409 (!!! second chance !!!)
Subcode: 0x2 FAST_FAIL_STACK_COOKIE_CHECK_FAILURE
eax=00000001 ebx=0c3ebfb8 ecx=00000002 edx=000001e9 esi=0019fa4c edi=0f0d8fa0
eip=79fb5b82 esp=0019f55c ebp=0019f880 iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000202
igMED19d!CPb_MED_init+0x22ae2:
79fb5b82 cd29            int     29h
```

Stack inspection show us the stack buffer overflow as follow:

```
0:000> kb
 # ChildEBP RetAddr      Args to Child
WARNING: Stack unwind information not available. Following frames may be wrong.
00 0019f880 79fa85d7     0019fa64 00000000 00000000 igMED19d!CPb_MED_init+0x22ae2
01 0019f9d8 32373334     31383533 31323533 34303030 igMED19d!CPb_MED_init+0x15537
02 0019f9dc 31383533     31323533 34303030 00080031 0x32373334
03 0019f9e0 31323533     34303030 00080031 49550016 0x31383533
04 0019f9e4 34303030     00080031 49550016 2e31001c 0x31323533
05 0019f9e8 00080031     49550016 2e31001c 34382e32 0x34303030
06 0019f9ec 49550016     2e31001c 34382e32 30312e30 0x80031
07 0019f9f0 2e31001c     34382e32 30312e30 2e383030 0x49550016
08 0019f9f4 34382e32     30312e30 2e383030 2e312e35 0x2e31001c
09 0019f9f8 30312e30     2e383030 2e312e35 2e312e34 0x34382e32
0a 0019f9fc 2e383030     2e312e35 2e312e34 2e332e31 0x30312e30
```

In our case the stack overwrite is happening in a do-while loop (LINE204 to LINE213) in function `perform_some_read_operations`. This is the pseudo-code of said function:

```
LINE1    BOOL perform_some_read_operations
LINE2                   (mys_table_function *table_function,int tag_dicom_id,uint size,uint index_vr_table,
LINE3                   void *ptr_buffer_allocated,uint *size_divided_by_4,int *param_7)
LINE4    {
            [...]
LINE25     _ptr_buffer_allocated = ptr_buffer_allocated;
LINE26     read_size = 0;
LINE27     *param_7 = 0;
LINE28     *size_divided_by_4 = 0;
LINE29     if (0x1a < (index_vr_table & 0xffff)) {
LINE30       if (tag_dicom_id != -0x12000) {
LINE31         if ((size != 0xffffffff) &&
LINE32            (read_size = IO_read(table_function,(byte *)ptr_buffer_allocated,size), read_size != size))
LINE33         {
LINE34           return 0;
LINE35         }
LINE36         *piVar2 = *piVar2 + read_size;
LINE37       }
LINE38       *puVar7 = 1;
LINE39     switchD_1001a8c4_caseD_12:
LINE40       return 1;
LINE41     }
LINE42     uVar10 = (undefined2)index_vr_table;
LINE43     switch(index_vr_table & 0xffff) {
LINE44     default:
LINE45       read_size = IO_read(table_function,(byte *)ptr_buffer_allocated,size);
LINE46       if (read_size != size) {
LINE47         return 0;
LINE48       }
LINE49       *piVar2 = *piVar2 + read_size;
LINE50       uVar5 = FUN_10024a80((int)ptr_buffer_allocated,size,uVar10,&index_vr_table);
LINE51       *puVar7 = uVar5;
LINE52       return 1;
LINE53     case 2:
LINE54       local_c = size >> 2;
LINE55       *size_divided_by_4 = local_c;
LINE56       uVar5 = 0;
LINE57       iVar8 = 0;
LINE58       local_8 = 0;
LINE59       if (local_c != 0) {
LINE60         do {
LINE61           dVar3 = read_short(table_function,&param_7);
LINE62           dVar4 = read_short(table_function,&size_divided_by_4);
LINE63           iVar8 = dVar3 + dVar4;
LINE64           if (iVar8 != 4) {
LINE65             return 0;
LINE66           }
