Talos Vulnerability Report

TALOS-2021-1349

LibreCad libdxfrw dwgCompressor::decompress18() out-of-bounds write vulnerability

NOVEMBER 17, 2021

CVE NUMBER

CVE-2021-21898

SUMMARY

A code execution vulnerability exists in the dwgCompressor::decompress18() functionality of LibreCad libdxfrw 2.2.0-rc2-19-ge02f3580. A specially-crafted .dwg file can lead to an out-of-bounds write. 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, but c

Libdxfrw is capable of reading many different versions of .dwg files. The backwards compatibility is quite extensive, and likewise the particulars for each version are quite extensive. For today's vulnerability, we deal with the `AC1024` version, the AutoCAD .dwg standard from 2010 through 2012. At least for Librecad and libdxfrw, this `AC1024` version oddly ends up hitting the same code as the `AC1018` version, the AutoCAD .dwg standard from 2004 through 2007. Even if a function ends in `18` (e.g. dwgReader18::readFileHeader), this vulnerability is applicable to both versions for libdxfrw.

To start, the `dwgReader24::readFileHeader` function is a good place to begin:

```
bool dwgReader24::readFileHeader() {
    DRW_DBG("dwgReader24::readFileHeader\n");
    bool ret = dwgReader18::readFileHeader();
    DRW_DBG("dwgReader24::readFileHeader END\n");
    return ret;
}
```

Simple and easy, this function immediately calls into `dwgReader18::readFileHeader`, which is a bit of a doozy. The below version of it is with all the `DRW_DBG` logging code removed :

```
bool dwgReader18::readFileHeader() {

    if (! fileBuf->setPosition(0x80))  // [1]
        return false;

    //    genMagicNumber(); DBG("\n"); DBG("\n");
    duint8 byteStr[0x6C];
    int size =0x6C;
    for (int i=0, j=0; i< 0x6C;i++) {
        duint8 ch = fileBuf->getRawChar8();

        byteStr[i] = DRW_magicNum18[i] ^ ch;
    }

   //[ ... ]

    dwgBuffer buff(byteStr, 0x6C, &decoder);
```

The first part of this function simply reads file offsets (0x80,0x10C) and then xors them against a hardcoded magic set of bytes:

```
static const int DRW_magicNum18[] = {
    0x29, 0x23, 0xbe, 0x84, 0xe1, 0x6c, 0xd6, 0xae,
    0x52, 0x90, 0x49, 0xf1, 0xf1, 0xbb, 0xe9, 0xeb,
    0xb3, 0xa6, 0xdb, 0x3c, 0x87, 0x0c, 0x3e, 0x99,
    0x24, 0x5e, 0x0d, 0x1c, 0x06, 0xb7, 0x47, 0xde,
    0xb3, 0x12, 0x4d, 0xc8, 0x43, 0xbb, 0x8b, 0xa6,
    0x1f, 0x03, 0x5a, 0x7d, 0x09, 0x38, 0x25, 0x1f,
    0x5d, 0xd4, 0xcb, 0xfc, 0x96, 0xf5, 0x45, 0x3b,
    0x13, 0x0d, 0x89, 0x0a, 0x1c, 0xdb, 0xae, 0x32,
    0x20, 0x9a, 0x50, 0xee, 0x40, 0x78, 0x36, 0xfd,
    0x12, 0x49, 0x32, 0xf6, 0x9e, 0x7d, 0x49, 0xdc,
    0xad, 0x4f, 0x14, 0xf2, 0x44, 0x40, 0x66, 0xd0,
    0x6b, 0xc4, 0x30, 0xb7, 0x32, 0x3b, 0xa1, 0x22,
    0xf6, 0x22, 0x91, 0x9d, 0xe1, 0x8b, 0x1f, 0xda,
    0xb0, 0xca, 0x99, 0x02
};
```

To proceed:

```
bool dwgReader18::readFileHeader() {
   // [...]

    dint32 secPageMapId = buff.getRawLong32();

    duint64 secPageMapAddr = buff.getRawLong64()+0x100;

    duint32 secMapId = buff.getRawLong32();

    //TODO: verify CRC
    for (duint8 i = 0x68; i < 0x6c; ++i)
        byteStr[i] = '\0';
    duint32 crcCalc = buff.crc32(0x00,0,0x6C);

    // At this point are parsed the first 256 bytes

    if (! fileBuf->setPosition(secPageMapAddr))  //
        return false;
    duint32 pageType = fileBuf->getRawLong32();
    duint32 decompSize = fileBuf->getRawLong32();
    if (pageType != 0x41630e3b){  // [2]
        return false;
    }
    std::vector<duint8> tmpDecompSec(decompSize);
    parseSysPage(tmpDecompSec.data(), decompSize);
```

