Talos Vulnerability Report

TALOS-2022-1465

Accusoft ImageGear IGXMPXMLParser::parseDelimiter stack-based buffer overflow vulnerability

MAY 2, 2022

CVE NUMBER

CVE-2022-23400

Summary

A stack-based buffer overflow vulnerability exists in the IGXMPXMLParser::parseDelimiter functionality of Accusoft ImageGear 19.10. A specially-crafted PSD file can overflow a stack buffer, which could either lead to denial of service or, depending on the application, to an information leak. An attacker can provide a malicious file to trigger this vulnerability.

Tested Versions

Accusoft ImageGear 19.10

Product URLs

ImageGear - https://www.accusoft.com/products/imagegear-collection/

CVSSv3 Score

7.1 - CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:N/A:H

CWE

CWE-193 - Off-by-one Error

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.

Trying to load a malformed PSD file, we end up with the following situation:

```
(758.8fc): C++ EH exception - code e06d7363 (first chance)
ModLoad: 75590000 75610000   C:\Windows\SysWOW64\uxtheme.dll

STATUS_STACK_BUFFER_OVERRUN encountered
(758.8fc): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=711732c8 ecx=76ae01c0 edx=0018ea2d esi=00000000 edi=000039ac
eip=76adffa1 esp=0018ec74 ebp=0018ecf0 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
kernel32!UnhandledExceptionFilter+0x5f:
76adffa1 cc              int     3
```

This kind of error `STATUS_STACK_BUFFER_OVERRUN` indicates an abnormal program termination. Looking at the call stack may indicate the culprit.

```
0:000> kb
ChildEBP RetAddr  Args to Child
0018ecf0 71ace2d9 711732c8 0018ed0c 7110c698 kernel32!UnhandledExceptionFilter+0x5f
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for
C:\Program Files (x86)\Talos Vrt Team\ImageGearFuzzing\bin\igCore19d.dll -
0018ecfc 7110c698 711732c8 00000001 0018f03c MSVCR110!__crtUnhandledException+0x14
WARNING: Stack unwind information not available. Following frames may be wrong.
0018ed0c 7110c7af 711732c8 029c02d0 029c95e0 igCore19d!IG_GUI_page_title_set+0x3c5e8
0018f03c 71089a04 029c76bd 029c3f8a 4848482f igCore19d!IG_GUI_page_title_set+0x3c6ff
0018f150 7108950c 029c0150 0018f1e8 029c0348 igCore19d!IG_mpi_page_set+0x12d9d4
0018f16c 7108966f 029c765d 000039ac 00000001 igCore19d!IG_mpi_page_set+0x12d4dc
0018f1b8 7109003d 0018fc54 029c0150 00000000 igCore19d!IG_mpi_page_set+0x12d63f
0018f670 7104cd0b 0018fc54 1000001e 029c0348 igCore19d!IG_mpi_page_set+0x13400d
0018f738 7104c242 0018fc54 1000001e 029c0098 igCore19d!IG_mpi_page_set+0xf0cdb
0018f774 7104bcba 0018fc54 0018f79c 0018f7c4 igCore19d!IG_mpi_page_set+0xf0212
0018fbcc 70f313d9 0018fc54 029c0060 00000001 igCore19d!IG_mpi_page_set+0xefc8a
0018fc04 70f708d7 00000000 029c0060 0018fc54 igCore19d!IG_image_savelist_get+0xb29
0018fe80 70f70239 00000000 00308230 00000001 igCore19d!IG_mpi_page_set+0x148a7
0018fea0 70f05757 00000000 00308230 00000001 igCore19d!IG_mpi_page_set+0x14209
*** WARNING: Unable to verify checksum for Fuzzme.exe
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for
Fuzzme.exe -
0018fec0 00402219 00308230 0018fed4 00000001 igCore19d!IG_load_file+0x47
0018fed8 00402524 00308230 0018ff10 003079a8 Fuzzme!fuzzme+0x19
0018ff40 0040668d 00000005 003066a8 003079a8 Fuzzme!fuzzme+0x324
0018ff88 76aa33ca 7efde000 0018ffd4 77d19ed2 Fuzzme!fuzzme+0x448d
0018ff94 77d19ed2 7efde000 7741e483 00000000 kernel32!BaseThreadInitThunk+0xe
0018ffd4 77d19ea5 00406715 7efde000 00000000 ntdll!__RtlUserThreadStart+0x70
0018ffec 00000000 00406715 7efde000 00000000 ntdll!_RtlUserThreadStart+0x1b
```

