Talos Vulnerability Report

TALOS-2022-1464

Leadtools fltSaveCMP integer overflow vulnerability

MARCH 15, 2022

CVE NUMBER

CVE-2022-21154

Summary

An integer overflow vulnerability exists in the fltSaveCMP functionality of Leadtools 22. A specially-crafted BMP file can lead to an integer overflow, that in turn causes a buffer overflow. An attacker can provide a malicious BMP file to trigger this vulnerability.

Tested Versions

Leadtools Leadtools 22

Product URLs

Leadtools - https://www.leadtools.com/

CVSSv3 Score

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

CWE

CWE-190 - Integer Overflow or Wraparound

Details

LEADTOOLS is a collection of comprehensive toolkits to integrate and document medical, multimedia, and imaging technologies into desktop, server, tablet, and mobile applications.

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

```
(14a0.1e64): Access violation - code c0000005 (!!! second chance !!!)
Ltkrnx!L_LicLibGetMachineInfo+0x1bb1e:
00007ffd`9a869a7e f3a4            rep movs byte ptr [rdi],byte ptr [rsi]
0:000> r
rax=0000023de16c9600 rbx=000000000f32fd00 rcx=000000000a6d6300
rdx=0000000060435800 rsi=0000023e46758800 rdi=0000023de6323000
rip=00007ffd9a869a7e rsp=000000517bf5d938 rbp=000000005b31ee00
 r8=000000000f32fd00  r9=000000000f32fd00 r10=0000023e41afee00
r11=0000023de16c9600 r12=0000000000000000 r13=0000000000000000
r14=0000000000000006 r15=0000000000001002
iopl=0         nv up ei pl nz na pe cy
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010201
Ltkrnx!L_LicLibGetMachineInfo+0x1bb1e:
00007ffd`9a869a7e f3a4            rep movs byte ptr [rdi],byte ptr [rsi]
```

The access violation is happening into a function named `copy_bytes_into_buffer` described below.

The exception is happening while writing to `_dest_buffer` at LINE495.

```
LINE1   undefined8 *
LINE2   copy_bytes_into_buffer(undefined8 *dest_buffer,undefined8
*source_buffer,ulonglong size)
LINE3   {
LINE4
        [...]
LINE492                     /* copy_bytes when size is large */
LINE493   _dest_buffer = dest_buffer;
LINE494   for (; size != 0; size = size - 1) {
LINE495     *(undefined *)_dest_buffer = *(undefined *)source_buffer;
LINE496     source_buffer = (undefined8 *)((longlong)source_buffer + 1);
LINE497     _dest_buffer = (undefined8 *)((longlong)_dest_buffer + 1);
LINE498   }
LINE499   return dest_buffer;
LINE500 }
```

This is corresponding to the following disassembly code:

```
LINE501 copy_bytes
LINE502
LINE503 xt:7ff85a749a70 57            PUSH      RDI
LINE504 xt:7ff85a749a71 56            PUSH      RSI
LINE505 xt:7ff85a749a72 49 8b c3      MOV       RAX,R11
LINE506 xt:7ff85a749a75 48 8b f9      MOV       RDI,RCX
dest_buffer
LINE507 xt:7ff85a749a78 49 8b c8      MOV       RCX,R8
r8: size
LINE508 xt:7ff85a749a7b 49 8b f2      MOV       RSI,R10
source_buffer
LINE509 xt:7ff85a749a7e f3 a4         MOVSB.REP RDI,RSI
LINE510 xt:7ff85a749a80 5e            POP       RSI
LINE511 xt:7ff85a749a81 5f            POP       RDI
LINE512 xt:7ff85a749a82 c3            RET
```

