Talos Vulnerability Report

TALOS-2021-1350

LibreCad libdxfrw dwgCompressor::copyCompBytes21 heap-based buffer overflow vulnerability

NOVEMBER 17, 2021

CVE NUMBER

CVE-2021-21899

SUMMARY

A code execution vulnerability exists in the dwgCompressor::copyCompBytes21 functionality of LibreCad libdxfrw 2.2.0-rc2-19-ge02f3580. A specially-crafted .dwg file can lead to a heap buffer overflow. An attacker can provide a malicious file to trigger this vulnerability.

CONFIRMED VULNERABLE VERSIONS

The versions below were either tested or verified to be vulnerable by Talos or confirmed to be vulnerable by the vendor.

LibreCad libdxfrw 2.2.0-rc2-19-ge02f3580

PRODUCT URLS

libdxfrw - https://librecad.org/

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-119 - Improper Restriction of Operations within the Bounds of a Memory Buffer

DETAILS

Libdfxfw is an opensource library facilitating the reading and writing of .dxf and .dwg files, the primary vector graphics file formats of CAD software. Libdxfrw is contained in and primarily used by LibreCAD for the aforementioned purposes.

Libdxfrw is capable of reading in many different versions of .dwg files. The backwards compatibility is quite extensive, and likewise the particulars for each given version are also quite extensive. For today's vulnerability, we deal with the `AC1021` version, the AutoCAD .dwg standard from 2007 through 2009.

To keep things relatively short, let's examine how libdxfrw reads in `AC1021` .dwg headers:

```
bool dwgReader21::readFileHeader() {

 //DRW_DBG("\n\ndwgReader21::parsing file header\n");
 if (! fileBuf->setPosition(0x80))               // [1]
     return false;
 duint8 fileHdrRaw[0x2FD];//0x3D8
 fileBuf->getBytes(fileHdrRaw, 0x2FD);  // [2]
 duint8 fileHdrdRS[0x2CD];
 dwgRSCodec::decode239I(fileHdrRaw, fileHdrdRS, 3); // [3]
```

As shown by [1], .dwg file headers start at offset 0x80, an example of which will soon be posted. At [2], the headers are read into a static buffer, and at [3] the headers are decoded. Thus, for an input file of:

```
00000000  41 43 31 30 32 31 00 00  00 00 00 25 03 c0 00 1d  |AC1021.....%....|
00000010  25 00 01 00 00 00 00 00  00 20 01 00 00 00 00 00  |%........ ......|
00000020  00 80 00 00 00 e0 11 02  00 00 15 02 00 00 00 01  |................|
00000030  01 00 00 00 00 00 01 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000070  00 00 00 00 00 00 00 00  00 68 f8 f7 92 2a b5 ef  |.........h...*..|
00000080  18 dd 0b f1 f1 bb e9 eb  00 00 00 01 00 00 00 09  |................|
00000090  00 41 00 75 00 74 00 6f  00 64 00 65 00 73 00 6b  |.A.u.t.o.d.e.s.k|
000000a0  06 50 04 31 40 40 4d 5c  a7 22 98 1d 02 54 01 48  |.P.1@@M\."...T.H|
000000b0  01 18 44 ab 54 f9 2d d1  04 41 27 d4 22 78 3b a1  |..D.T.-..A'."x;.|
000000c0  22 bd 22 91 9d e1 8b 1f  da 54 9a fc 5f           |"."......T.._|
000000cd
```

We end up with a decoded input buffer of:

```
[~.~]> x/191wx fileHdrdRS
0x7fff66b00b70: 0x00e9f118      0x74000900      0x50006500      0x54985c40
0x7fff66b00b80: 0x412dab01      0xe122a122      0x0000fcda      0x00000000
0x7fff66b00b90: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fff66b00ba0: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fff66b00bb0: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fff66b00bc0: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fff66b00bd0: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fff66b00be0: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fff66b00bf0: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fff66b00c00: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fff66b00c10: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fff66b00c20: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fff66b00c30: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fff66b00c40: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fff66b00c50: 0x00000000      0x00000000      0x00000000      0xdd000000
0x7fff66b00c60: 0x0000ebf1      0x64007500      0x40046b00      0x18011da7
0x7fff66b00c70: 0x7827d154      0x548b9122      0x0000005f      0x00000000
0x7fff66b00c80: 0x00000000      0x00000000      0x00000000      0x00000000
```