Without getting too in-depth into the code here, the important thing to note is that certain types of “pages” are expected at offsets given by our file headers. As shown above at [2], a pageType of 0x41630e3b is expected, which corresponds to a `SysPage`. We now delve into `dwgReader18::parseSysPage`, who’s arguments are an allocated buffer with its size, which we control:

```
//called: Section page map: 0x41630e3b
void dwgReader18::parseSysPage(duint8 *decompSec, duint32 decompSize){

    duint32 compSize = fileBuf->getRawLong32();  //[3]

    duint8 hdrData[20];
    fileBuf->moveBitPos(-160);
    fileBuf->getBytes(hdrData, 20);
    for (duint8 i= 16; i<20; ++i)
        hdrData[i]=0;
    duint32 calcsH = checksum(0, hdrData, 20);

    std::vector<duint8> tmpCompSec(compSize);  //[4]
    fileBuf->getBytes(tmpCompSec.data(), compSize); //[5]
    duint32 calcsD = checksum(calcsH, tmpCompSec.data(), compSize);

    dwgCompressor comp;
    comp.decompress18(tmpCompSec.data(), decompSec, compSize, decompSize); // [6]
}
```

At [3] we grab the size of the compressed data, and at [4] we allocate a temporary buffer of that size. At [5] we populate the buffer with our compressed data from our file, and then at [6] we do the decompression. Please note the lack of comparison of the `decompSize` to the `compSize` before we continue on into `dwgReader18::decompress18()`:

```
void dwgCompressor::decompress18(duint8 *cbuf, duint8 *dbuf, duint32 csize, duint32 dsize){
    bufC = cbuf;
    bufD = dbuf;
    sizeC = csize -2;
    sizeD = dsize;
    sizeC = csize;

    duint32 compBytes;
    duint32 compOffset;
    duint32 litCount;

    pos=0; //current position in compresed buffer
    rpos=0; //current position in resulting decompresed buffer
    litCount = litLength18();              // [7]
    //copy first lileral lenght
    for (duint32 i=0; i < litCount; ++i) {
        bufD[rpos++] = bufC[pos++];          // [8]
    }

    while (pos < csize && (rpos < dsize+1)){//rpos < dsize to prevent crash more robust are needed
        duint8 oc = bufC[pos++]; //next opcode
        if (oc == 0x10){
            compBytes = longCompressionOffset()+ 9;
            compOffset = twoByteOffset(&litCount) + 0x3FFF;
            if (litCount == 0)
                litCount= litLength18();
        } else if (oc > 0x11 && oc< 0x20){
            compBytes = (oc & 0x0F) + 2;
            compOffset = twoByteOffset(&litCount) + 0x3FFF;
            if (litCount == 0)
                litCount= litLength18();

            // [...]
        }
        //copy "compresed data", TODO Needed verify out of bounds
        duint32 remaining = sizeD - (litCount+rpos);
        if (remaining < compBytes){
            compBytes = remaining;
        }
        for (duint32 i=0, j= rpos - compOffset -1; i < compBytes; i++) {
            bufD[rpos++] = bufD[j++];
        }
        //copy "uncompresed data", TODO Needed verify out of bounds
        for (duint32 i=0; i < litCount; i++) {
            bufD[rpos++] = bufC[pos++];
        }
    }
```

Needless to say, basically every line in this function can be either an out-of-bounds read or an out-of-bounds write on the heap, but we're going to keep things short and only look at [7] and [8] (since the rest is the same anyways). Remember that both the size of our compressed and uncompressed buffers are controlled entirely by the file, and then let us examine the `litLength18()` function at [7]:

```
duint32 dwgCompressor::litLength18(){
    duint32 cont=0;
    duint8 ll = bufC[pos++];
    //no literal length, this byte is next opCode
    if (ll > 0x0F) {
        pos--;
        return 0;
    }

    if (ll == 0x00) {
        cont = 0x0F;
        ll = bufC[pos++];
        while (ll == 0x00){//repeat until ll != 0x00
            cont +=0xFF;
            ll = bufC[pos++];
        }
    }
    cont +=ll;
    cont +=3; //already sum 3
    return cont;
}
```

The length returned from this function is also completely arbitrary. So back up to `dwgCompressor::decompress18()`:

```
void dwgCompressor::decompress18(duint8 *cbuf, duint8 *dbuf, duint32 csize, duint32 dsize){
    bufC = cbuf;
    bufD = dbuf;
    sizeC = csize -2;
    sizeD = dsize;
    sizeC = csize;

    duint32 compBytes;
    duint32 compOffset;
    duint32 litCount;

    pos=0; //current position in compresed buffer
    rpos=0; //current position in resulting decompresed buffer
    litCount = litLength18();              // [9]
    //copy first lileral lenght
    for (duint32 i=0; i < litCount; ++i) { // [10]
        bufD[rpos++] = bufC[pos++];
    }
```

Since there's no check on the size of the length from [9] to [10], there's a trivial out-of-bounds write on the heap, fully controlled by the input file.