LINE67           local_8 = local_8 + 4;
LINE68           *(uint *)((int)ptr_buffer_allocated + uVar5 * 4) =
LINE69                (uint)size_divided_by_4 & 0xffff | (int)param_7 << 0x10;
LINE70           uVar5 = uVar5 + 1;
LINE71         } while (uVar5 < local_c);
LINE72       }
LINE73       if ((size & 3) != 0) {
LINE74         IO_seek(table_function,size - local_8,1);
LINE75       }
LINE76       *piVar2 = *piVar2 + iVar8;
LINE77       return 1;
LINE78     case 7:
LINE79       puVar7 = (uint *)(size >> 2);
LINE80       *size_divided_by_4 = (uint)puVar7;
LINE81       _index = NULL;
LINE82       pvVar9 = NULL;
LINE83       param_7 = NULL;
LINE84       size_divided_by_4 = puVar7;
LINE85       if (puVar7 != NULL) {
LINE86         do {
LINE87           pvVar9 = (void *)read_long(table_function,_ptr_buffer_allocated);
LINE88           if (pvVar9 != (void *)0x4) {
LINE89             return 0;
LINE90           }
LINE91           param_7 = param_7 + 1;
LINE92           _index = (uint *)((int)_index + 1);
LINE93           _ptr_buffer_allocated = (void *)((int)_ptr_buffer_allocated + 4);
LINE94           ptr_buffer_allocated = pvVar9;
LINE95         } while (_index < size_divided_by_4);
LINE96       }
LINE97       break;
LINE98     case 8:
LINE99       puVar7 = (uint *)(size >> 3);
LINE100      *size_divided_by_4 = (uint)puVar7;
LINE101      _index = NULL;
LINE102      pvVar9 = NULL;
LINE103      param_7 = NULL;
LINE104      buffer = (byte *)ptr_buffer_allocated;
LINE105      size_divided_by_4 = puVar7;
LINE106      if (puVar7 != NULL) {
LINE107        do {
LINE108          pvVar9 = (void *)IO_read(table_function,buffer,8);
LINE109          if (pvVar9 != (void *)0x8) {
LINE110            return 0;
LINE111          }
LINE112          param_7 = (int *)((int)param_7 * 2);
LINE113          _index = (uint *)((int)_index + 1);
LINE114          buffer = buffer + 8;
LINE115          ptr_buffer_allocated = pvVar9;
LINE116        } while (_index < size_divided_by_4);
LINE117      }
LINE118      if ((size & 7) != 0) {
LINE119        IO_seek(table_function,size - (int)param_7,1);
LINE120        *piVar2 = *piVar2 + (int)pvVar9;
LINE121        return 1;
LINE122      }
LINE123      goto LAB_1001a9f0;
LINE124    case 0xc:
LINE125    case 0xd:
LINE126    case 0x19:
LINE127    case 0x1a:
LINE128      if ((size != 0xffffffff) &&
LINE129         (read_size = IO_read(table_function,(byte *)ptr_buffer_allocated,size), read_size != size)) {
LINE130        return 0;
LINE131      }
LINE132      *piVar2 = *piVar2 + read_size;
LINE133      uVar5 = FUN_10024a80((int)ptr_buffer_allocated,size,uVar10,&index_vr_table);
LINE134      *puVar7 = uVar5;
LINE135      return 1;
LINE136    case 0xe:
```

```
LINE137    case 0x13:
LINE138    case 0x18:
LINE139      uVar5 = size >> 1;
LINE140      *size_divided_by_4 = uVar5;
LINE141      ptr_buffer_allocated =
LINE142           (void *)IO_read(table_function,(byte *)ptr_buffer_allocated,size & 0xfffffffe);
LINE143      (*_DAT_10051634)(table_function,&local_10);
LINE144      pvVar9 = ptr_buffer_allocated;
LINE145      if ((local_10 == 1) && (uVar6 = 0, uVar5 != 0)) {
LINE146        do {
LINE147          uVar1 = *(ushort *)((int)_ptr_buffer_allocated + uVar6 * 2);
LINE148          *(ushort *)((int)_ptr_buffer_allocated + uVar6 * 2) = uVar1 << 8 | uVar1 >> 8;