Looking at the callstack of the memory allocation for the destination buffer `_dest_buffer` pointed by `rdi`:

```
0:000> !ext.heap -p -a rdi
    address 0000023fdd8f3000 found in
    _DPH_HEAP_ROOT @ 23fbf5a1000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -
VirtAddr         VirtSize)
                             2404b590ea0:      23fd8c99600          4c59a00 -
23fd8c99000          4c5b000
    00007ffdcb10867b ntdll!RtlDebugAllocateHeap+0x000000000000003b
    00007ffdcb03d255 ntdll!RtlpAllocateHeap+0x00000000000000f5
    00007ffdcb03b44d ntdll!RtlpAllocateHeapInternal+0x0000000000000a2d
    00007ffda207bcc8 Ltkrnx!L_LicLibGetMachineInfo+0x000000000002dd68
    00007ffda2038e45 Ltkrnx!L_LocalAlloc+0x00000000000000a5
    00007ffda32eea37 lfCmpX!fltSaveCMP+0x0000000000000897
    00007ffdb353056a Ltfilx!L_GetRasterPdfInfo+0x000000000000132a
    00007ffdb3530ad7 Ltfilx!L_GetRasterPdfInfo+0x0000000000001897
    00007ffdb3531a79 Ltfilx!L_SaveCustomFile+0x0000000000000e69
    00007ffdb3531ad0 Ltfilx!L_SaveBitmap+0x0000000000000040
    00007ff745301341 Fuzzme!fuzzme+0x00000000000000b1
[C:\Users\User\source\repos\Project1\LoadBitmap\LoadBitmap.cpp @ 50]
    00007ff745301c98 Fuzzme!main+0x0000000000000588
[C:\Users\User\source\repos\Project1\LoadBitmap\LoadBitmap.cpp @ 198]
    00007ff7453035d8 Fuzzme!__scrt_common_main_seh+0x000000000000010c
[d:\a01\_work\6\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl @ 288]
    00007ffdc9f57034 KERNEL32!BaseThreadInitThunk+0x0000000000000014
    00007ffdcb062651 ntdll!RtlUserThreadStart+0x0000000000000021
```

The allocation `lfCmpX!fltSaveCMP+0x0000000000000897` where the size of the destination is computed is made inside the function named `lfCmpX!fltSaveCMP`:

```
LINE513   ulonglong lfCmpX!fltSaveCMP(longlong *param_1)
LINE514   {
    ...
LINE515   puVar9 = (uint *)L_LocalAlloc((uVar2 + 1) * BytesPerLine,2,0x210,
LINE516
L"c:\\a2\\_w\\19eb7f22c663b560\\src\\fileformats\\c\\cmp\\common\\cm p.cpp"
LINE517                              );
    ...
```

The allocation buffer is made LINE515 with a call to `L_LocalAlloc` function:

```
LINE518 void L_LocalAlloc(longlong param_1,longlong param_2,undefined8 param_3,
LINE519                  LPCSTR possible_source_code_location)
LINE520
LINE521 {
        [...]
LINE527                     /* 0xa8da0  135  L_LocalAlloc */
LINE528   local_28 = DAT_7ff85a7db010 ^ (ulonglong)auStackY1144;
LINE529   FUN_7ff85a747b90(&local_448,'\0',0x210);
LINE530   if ((possible_source_code_location == (LPCSTR)0x0) ||
(possible_source_code_location[1] != '\0'))
LINE531   {
LINE532     FUN_7ff85a747b90(&local_238,'\0',0x210);
LINE533     if (possible_source_code_location != (LPCSTR)0x0) {
LINE534       MultiByteToWideChar(0,0,possible_source_code_location,0x108,
(LPWSTR)&local_238,0x108);
LINE535     }
LINE536   }
LINE537   else {
LINE538     FUN_7ff85a74d21c(&local_448,0x108,possible_source_code_location);
LINE539   }
LINE540   _malloc_base(param_2 * param_1);
LINE541   handle_canary(local_28 ^ (ulonglong)auStackY1144);
LINE542   return;
LINE543 }
```

We can see into this function `L_LocalAlloc` the call to a wrapper of allocation of memory LINE540, taking the first two parameters of the function and multiplying them to get the total size to allocate. Now below the same code in assembly corresponding to the call to `L_LocalAlloc` LINE515:

```
LINE518                               LAB_7ff8556eea10
XREF[1]:     7ff8556ee9de(j)
LINE519      xt:7ff8556eea10 8b 4b 0c      MOV       ECX,dword ptr [RBX + 0xc]
LINE520                               LAB_7ff8556eea13
XREF[1]:     7ff8556eea0e(j)
LINE521      xt:7ff8556eea13 89 8b         MOV       dword ptr [RBX + 0xe4],ECX
LINE522      xt:7ff8556eea19 4c 8d         LEA       R9,
[u_c:\a2\_w\19eb7f22c663b560\src\fi_7ff  =
u"c:\\a2\\_w\\19eb7f22c663b560\\src\\filefor
LINE523      xt:7ff8556eea20 ff c1         INC       ECX
LINE524      xt:7ff8556eea22 ba 02         MOV       EDX,0x2
LINE525      xt:7ff8556eea27 41 0f         IMUL      ECX,R15D
LINE526      xt:7ff8556eea2b 41 b8         MOV       R8D,0x210
LINE527      xt:7ff8556eea31 ff 15         CALL      qword ptr [-
>LTKRNX.DLL::L_LocalAlloc]      = 8000000000000087
```