Investigating the callback stack leads us to the function pointed by `igCore19d!IG_mpi_page_set+0x12d9d4`, which points to the end to the following pseudo code function at LINE165 pointed by the call to `security_check_cookie` :

```
LINE1   void __thiscall IGXMPXMLParser::FUN_74239720(IGXMPXMLParser *this)
LINE2   {
LINE3     char cVar1;
LINE4     void *pvVar2;
LINE5     code *pcVar3;
LINE6     bool bVar4;
LINE7     int iVar5;
LINE8     int iVar6;
LINE9     char *pcVar7;
LINE10    char *pcVar8;
LINE11    char *pcVar9;
LINE12    char *local_10c;
LINE13    char buffer_ovw [256];
LINE14    uint stack_canary;
LINE15
LINE16    stack_canary = DAT_7435cea8 ^ (uint)&stack0xfffffffc;
LINE17    pcVar7 = this->field2_0x8;
LINE18    pcVar9 = pcVar7 + this->field3_0xc;
LINE19    bVar4 = false;
LINE20    this->field10_0x1c = pcVar7;
LINE21    pcVar8 = pcVar7;
LINE22    while (pcVar8 < pcVar9) {
LINE23      cVar1 = *(char *)this->field10_0x1c;
[...]
LINE48      else if (cVar1 == '<') {
[...]
LINE91                      /* ------------------------------------------------------
----------------------
LINE92                         */
LINE93        iVar5 = parseDelimiter(this,(char (*) [256])buffer_ovw);
LINE94                      /* ------------------------------------------------------
----------------------
LINE95                         */
[...]
LINE161     this->field10_0x1c = this->field10_0x1c + 1;
LINE162     pcVar8 = (char *)this->field10_0x1c;
LINE163   }
LINE165   security_check_cookie(stack_canary ^ (uint)&stack0xfffffffc);
LINE166   return;
LINE167 }
```

So this indicates to us that somehow the `stack_canary` declared at LINE14, which is initialized LINE16, has been overwritten along the flow of this function. After investigation, it appears the `stack_canary` is overwritten during the call to the `parseDelimiter` at LINE93, which takes as a parameter the variable `buffer_ovw` declared at LINE13 just before the `stack_canary`.

Below is the `parseDelimiter` pseudo-code:

```
LINE168 void __thiscall IGXMPXMLParser::parseDelimiter(IGXMPXMLParser *this,char
(*buffer_ovw) [256])
LINE169 {
LINE170   bool bVar1;
LINE171   int *piVar2;
LINE172   char (*target_buffer) [256];
LINE173   int index;
LINE174   bool bVar3;
LINE175   int local_410;
LINE176   int local_40c;
LINE177   char string_buffer [1024];
LINE178   uint stack_cookie;
LINE179
LINE180   stack_cookie = DAT_7435cea8 ^ (uint)&stack0xfffffffc;
LINE181   index = 0;
LINE182   bVar1 = false;
LINE183   bVar3 = *(char *)this->field10_0x1c == '/';
LINE184   if (bVar3) {
LINE185     (*buffer_ovw)[0] = '/';
LINE186     this->field10_0x1c = this->field10_0x1c + 1;
LINE187   }
LINE188   target_buffer = (char (*) [256])(*buffer_ovw + bVar3);
LINE189   while (index < 256) {
LINE190     switch(*(char *)this->field10_0x1c) {
LINE191     case '\t':
LINE192     case '\n':
LINE193     case '\r':
LINE194     case ' ':
LINE195     case '/':
LINE196     case '>':
LINE197       *(char *)target_buffer = '\0';
LINE198       this->field10_0x1c = this->field10_0x1c + -1;
LINE199       bVar1 = true;
LINE200       break;
LINE201     default:
LINE202       *(char *)target_buffer = *(char *)this->field10_0x1c;
LINE203       target_buffer = (char (*) [256])((int)target_buffer + 1);
LINE204       index = index + 1;
LINE205     }
LINE206     this->field10_0x1c = this->field10_0x1c + 1;
LINE207     if (bVar1) {
LINE208       security_check_cookie(stack_cookie ^ (uint)&stack0xfffffffc);
LINE209       return;
LINE210     }
LINE211   }
LINE212   if (this->field21_0x50 != 0) {
LINE213     index = this->field31_0x78;
LINE214     local_40c = this->field10_0x1c - (int)this->field2_0x8;
LINE215     local_410 = 0;
LINE216     if (0 < index) {
LINE217       piVar2 = (int *)this->field30_0x74;
LINE218       do {
LINE219         if (local_40c < *piVar2) break;
LINE220         local_410 = local_410 + 1;
LINE221         piVar2 = piVar2 + 1;
LINE222       } while (local_410 < index);
LINE223     }
LINE224     if (local_410 < index) {
```

```
LINE225       local_40c = *(int *)((int)this->field30_0x74 + local_410 * 4) -
local_40c;
LINE226     }
LINE227     else {
LINE228       local_40c = 0;
LINE229     }
LINE230     if (this->field38_0x88 == 2) {
LINE231       local_40c = local_40c * 2;
LINE232     }
LINE233     strncpy(string_buffer,"Tag name exceed max character count",0x400);
LINE234     (*(code *)this->field21_0x50)
LINE235
(this,"C:\\BuildAgent\\work\\76e9801d497051a9\\Source\\Common\\Inc_prv\\xmlparser.cp
p"
LINE236                ,0x12f,&local_410);
LINE237   }
LINE238   security_check_cookie(stack_cookie ^ (uint)&stack0xfffffffc);
LINE239   return;
LINE240 }
```