Immediately following the decoding, we populate two more variables, `fileHdrCompLength` and `fileHdrCompLength2`:

```
 dwgRSCodec::decode239I(fileHdrRaw, fileHdrdRS, 3);

dint32 fileHdrCompLength = fileHdrBuf.getRawLong32();
dcint32 fileHdrCompLength2 = fileHdrBuf.getRawLong32();
```

Which get populated with 0x0000fcda and 0x00000000 respectively from the decoded buffer:

```
[~.~]> x/191wx fileHdrdRS
0x7fff66b00b70: 0x00e9f118      0x74000900      0x50006500      0x54985c40
0x7fff66b00b80: 0x412dab01      0xe122a122      0x0000fcda      0x00000000  //<- fileHdrCompLength/fileHdrCompLength2
```

Needless to say, the `fileHdrCompLength` variable is completely controlled by the input file. Continuing on within `dwgReader21::readFileHeader()`:

```
 dwgRSCodec::decode239I(fileHdrRaw, fileHdrdRS, 3);

dint32 fileHdrCompLength = fileHdrBuf.getRawLong32();
dcint32 fileHdrCompLength2 = fileHdrBuf.getRawLong32();

int fileHdrDataLength = 0x110;
std::vector<duint8> fileHdrData;
if (fileHdrCompLength < 0) {           // [4]
    // [...] no compression
}else {                                // [5]
    //DRW_DBG("\ndwgReader21:: file header are compresed:\n");
    std::vector<duint8> compByteStr(fileHdrCompLength);
    fileHdrBuf.getBytes(compByteStr.data(), fileHdrCompLength);
    fileHdrData.resize(fileHdrDataLength);   // [6]
    dwgCompressor::decompress21(compByteStr.data(), &fileHdrData.front(), // [7]
                                fileHdrCompLength, fileHdrDataLength);
}
```

The branch at [4] is for a .dwg file without compression, but frankly that's boring so it's skipped. At [5] we deal with the much more fun and compressed .dwg files. Since we control `fileHdrCompLength`, we have a choice of branch, and we go with compressed. At [6], a static sized buffer of 0x110 bytes is created to contain the resultant decompressed header bytes, and the decompression itself occurs at [7] in `dwgCompressor::decompress21()`, which we now examine:

```
void dwgCompressor::decompress21(duint8 *cbuf, duint8 *dbuf, duint32 csize, duint32 dsize){
    duint32 srcIndex=0;         // controlled, 0x110 malloc, controlled,    0x110, always
    duint32 dstIndex=0;
    duint32 length=0;
    duint32 sourceOffset;
    duint8 opCode;

    opCode = cbuf[srcIndex++];
    if ((opCode >> 4) == 2){
        srcIndex = srcIndex +2;
        length = cbuf[srcIndex++] & 0x07;
    }

    while (srcIndex < csize && (dstIndex < dsize+1)){  // [8] //dstIndex < dsize to prevent crash more robust are needed
        if (length == 0)
            length = litLength21(cbuf, opCode, &srcIndex);           // [9]
        copyCompBytes21(cbuf, dbuf, length, srcIndex, dstIndex);   // [10]
        srcIndex += length;
        dstIndex += length;
        if (dstIndex >=dsize) break; // [11] //check if last chunk are compresed & terminate
```

To ellucidate and reiterate, the `cbuf` and csize arguments are fully controlled by the input file, while `dbuf` and`dsize` are a 0x110 sized allocation and corresponding size. This hopefully gives away the vulnerability, as the only real length checks are at the aptly-commented [8] and [11]. At [9] we find the `length` field to pass into the `copyCompBytes21()` function at [10] (assuming that length is currently 0x0, something fully within the file's control). To proceed, we first look at `litLength21()` and then `copyCompBytes21`:

```
duint32 dwgCompressor::litLength21(duint8 *cbuf, duint8 oc, duint32 *si){

    duint32 srcIndex=*si;

    duint32 length = oc + 8;
    if (length == 0x17) {
        duint32 n = cbuf[srcIndex++];
        length += n;
        if (n == 0xff) {
            do {
                n = cbuf[srcIndex++];
                n |= (duint32)(cbuf[srcIndex++] << 8);
                length += n;
            } while (n == 0xffff);
        }
    }

    *si = srcIndex;
    return length;
}
```