As this is an x64 calling convention, the first parameters are in `RCX` register, so we are looking to get the code `(uVar2 + 1) * BytesPerLine` into RCX. LINE521 we can see the `ECX` register used, containing in fact `uVar2` earlier in the code, then incremented by one LINE523, and multiplied by `R15` which is `BytesPerLine`.
The integer overflow happens in LINE525 during the multiplication, as it's a 32-bits based operation, causing an allocation size smaller than really what it should be.

`BytesPerLine` is totally controlled and computed in an earlier function `L_InitBitmapWithCallbacks` :

```
LINE544 void L_InitBitmapWithCallbacks
LINE545                (pBITMAPHANDLE pBitmapHandle,uint uStructSize,int nWidth,int
nHeigth,
LINE546                int nBitsPerPixel,undefined8 param_6)
LINE547
LINE548 {
LINE549        /* 0x21950  405  L_InitBitmapWithCallbacks */
LINE550 ...
LINE551   pBitmapHandle->Width = nWidth;
LINE552   iVar2 = 1;
LINE553   pBitmapHandle->Height = nHeigth;
LINE554   pBitmapHandle->BitsPerPixel = nBitsPerPixel;
LINE555   BytesPerLine = nWidth * nBitsPerPixel + 0x1fU >> 3 & 0x1ffffffc;
LINE556   pBitmapHandle->BytesPerLine = BytesPerLine;
LINE557 ...
LINE558 }
```

In LINE555 we can see `BytesPerLine` computed from the `nWidth` variable directly read from the file and `nBitsPerPixel` passed as a argument to the `L_SaveBitmap` call function, leading to the crash. The loop counter, which is controlling the `rep movs byte ptr` LINE509 in the assembly presented earlier, causes the crash and corresponds to the `BytesPerLine` which is also controlled. Thus a specially-crafted BMP file could trigger this integer overflow, which could lead to memory corruption.

Crash Information

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

*** WARNING: Unable to verify checksum for C:\Program Files\Talos Vrt
Team\LeadToolsFuzzing\bin\Fuzzme.exe

KEY_VALUES_STRING: 1

    Key  : AV.Fault
    Value: Write

    Key  : Analysis.CPU.mSec
    Value: 2890

    Key  : Analysis.DebugAnalysisManager
    Value: Create

    Key  : Analysis.Elapsed.mSec
    Value: 7159

    Key  : Analysis.Init.CPU.mSec
    Value: 1171

    Key  : Analysis.Init.Elapsed.mSec
    Value: 15966

    Key  : Analysis.Memory.CommitPeak.Mb
    Value: 79

    Key  : Timeline.OS.Boot.DeltaSec
    Value: 12422

    Key  : Timeline.Process.Start.DeltaSec
    Value: 21

    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


NTGLOBALFLAG:  2000000

APPLICATION_VERIFIER_FLAGS:  0

APPLICATION_VERIFIER_LOADED: 1

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 00007ffd9a869a7e
(Ltkrnx!L_LicLibGetMachineInfo+0x000000000001bb1e)
```

```
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 0000000000000001
   Parameter[1]: 0000023de6323000
Attempt to write to address 0000023de6323000