Looking through the parameter `buffer_ovw`, we can observe the code is impacting the table in two places: LINE185 and LINE188. In LINE185, we see it depends on the boolean variable `bVar3`, which is set to true when some specific char is found in LINE183. The side effect of this boolean check isthat it impacts the shift to the start of `target_buffer` LINE188, as it's added to `buffer_ovw` by one byte.
We observe then a while loop starting LINE189 with a hardcoded constant of 256, which is supposed to prevent the overflow of the destination buffer `buffer_ovw`.
But the code is not taking into account the boolean positive result shift and overflows the `buffer_ovw` if the default case is observed for more than 256 bytes.

The condition to make this happen depends on the PSD file records contained, identified as XMP metadata.
This leads to a denial-of-service caused by the `security_check_cookie`. However, depending on how the Imagegear SDK is employed in an application, this could also lead to leaking 1 byte from the canary.

Crash Information

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

FAULTING_IP:
kernel32!UnhandledExceptionFilter+5f
76adffa1 cc              int     3

EXCEPTION_RECORD:  711ae3e0 -- (.exr 0x711ae3e0)
ExceptionAddress: 71089a04 (igCore19d!IG_mpi_page_set+0x0012d9d4)
   ExceptionCode: c0000409 (Security check failure or stack buffer overrun)
  ExceptionFlags: 00000001
NumberParameters: 1
   Parameter[0]: 00000002

CONTEXT:  711ae430 -- (.cxr 0x711ae430;r)
eax=00000000 ebx=029c0150 ecx=8a712c18 edx=00000100 esi=000000a1 edi=000039ac
eip=71089a04 esp=0018f044 ebp=0018f150 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
igCore19d!IG_mpi_page_set+0x12d9d4:
71089a04 8be5            mov     esp,ebp
Last set context:
eax=00000000 ebx=029c0150 ecx=8a712c18 edx=00000100 esi=000000a1 edi=000039ac
eip=71089a04 esp=0018f044 ebp=0018f150 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
igCore19d!IG_mpi_page_set+0x12d9d4:
71089a04 8be5            mov     esp,ebp
Resetting default scope

FAULTING_THREAD:  000008fc

PROCESS_NAME:  Fuzzme.exe

ERROR_CODE: (NTSTATUS) 0x80000003 - {EXCEPTION}  Breakpoint  A breakpoint has been
reached.