Not worth going into too much detail; all we really care about is that the return `length` can be any value from 0x0 to 0xFFFFFFFF. With that in mind we now peek at `copyCompBytes21(cbuf, dbuf, length, srcIndex, dstIndex)`:

```
void dwgCompressor::copyCompBytes21(duint8 *cbuf, duint8 *dbuf, duint32 l, duint32 si, duint32 di){
    duint32 length =l;
    duint32 dix = di;
    duint32 six = si;

    while (length > 31){
            //in doc: 16-31, 0-15
        for (duint32 i = six+24; i<six+32; i++)
            dbuf[dix++] = cbuf[i];
        for (duint32 i = six+16; i<six+24; i++)
            dbuf[dix++] = cbuf[i];
        for (duint32 i = six+8; i<six+16; i++)
            dbuf[dix++] = cbuf[i];
        for (duint32 i = six; i<six+8; i++)
            dbuf[dix++] = cbuf[i];
        six = six + 32;
        length = length -32;
    }

    switch (length) {
        case 0:
            break;
        case 1: //Ok
            dbuf[dix] = cbuf[six];
            break;
        case 2: //Ok
            dbuf[dix++] = cbuf[six+1];
            dbuf[dix] = cbuf[six];
            break;

  //[...]

           case 31:
        //in doc: six+30, 26-29, 18-25, 2-17, 0-1
        dbuf[dix++] = cbuf[six+30];
        for (int i = 26; i<30;i++)
            dbuf[dix++] = cbuf[six+i];
        for (int i = 18; i<26;i++)
            dbuf[dix++] = cbuf[six+i];
/*        for (int i = 2; i<18; i++)
            dbuf[dix++] = cbuf[six+i];*/
        for (int i = 10; i<18; i++)
            dbuf[dix++] = cbuf[six+i];
        for (int i = 2; i<10; i++)
            dbuf[dix++] = cbuf[six+i];
        dbuf[dix++] = cbuf[six+1];
        dbuf[dix] = cbuf[six];
        break;
    default:
        DRW_DBG("WARNING dwgCompressor::copyCompBytes21, bad output.\n");
        break;
} }
```

As one can quite clearly see, still no length checks anywhere within this function. If we pass an input buffer that is greater than 0x110 bytes, or if we pass a compressed chunk when our dstIndex is at 0x10F, we end up copying bytes directly from our fully controlled input buffer into the fixed 0x110 size heap buffer, resulting in a fully controlled heap buffer overflow.

Crash Information

=== ==996143==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x6120000005d0 at pc 0x7f34d4675a85 bp 0x7fff66afecb0 sp 0x7fff66afeca8 WRITE of size 1 at 0x6120000005d0 thread T0 #0 0x7f34d4675a84 in dwgCompressor::copyCompBytes21(unsigned char, unsigned char, unsigned int, unsigned int, unsigned int) /root/boop/assorted_fuzzing/librecad/LibreCAD_master/libraries/libdxfrw/src/intern/dwgutil.cpp:423:25 #1 0x7f34d46722d5 in dwgCompressor::decompress21(unsigned char, unsigned char, unsigned int, unsigned int) /root/boop/assorted_fuzzing/librecad/LibreCAD_master/libraries/libdxfrw/src/intern/dwgutil.cpp:272:9 #2 0x7f34d461feb2 in dwgReader21::readFileHeader() /root/boop/assorted_fuzzing/librecad/LibreCAD_master/libraries/libdxfrw/src/intern/dwgreader21.cpp:167:9 #3 0x7f34d4128364 in dwgR::read(DRW_Interface, bool) /root/boop/assorted_fuzzing/librecad/LibreCAD_master/libraries/libdxfrw/src/libdwgr.cpp:156:24 #4 0x55073c in LLVMFuzzerTestOneInput /root/boop/assorted_fuzzing/librecad/fuzzing/dwg/./dwg_harness.cpp:65:9 #5 0x4587e1 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const, unsigned long) (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x4587e1) #6 0x443f52 in fuzzer::RunOneTest(fuzzer::Fuzzer, char const, unsigned long) (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x443f52) #7 0x449a06 in fuzzer::FuzzerDriver(int, char*, int ()(unsigned char const*, unsigned long)) (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x449a06) #8 0x4726c2 in main (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x4726c2) #9 0x7f34d37430b2 in __libc_start_main /build/glibc-eX1tMB/glibc-2.31/csu/../csu/libc-start.c:308:16 #10 0x41e61d in _start (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x41e61d)