FAULTING_THREAD:  00001e64

PROCESS_NAME:  Fuzzme.exe

WRITE_ADDRESS:  0000023de6323000

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:  0000000000000001

EXCEPTION_PARAMETER2:  0000023de6323000

STACK_TEXT:
00000051`7bf5d938 00007ffd`9a82887f     : 00000000`00000000 00000000`00000000
00008cca`61bd98fb 00007ffd`9a7f128c : Ltkrnx!L_LicLibGetMachineInfo+0x1bb1e
00000051`7bf5d950 00007ffd`9a828f56     : 00000051`7bf5d9e0 0000023d`cd361ee0
00000000`00000006 00000000`0f32fd00 : Ltkrnx!L_RotateBitmapViewPerspective+0x24f
00000051`7bf5d980 00007ffd`b3801595     : 0000023d`cd361ee0 0000023e`5409eee0
0000023d`e16c9600 00007ffd`9a7f8bf5 : Ltkrnx!L_GetBitmapRow+0x116
00000051`7bf5d9c0 00007ffd`99d3ec62     : 0000023e`54090cc0 00000051`7bf5dc90
0000023d`e16c9600 0000023d`00000007 : Ltfilx!L_SaveCustomFile+0x985
00000051`7bf5da70 00007ffd`b380056a     : 00000000`00001002 00000051`00001002
0000023d`00000000 00000000`00000006 : lfCmpX!fltSaveCMP+0xac2
00000051`7bf5dbd0 00007ffd`b3800ad7     : 00000000`00000013 00000000`00000000
00000000`00000000 00000000`00000000 : Ltfilx!L_GetRasterPdfInfo+0x132a
00000051`7bf5e790 00007ffd`b3801a79     : 00007ff7`45305578 00007ffd`c87ab3e4
00007ffd`c887f4e8 00000000`00000000 : Ltfilx!L_GetRasterPdfInfo+0x1897
00000051`7bf5efe0 00007ffd`b3801ad0     : 00000000`00000000 00000051`7bf5f330
0000023d`c90adfe0 00000051`7bf5f0b8 : Ltfilx!L_SaveCustomFile+0xe69
00000051`7bf5f060 00007ff7`45301341     : 00007ff7`45305578 0000023d`c90adfe0
0000023d`cd3117c0 00000000`00000001 : Ltfilx!L_SaveBitmap+0x40
00000051`7bf5f0b0 00007ff7`45301c98     : 0000023d`c80e3000 ffffffff`00000076
0000023d`ccba0fe0 00000051`7bf5f290 : Fuzzme!fuzzme+0xb1
00000051`7bf5f230 00007ff7`453035d8     : 0000023d`c907dfa0 00000000`00000000
00000000`00000000 0000023d`c8466ea0 : Fuzzme!main+0x588
00000051`7bf5f990 00007ffd`c9f57034     : 00000000`00000000 00000000`00000000
00000000`00000000 00000000`00000000 : Fuzzme!__scrt_common_main_seh+0x10c
00000051`7bf5f9d0 00007ffd`cb062651     : 00000000`00000000 00000000`00000000
00000000`00000000 00000000`00000000 : KERNEL32!BaseThreadInitThunk+0x14
00000051`7bf5fa00 00000000`00000000     : 00000000`00000000 00000000`00000000
00000000`00000000 00000000`00000000 : ntdll!RtlUserThreadStart+0x21


SYMBOL_NAME:  Ltkrnx!L_LicLibGetMachineInfo+1bb1e

MODULE_NAME: Ltkrnx

IMAGE_NAME:  Ltkrnx.dll
```

```
STACK_COMMAND:  dt ntdll!LdrpLastDllInitializer BaseDllName ; dt
ntdll!LdrpFailureData ; ~0s ; .cxr ; kb

FAILURE_BUCKET_ID:
INVALID_POINTER_WRITE_STRING_DEREFERENCE_AVRF_c0000005_Ltkrnx.dll!L_LicLibGetMachine
Info

OS_VERSION:  10.0.19041.1

BUILDLAB_STR:  vb_release

OSPLATFORM_TYPE:  x64

OSNAME:  Windows 10

IMAGE_VERSION:  22.0.0.9

FAILURE_ID_HASH:  {6a0d6765-25be-9357-f56d-a64cbc643a17}

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

Vendor Response

The fix is included in C / C++: LfCmp v21.0.0.12 v22.0.0.6

.NET / Java: Leadtools.Codecs.Cmp v21.0.0.13 v22.0.0.6

https://files.leadtools.com/index.php/s/joFz7BcCZYMot5Q

Timeline

2022-02-02 - Initial vendor contact

2022-03-15 - Public Release

CREDIT

Discovered by Emmanuel Tacheau of Cisco Talos.