LINE149          uVar6 = uVar6 + 1;
LINE150        } while (uVar6 < uVar5);
LINE151      }
LINE152      if ((size & 1) != 0) {
LINE153        IO_seek(table_function,size - (int)ptr_buffer_allocated,1);
LINE154        *piVar2 = *piVar2 + (int)pvVar9;
LINE155        return 1;
LINE156      }
LINE157      goto LAB_1001a9f0;
LINE158    case 0xf:
LINE159      read_size = IO_read(table_function,(byte *)ptr_buffer_allocated,size);
LINE160      goto joined_r0x1001aaf9;
LINE161    case 0x10:
LINE162    case 0x14:
LINE163    case 0x15:
LINE164    case 0x16:
LINE165      read_size = IO_read(table_function,(byte *)ptr_buffer_allocated,size);
LINE166  joined_r0x1001aaf9:
LINE167      if (read_size != size) {
LINE168        return 0;
LINE169      }
LINE170      *piVar2 = *piVar2 + read_size;
LINE171      uVar5 = FUN_10024a80((int)ptr_buffer_allocated,size,uVar10,&index_vr_table);
LINE172      *puVar7 = uVar5;
LINE173      return 1;
LINE174    case 0x11:
LINE175      puVar7 = (uint *)(size >> 2);
LINE176      *size_divided_by_4 = (uint)puVar7;
LINE177      _index = NULL;
LINE178      pvVar9 = NULL;
LINE179      param_7 = NULL;
LINE180      size_divided_by_4 = puVar7;
LINE181      if (puVar7 != NULL) {
LINE182        do {
LINE183          pvVar9 = (void *)read_long(table_function,_ptr_buffer_allocated);
LINE184          if (pvVar9 != (void *)0x4) {
LINE185            return 0;
LINE186          }
LINE187          param_7 = param_7 + 1;
LINE188          _index = (uint *)((int)_index + 1);
LINE189          _ptr_buffer_allocated = (void *)((int)_ptr_buffer_allocated + 4);
LINE190          ptr_buffer_allocated = pvVar9;
LINE191        } while (_index < size_divided_by_4);
LINE192      }
LINE193      break;
LINE194    case 0x12:
LINE195      goto switchD_1001a8c4_caseD_12;
LINE196    case 0x17:
LINE197      puVar7 = (uint *)(size >> 2);
LINE198      *size_divided_by_4 = (uint)puVar7;
LINE199      _index = NULL;
LINE200      pvVar9 = NULL;
LINE201      param_7 = NULL;
LINE202      size_divided_by_4 = puVar7;
LINE203      if (puVar7 != NULL) {
LINE204        do {
LINE205          pvVar9 = (void *)read_long(table_function,_ptr_buffer_allocated);
LINE206          if (pvVar9 != (void *)0x4) {
LINE207            return 0;
LINE208          }
LINE209          param_7 = param_7 + 1;
LINE210          _index = (uint *)((int)_index + 1);
LINE211          _ptr_buffer_allocated = (void *)((int)_ptr_buffer_allocated + 4);
LINE212          ptr_buffer_allocated = pvVar9;
LINE213        } while (_index < size_divided_by_4);
LINE214      }
LINE215    }
LINE216    if ((size & 3) != 0) {
LINE217      IO_seek(table_function,size - (int)param_7,1);
LINE218    }
LINE219  LAB_1001a9f0:
LINE220    *piVar2 = *piVar2 + (int)pvVar9;
LINE221    return 1;
LINE222  }
```

The variable corresponding to our stack buffer is represented by the variable named `_ptr_buffer_allocated`, and it is written into through the call to `read_long` function in LINE205. The do-while loop is controlled by the `size_divided_by_4`. The `_ptr_buffer_allocated` is corresponding to the argument `ptr_buffer_allocated` passed to this function (see LINE25) and `size_divided_by_4` is the value of `size` divided by 4 (see LINE197 and LINE202).