EXCEPTION_CODE: (HRESULT) 0x80000003 (2147483651) - One or more arguments are
invalid

EXCEPTION_PARAMETER1:  00000000

NTGLOBALFLAG:  470

APPLICATION_VERIFIER_FLAGS:  0

APP:  fuzzme.exe

ANALYSIS_VERSION: 6.3.9600.17336 (debuggers(dbg).150226-1500) x86fre

BUGCHECK_STR:  APPLICATION_FAULT_STACK_BUFFER_OVERRUN_MISSING_GSFRAME

PRIMARY_PROBLEM_CLASS:  STACK_BUFFER_OVERRUN

DEFAULT_BUCKET_ID:  STACK_BUFFER_OVERRUN
```

```
LAST_CONTROL_TRANSFER:  from 7108950c to 71089a04

STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
0018f150 7108950c 029c0150 0018f1e8 029c0348 igCore19d!IG_mpi_page_set+0x12d9d4
0018f16c 7108966f 029c765d 000039ac 00000001 igCore19d!IG_mpi_page_set+0x12d4dc
0018f1b8 7109003d 0018fc54 029c0150 00000000 igCore19d!IG_mpi_page_set+0x12d63f
0018f670 7104cd0b 0018fc54 1000001e 029c0348 igCore19d!IG_mpi_page_set+0x13400d
0018f738 7104c242 0018fc54 1000001e 029c0098 igCore19d!IG_mpi_page_set+0xf0cdb
0018f774 7104bcba 0018fc54 0018f79c 0018f7c4 igCore19d!IG_mpi_page_set+0xf0212
0018fbcc 70f313d9 0018fc54 029c0060 00000001 igCore19d!IG_mpi_page_set+0xefc8a
0018fc04 70f708d7 00000000 029c0060 0018fc54 igCore19d!IG_image_savelist_get+0xb29
0018fe80 70f70239 00000000 00308230 00000001 igCore19d!IG_mpi_page_set+0x148a7
0018fea0 70f05757 00000000 00308230 00000001 igCore19d!IG_mpi_page_set+0x14209
0018fec0 00402219 00308230 0018fed4 00000001 igCore19d!IG_load_file+0x47
0018fed8 00402524 00308230 0018ff10 003079a8 Fuzzme!fuzzme+0x19
0018ff40 0040668d 00000005 003066a8 003079a8 Fuzzme!fuzzme+0x324
0018ff88 76aa33ca 7efde000 0018ffd4 77d19ed2 Fuzzme!fuzzme+0x448d
0018ff94 77d19ed2 7efde000 7741e483 00000000 kernel32!BaseThreadInitThunk+0xe
0018ffd4 77d19ea5 00406715 7efde000 00000000 ntdll!__RtlUserThreadStart+0x70
0018ffec 00000000 00406715 7efde000 00000000 ntdll!_RtlUserThreadStart+0x1b


FOLLOWUP_IP:
igCore19d!IG_mpi_page_set+12d9d4
71089a04 8be5            mov     esp,ebp

SYMBOL_STACK_INDEX:  0

SYMBOL_NAME:  igcore19d!IG_mpi_page_set+12d9d4

FOLLOWUP_NAME:  MachineOwner

MODULE_NAME: igCore19d

IMAGE_NAME:  igCore19d.dll

DEBUG_FLR_IMAGE_TIMESTAMP:  60aceda9

STACK_COMMAND:  .cxr 0x711ae430 ; kb

FAILURE_BUCKET_ID:  STACK_BUFFER_OVERRUN_80000003_igCore19d.dll!IG_mpi_page_set

BUCKET_ID:
APPLICATION_FAULT_STACK_BUFFER_OVERRUN_MISSING_GSFRAME_MISSING_GSFRAME_igcore19d!IG_
mpi_page_set+12d9d4

ANALYSIS_SOURCE:  UM

FAILURE_ID_HASH_STRING:
um:stack_buffer_overrun_80000003_igcore19d.dll!ig_mpi_page_set

FAILURE_ID_HASH:  {e0459bbd-9052-42e3-75ad-df79bbfa6dcb}

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

Vendor Response

ImageGear Pro v20.0 release

Windows: https://download.accusoft.com/imagegear/pro/ImageGear_for_C_and_CPP_v20.0.exe Linux: https://download.accusoft.com/imagegear/pro/unix/ImageGear_for_C_Cpp20.0.0-Linux64.tar.gz

Documentation Windows: http://help.accusoft.com/ImageGear/v20.0/Windows/DLL/webframe.html Linux: http://help.accusoft.com/ImageGear/v20.0/Linux/webframe.html

https://download.accusoft.com/imagegear/pro/ImageGear_for_C_and_CPP_v20.0.exe

Timeline

2022-02-07 - Vendor disclosure
2022-04-29 - Vendor patched
2022-05-02 - Public Release

CREDIT

Discovered by Emmanuel Tacheau of Cisco Talos.