0x6120000005d0 is located 0 bytes to the right of 272-byte region [0x6120000004c0,0x6120000005d0) allocated by thread T0 here: #0 0x54da9d in operator new(unsigned long) (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x54da9d) #1 0x7f34d45ba38b in __gnu_cxx::new_allocator::allocate(unsigned long, void const*) /usr/bin/../lib/gcc/x86_64-linux-gnu/9/../../../../include/c++/9/ext/new_allocator.h:114:27 #2 0x7f34d45ba2b8 in std::allocator_traits<std::allocator >::allocate(std::allocator&, unsigned long) /usr/bin/../lib/gcc/x86_64-linux-gnu/9/../../../../include/c++/9/bits/alloc_traits.h:444:20 #3 0x7f34d45b985f in std::_Vector_base<unsigned char, std::allocator >::_M_allocate(unsigned long) /usr/bin/../lib/gcc/x86_64-linux-gnu/9/../../../../include/c++/9/bits/stl_vector.h:343:20 #4 0x7f34d45b8d46 in std::vector<unsigned char, std::allocator >::_M_default_append(unsigned long) /usr/bin/../lib/gcc/x86_64-linux-gnu/9/../../../../include/c++/9/bits/vector.tcc:635:34 #5 0x7f34d4584342 in std::vector<unsigned char, std::allocator >::resize(unsigned long) /usr/bin/../lib/gcc/x86_64-linux-gnu/9/../../../../include/c++/9/bits/stl_vector.h:937:4 #6 0x7f34d461fe65 in dwgReader21::readFileHeader() /root/boop/assorted_fuzzing/librecad/LibreCAD_master/libraries/libdxfrw/src/intern/dwgreader21.cpp:166:21 #7 0x7f34d4128364 in dwgR::read(DRW_Interface*, bool) /root/boop/assorted_fuzzing/librecad/LibreCAD_master/libraries/libdxfrw/src/libdwgr.cpp:156:24 #8 0x55073c in LLVMFuzzerTestOneInput /root/boop/assorted_fuzzing/librecad/fuzzing/dwg/./dwg_harness.cpp:65:9 #9 0x4587e1 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x4587e1) #10 0x443f52 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long) (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x443f52) #11 0x449a06 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long)) (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x449a06) #12 0x4726c2 in main (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x4726c2) #13 0x7f34d37430b2 in __libc_start_main /build/glibc-eX1tMB/glibc-2.31/csu/../csu/libc-start.c:308:16

SUMMARY: AddressSanitizer: heap-buffer-overflow /root/boop/assorted_fuzzing/librecad/LibreCAD_master/libraries/libdxfrw/src/intern/dwgutil.cpp:423:25 in dwgCompressor::copyCompBytes21(unsigned char, unsigned char, unsigned int, unsigned int, unsigned int) Shadow bytes around the buggy address: 0x0c247fff8060: fa fa fa fa fa fa fa fa 00 00 00 00 00 00 00 00 0x0c247fff8070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0x0c247fff8080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 fa fa 0x0c247fff8090: fa fa fa fa fa fa fa fa 00 00 00 00 00 00 00 00 0x0c247fff80a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 =>0x0c247fff80b0: 00 00 00 00 00 00 00 00 00 00 00[fa]fa fa fa fa fa 0x0c247fff80c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa 0x0c247fff80d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa 0x0c247fff80e0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa 0x0c247fff80f0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa 0x0c247fff8100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa Shadow byte legend (one shadow byte represents 8 application bytes): Addressable: 00 Partially addressable: 01 02 03 04 05 06 07 Heap left redzone: fa Freed heap region: fd Stack left redzone: f1 Stack mid redzone: f2 Stack right redzone: f3 Stack after return: f5 Stack use after scope: f8 Global redzone: f9 Global init order: f6 Poisoned by user: f7 Container overflow: fc Array cookie: ac Intra object redzone: bb ASan internal: fe Left alloca redzone: ca Right alloca redzone: cb Shadow gap: cc ==996143==ABORTING

TIMELINE

2021-08-04 - Vendor Disclosure
2021-11-10 - Vendor Patched
2021-11-17 - Public Release

CREDIT

Discovered by Lilith >_> of Cisco Talos.