We can see during the loop the address of the buffer is incremented by 4 in LINE211.

Now the function `perform_some_read_operations` above is called by the function `parse_dicom_meta_info` in LINE363. The two interesting parameters are represented by the `_size` LINE362 and the variable `copy_stack_buffer` corresponding respectively to our previously seen variables `size` and `ptr_buffer_allocated` argument of `perform_some_read_operations`.

```
LINE223  dword parse_dicom_meta_info
LINE224                 (mys_table_function *table_function,dicom_data_set *dicom_data_set,
LINE225                 undefined8 *param_3)
LINE226  {
             [...]
LINE256    int stack_buffer [64];
LINE257    uint local_8;
LINE258    uint _size_copy;
LINE259    uint _tag_dicom_id;
LINE260
LINE261    local_8 = DAT_10050fe0 ^ (uint)&stack0xfffffffc;
LINE262                     /* size_t _Size for memset */
LINE263    bVar1 = false;
LINE264    iVar6 = 0;
LINE265                     /* int _Val for memset */
LINE266                     /* void * _Dst for memset */
LINE267    local_12c = 0;
LINE268    copy_stack_buffer = NULL;
LINE269    local_140 = 0;
LINE270    local_120 = 0;
LINE271    local_138 = 0;
LINE272    local_11c = 0;
LINE273    local_114 = 0;
LINE274    local_10c = 0;
LINE275    memset(stack_buffer,0,256);
LINE276    set_endian(table_function,0);
LINE277    IO_seek(table_function,0,0);
LINE278    IO_read(table_function,(byte *)stack_buffer,128);
LINE279    copy_preambule_buffer_into_dicom_data(dicom_data_set,stack_buffer,128);
LINE280    IO_seek(table_function,4,1);
LINE281    local_130 = 0;
LINE282    _current_offset = get_current_offset(table_function);
LINE283    constant_1 = FUN_1000ab00(table_function,1,3,&tag_dicom_id,0);
LINE284    IO_seek(table_function,_current_offset,0);
LINE285    local_148 = 0;
LINE286    do {
LINE287      pmVar7 = table_function;
LINE288      iVar9 = constant_1;
LINE289      iVar2 = get_dicom_tag_info(table_function,constant_1,&tag_dicom_id,&index_into_vr_code_records,
LINE290                                 &_size,&local_128);
LINE291      _current_offset = local_128;
LINE292      _tag_dicom_id = tag_dicom_id;
LINE293      if (iVar2 == 0) {
LINE294        AF_err_record_set("..\\..\\..\\..\\Common\\Components\\MED\\Dicom\\dcmread.c",0x122a,0x5622,0,
LINE295                          _size,local_128,NULL);
LINE296        bVar1 = true;
LINE297  LAB_100183b6:
LINE298        if (copy_stack_buffer == NULL) {
LINE299          AF_err_record_set("..\\..\\..\\..\\Common\\Components\\MED\\Dicom\\dcmread.c",0x1272,0x5623,
LINE300                            0,0,0,NULL);
LINE301          bVar1 = true;
LINE302        }
LINE303        else {
LINE304          if (!bVar1) goto LAB_100183ee;
LINE305        }
LINE306      }
LINE307      else {
LINE308        if (local_12c != 0) {
LINE309          local_120 = local_120 + local_128;
LINE310        }
LINE311        puVar3 = &DAT_10044008;
LINE312        do {
LINE313          if (*puVar3 == tag_dicom_id) {
LINE314            local_138 = local_138 + 1;
LINE315            break;
LINE316          }
LINE317          puVar3 = puVar3 + 1;
LINE318        } while ((int)puVar3 < 0x10044020);
LINE319        if ((short)(tag_dicom_id >> 0x10) != 2) {