Crash Information

```
==1043701==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x62c000007600 at pc 0x7f6d842ee9cb bp 0x7fffea9c2c10 sp 0x7fffea9c2c08 WRITE of size 1 at 0x62c000007600 thread T0 #0 0x7f6d842ee9ca in dwgCompressor::decompress18(unsigned char, unsigned char, unsigned int, unsigned int) /root/boop/assorted_fuzzing/librecad/LibreCAD_master/libraries/libdxfrw/src/intern/dwgutil.cpp:208:26 #1 0x7f6d842c39b7 in dwgReader18::parseDataPage(dwgSectionInfo) /root/boop/assorted_fuzzing/librecad/LibreCAD_master/libraries/libdxfrw/src/intern/dwgreader18.cpp:168:14 #2 0x7f6d842d0c54 in dwgReader18::readDwgHeader(DRW_Header&) /root/boop/assorted_fuzzing/librecad/LibreCAD_master/libraries/libdxfrw/src/intern/dwgreader18.cpp:410:16 #3 0x7f6d8432b37b in dwgReader24::readDwgHeader(DRW_Header&) /root/boop/assorted_fuzzing/librecad/LibreCAD_master/libraries/libdxfrw/src/intern/dwgreader24.cpp:33:29 #4 0x7f6d83da7189 in dwgR::processDwg() /root/boop/assorted_fuzzing/librecad/LibreCAD_master/libraries/libdxfrw/src/libdwgr.cpp:241:19 #5 0x7f6d83da6410 in dwgR::read(DRW_Interface, bool) /root/boop/assorted_fuzzing/librecad/LibreCAD_master/libraries/libdxfrw/src/libdwgr.cpp:158:20 #6 0x55073c in LLVMFuzzerTestOneInput /root/boop/assorted_fuzzing/librecad/fuzzing/dwg/./dwg_harness.cpp:65:9 #7 0x4587e1 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const, unsigned long) (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x4587e1) #8 0x443f52 in fuzzer::RunOneTest(fuzzer::Fuzzer, char const, unsigned long) (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x443f52) #9 0x449a06 in fuzzer::FuzzerDriver(int, char*, int ()(unsigned char const*, unsigned long)) (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x449a06) #10 0x4726c2 in main (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x4726c2) #11 0x7f6d833c10b2 in __libc_start_main /build/glibc-eX1tMB/glibc-2.31/csu/../csu/libc-start.c:308:16 #12 0x41e61d in _start (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x41e61d)

0x62c000007600 is located 0 bytes to the right of 29696-byte region [0x62c000000200,0x62c000007600) 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 0x7f6d8423838b 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 0x7f6d842382b8 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 0x7f6d8423785f 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 0x7f6d84236d46 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 0x7f6d84202342 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 0x7f6d842c0bc2 in dwgReader18::parseDataPage(dwgSectionInfo) /root/boop/assorted_fuzzing/librecad/LibreCAD_master/libraries/libdxfrw/src/intern/dwgreader18.cpp:111:13 #7 0x7f6d842d0c54 in dwgReader18::readDwgHeader(DRW_Header&) /root/boop/assorted_fuzzing/librecad/LibreCAD_master/libraries/libdxfrw/src/intern/dwgreader18.cpp:410:16 #8 0x7f6d8432b37b in dwgReader24::readDwgHeader(DRW_Header&) /root/boop/assorted_fuzzing/librecad/LibreCAD_master/libraries/libdxfrw/src/intern/dwgreader24.cpp:33:29 #9 0x7f6d83da7189 in dwgR::processDwg() /root/boop/assorted_fuzzing/librecad/LibreCAD_master/libraries/libdxfrw/src/libdwgr.cpp:241:19 #10 0x7f6d83da6410 in dwgR::read(DRW_Interface*, bool) /root/boop/assorted_fuzzing/librecad/LibreCAD_master/libraries/libdxfrw/src/libdwgr.cpp:158:20 #11 0x55073c in LLVMFuzzerTestOneInput /root/boop/assorted_fuzzing/librecad/fuzzing/dwg/./dwg_harness.cpp:65:9 #12 0x4587e1 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x4587e1) #13 0x443f52 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long) (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x443f52) #14 0x449a06 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long)) (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x449a06) #15 0x4726c2 in main (/root/boop/assorted_fuzzing/librecad/fuzzing/dwg/triaged/dwg_fuzzer.bin+0x4726c2) #16 0x7f6d833c10b2 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:208:26 in dwgCompressor::decompress18(unsigned char, unsigned char, unsigned int, unsigned int) Shadow bytes around the buggy address: 0x0c587fff8e70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0x0c587fff8e80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0x0c587fff8e90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0x0c587fff8ea0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0x0c587fff8eb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 =>0x0c587fff8ec0:[fa]fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa 0x0c587fff8ed0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa 0x0c587fff8ee0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa 0x0c587fff8ef0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa 0x0c587fff8f00: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa 0x0c587fff8f10: 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 ==1043701==ABORTING
```

TIMELINE

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

CREDIT

Discovered by Lilith >_> of Cisco Talos.