LINE320          if ((local_12c != 0) && (local_138 != 6)) {
LINE321            AF_err_record_set("..\\..\\..\\..\\Common\\Components\\MED\\Dicom\\dcmread.c",0x1245,
LINE322                              0x55f6,0,tag_dicom_id >> 0x10,tag_dicom_id & 0xffff,NULL);
LINE323          }
LINE324          IO_seek(table_function,-_current_offset,1);
LINE325  LAB_100183ab:
LINE326          bVar1 = true;
LINE327          iVar6 = local_120;
LINE328          goto LAB_100183b6;
LINE329        }
LINE330        if ((ushort)tag_dicom_id < (ushort)local_130) {
LINE331          _tag_dicom = tag_dicom_id & 0xffff;
LINE332          uVar4 = local_130 & 0xffff;
LINE333          _current_offset = 0;
LINE334          iVar9 = 0x5625;
LINE335          pmVar7 = (mys_table_function *)0x1251;
LINE336  error:
LINE337          AF_err_record_set("..\\..\\..\\..\\Common\\Components\\MED\\Dicom\\dcmread.c",pmVar7,iVar9,
LINE338                            _current_offset,uVar4,_tag_dicom,NULL);
LINE339          goto LAB_100183ab;
LINE340        }
LINE341        uVar4 = _size + 2;
LINE342        if (0x100 < uVar4) {
LINE343          iVar6 = perform_checking(tag_dicom_id,(short)index_into_vr_code_records,_size);
LINE344          if (iVar6 == 0) {
LINE345            _tag_dicom = 0;
LINE346            _current_offset = 0;
LINE347            iVar9 = 0x5614;
LINE348            pmVar7 = (mys_table_function *)0x1269;
LINE349          }
LINE350          else {
LINE351            _current_offset = 0x10018375;
LINE352            dVar5 = allocate_mem_zero_buffer(dicom_data_set,uVar4,&copy_stack_buffer);
LINE353            iVar6 = local_120;
LINE354            if (dVar5 == 0) goto LAB_100183b6;
LINE355            _tag_dicom = 0x5623;
LINE356            uVar4 = 0x1262;
LINE357          }
LINE358          goto error;
LINE359        }
LINE360        copy_stack_buffer = stack_buffer;
LINE361  LAB_100183ee:
LINE362        _size_copy = _size;
LINE363        uVar4 = perform_some_read_operations
LINE364                          (table_function,_tag_dicom_id,_size,index_into_vr_code_records,
LINE365                           copy_stack_buffer,&local_130,&local_128);
LINE366        if (uVar4 == 0) {
LINE367          iVar6 = 0x5624;
```

```
LINE368          uVar8 = 0x127b;
LINE369          _tag_dicom = uVar4;
LINE370  LAB_1001842d:
LINE371          AF_err_record_set("..\\..\\..\\..\\Common\\Components\\MED\\Dicom\\dcmread.c",uVar8,iVar6,
LINE372                            uVar4,_tag_dicom,_size_copy,(char *)uVar4);
LINE373          bVar1 = true;
LINE374          iVar6 = local_120;
LINE375        }
LINE376        else {
LINE377          *(undefined *)(_size_copy + (int)copy_stack_buffer) = 0;
LINE378          if (local_12c != 0) {
LINE379            local_120 = local_120 + local_128;
LINE380          }
LINE381          if (_tag_dicom_id == 0x20000) {
LINE382            local_140 = *copy_stack_buffer;
LINE383            local_120 = 0;
LINE384            local_12c = 1;
LINE385            iVar6 = 0;
LINE386          }
LINE387          else {
LINE388            piVar10 = copy_stack_buffer;
LINE389            iVar6 = FUN_100131c0(_tag_dicom_id);
LINE390            iVar6 = FUN_1000e5a0(dicom_data_set,iVar6,piVar10,_size_copy);
LINE391            if (iVar6 != 0) {
LINE392              uVar4 = 0;
LINE393              _size_copy = 0;
LINE394              iVar6 = 0x5615;
LINE395              uVar8 = 0x1297;
LINE396              _tag_dicom = _tag_dicom_id;
LINE397              goto LAB_1001842d;
LINE398            }
LINE399            iVar6 = local_120;
LINE400            if (_tag_dicom_id == 0x20010) {
LINE401              FUN_100248a0((char *)copy_stack_buffer,&local_14c,0,NULL);
LINE402              FUN_10024a00(local_14c,NULL,NULL,(undefined4 *)&local_11c);
LINE403              iVar6 = local_120;
LINE404            }
LINE405          }
LINE406        }
LINE407      }
LINE408      local_130 = _tag_dicom_id;
LINE409      if ((copy_stack_buffer != stack_buffer) && (copy_stack_buffer != NULL)) {
LINE410        FUN_1000e980((int *)dicom_data_set,copy_stack_buffer);
LINE411        copy_stack_buffer = NULL;
LINE412      }
LINE413      if ((((iVar6 == local_140) && (local_12c != 0)) || (bVar1)) ||
LINE414         (local_148 = local_148 + 1, 9 < (ushort)local_148)) {
LINE415        local_114 = CONCAT44(0x270d,(int)local_114);
LINE416        local_10c = 1;
LINE417        if ((int)local_114 == 0) {
LINE418          AF_err_record_set("..\\..\\..\\..\\Common\\Components\\MED\\Dicom\\dcmread.c",0x12c6,0x55f6,
LINE419                            0,2,0x10,NULL);
LINE420        }
LINE421        if (param_3 != NULL) {
LINE422          *param_3 = local_11c;
LINE423          param_3[1] = local_114;
LINE424          *(undefined4 *)(param_3 + 2) = local_10c;
LINE425        }
LINE426        AF_error_check();
LINE427        dVar5 = raise_security_failure(local_8 ^ (uint)&stack0xfffffffc,extraout_DL,(char)param_3);
LINE428        return dVar5;
LINE429      }
LINE430    } while( true );
LINE431  }
```

The `_size` value is directly read from the file through the call to the function `get_dicom_tag_info` in LINE289 and an integer overflow is happening in LINE341 if `size` is set to the value 0xFFFFFFFF. The check at line LINE342 tests if the size of `stack_buffer` is bigger or equal than `_size + 2`, and if it is, a new larger buffer is allocated. Because of the integer overflow, the check at LINE342 succeeds and no buffer is allocated. This in turn leads to a stack buffer overflow which could lead to code execution.

Crash Information

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


KEY_VALUES_STRING: 1

    Key  : Analysis.CPU.mSec
    Value: 2281

    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: 6706

    Key  : Analysis.Memory.CommitPeak.Mb
    Value: 184

    Key  : Analysis.System
    Value: CreateObject

    Key  : Timeline.OS.Boot.DeltaSec
    Value: 72299

    Key  : Timeline.Process.Start.DeltaSec
    Value: 87

    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: 79fb5b82 (igMED19d!CPb_MED_init+0x00022ae2)
   ExceptionCode: c0000409 (Security check failure or stack buffer overrun)
  ExceptionFlags: 00000001
NumberParameters: 1
   Parameter[0]: 00000002
Subcode: 0x2 FAST_FAIL_STACK_COOKIE_CHECK_FAILURE

FAULTING_THREAD:  0000b52c

PROCESS_NAME:  Fuzzme.exe

WATSON_BKT_EVENT:  BEX

ERROR_CODE: (NTSTATUS) 0xc0000409 - The system detected an overrun of a stack-based buffer in this application. This overrun could
potentially allow a malicious user to gain control of this application.

EXCEPTION_CODE_STR:  c0000409

EXCEPTION_PARAMETER1:  00000002

STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
0019f880 79fa85d7     0019fa64 00000000 00000000 igMED19d!CPb_MED_init+0x22ae2
0019f9d8 32373334     31383533 31323533 34303030 igMED19d!CPb_MED_init+0x15537
0019f9dc 31383533     31323533 34303030 00080031 0x32373334
0019f9e0 31323533     34303030 00080031 49550016 0x31383533
0019f9e4 34303030     00080031 49550016 2e31001c 0x31323533
0019f9e8 00080031     49550016 2e31001c 34382e32 0x34303030
0019f9ec 49550016     2e31001c 34382e32 30312e30 0x80031
0019f9f0 2e31001c     34382e32 30312e30 2e383030 0x49550016
0019f9f4 34382e32     30312e30 2e383030 2e312e35 0x2e31001c
0019f9f8 30312e30     2e383030 2e312e35 2e312e34 0x34382e32
0019f9fc 2e383030     2e312e35 2e312e34 2e332e31 0x30312e30
0019fa00 2e312e35     2e312e34 2e332e31 00080031 0x2e383030
0019fa04 2e312e34     2e332e31 00080031 49550018 0x2e312e35
0019fa08 2e332e31     00080031 49550018 2e31002a 0x2e312e34
0019fa0c 00080031     49550018 2e31002a 34382e32 0x2e332e31
0019fa10 49550018     2e31002a 34382e32 31312e30 0x80031
0019fa14 2e31002a     34382e32 31312e30 37353833 0x49550018
0019fa18 34382e32     31312e30 37353833 3236312e 0x2e31002a
0019fa1c 31312e30     37353833 3236312e 36312e36 0x34382e32
0019fa20 37353833     3236312e 36312e36 35333630 0x31312e30
0019fa24 3236312e     36312e36 35333630 3237312e 0x37353833
0019fa28 36312e36     35333630 3237312e 35312e37 0x3236312e
0019fa2c 35333630     3237312e 35312e37 35313831 0x36312e36
0019fa30 3237312e     35312e37 35313831 342e312e 0x35333630
0019fa34 35312e37     35313831 342e312e 00200008 0x3237312e
0019fa38 35313831     342e312e 00200008 00084144 0x35312e37
0019fa3c 342e312e     00200008 00084144 30303032 0x35313831
0019fa40 00200008     00084144 30303032 33323230 0x342e312e
0019fa44 00084144     30303032 33323230 00230008 0x200008
0019fa48 30303032     33323230 00230008 00084144 0x84144
```

```
0019fa4c 33323230     00230008 00084144 30303032 0x30303032
0019fa50 00230008     00084144 30303032 33323230 0x33323230
0019fa54 00084144     30303032 33323230 00000000 0x230008
0019fa58 30303032     33323230 00000000 00000000 0x84144
0019fa5c 33323230     00000000 00000000 00000000 0x30303032
0019fa60 00000000     00000000 00000000 0000270d 0x33323230


STACK_COMMAND:  ~0s ; .cxr ; kb

SYMBOL_NAME:  igMED19d!CPb_MED_init+22ae2

MODULE_NAME: igMED19d

IMAGE_NAME:  igMED19d.dll

FAILURE_BUCKET_ID:  FAIL_FAST_STACK_BUFFER_OVERRUN_STACK_COOKIE_CHECK_FAILURE_MISSING_GSFRAME_AVRF_c0000409_igMED19d.dll!CPb_MED_init

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:  {dc561a0a-1f81-d3b1-565f-3f9fcd5b0c15}

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

## Timeline

2021-03-22 - Vendor Disclosure

2021-06-01 - Public Release

## CREDIT

Discovered by Emmanuel Tacheau of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2021-1286

NEXT REPORT TALOS-2021-1276