Talos Vulnerability Report

TALOS-2020-1223

Openscad import_stl.cc:import_stl() stack-based buffer overflow vulnerability

FEBRUARY 23, 2021

CVE NUMBER

CVE-2020-28599

Summary

A stack-based buffer overflow vulnerability exists in the import_stl.cc:import_stl() functionality of Openscad openscad-2020.12-RC2. A specially crafted STL file can lead to code execution. An attacker can provide a malicious file to trigger this vulnerability.

Tested Versions

Openscad openscad-2020.12-RC2

Product URLs

https://github.com/openscad/openscad

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-121 - Stack-based Buffer Overflow

Details

Openscad is an open-source program for creating 3-D CAD models, available for all platforms. Aside from describing and creating objects from scripts, it's also possible to import existing .stl, .amf, .3mf, .svg and .dxf files into a scene for rendering.

When importing a given `.stl` file into a scene via the `import("file.stl");` command, the first stl-specific function we hit is `PolySet *import_stl(const std::string &filename, const Location &loc)`:

```
PolySet *import_stl(const std::string &filename, const Location &loc)
{
    PolySet *p = new PolySet(3);

    // Open file and position at the end
    std::ifstream f(filename.c_str(), std::ios::in | std::ios::binary | std::ios::ate);  // [1]
    if (!f.good()) {
        LOG(message_group::Warning,Location::NONE,"","Can't open import file '%1$s', import() at line %2$d",filename,loc.firstLine());
        return p;
    }

    boost::regex ex_sfe("solid|facet|endloop");   // [2]
    boost::regex ex_outer("outer loop");
    boost::regex ex_vertex("vertex");
    boost::regex ex_vertices("\\s*vertex\\s+([^\\s]+)\\s+([^\\s]+)\\s+([^\\s]+)");  // [3]

    bool binary = false;
    std::streampos file_size = f.tellg();
    f.seekg(80);
    if (f.good() && !f.eof()) {                   // [4]
        uint32_t facenum = 0;
        f.read((char *)&facenum, sizeof(uint32_t));
#if BOOST_ENDIAN_BIG_BYTE
        uint32_byte_swap( facenum );
#endif
        if (file_size ==  static_cast<std::streamoff>(80 + 4 + 50*facenum)) {
            binary = true;
        }
    }
```

At [1], our input file is opened, and at [2] through [3] we notice some important regexes that will be used further on. Assuming we pass the check at [4], which makes sure our file is at least 80 bytes, then we move on to the following code:

```
PolySet *import_stl(const std::string &filename, const Location &loc)
{
    // [...]
    char data[5];
    f.read(data, 5);
    if (!binary && !f.eof() && f.good() && !memcmp(data, "solid", 5)) {
        int i = 0;
        double vdata[3][3];      // [1]
        std::string line;
        std::getline(f, line);
        while (!f.eof()) {       // [2]
            std::getline(f, line);
            boost::trim(line);
            if (boost::regex_search(line, ex_sfe)) {    // "solid|facet|endloop" // [3]
                continue;
            }
            if (boost::regex_search(line, ex_outer)) {  // "outer loop"          // [4]
                i = 0;
                continue;
            }
            boost::smatch results;
            if (boost::regex_search(line, results, ex_vertices)) {  // "\\s*vertex\\s+([^\\s]+)\\s+([^\\s]+)\\s+([^\\s]+)" // [5]
                //[...]
            }
        }
    }
```

At [2] we hit our parsing loop, iterating over each line of the input `.stl` file, looking for different regexes as we go along. Lines matching the regex at [3], `"solid|facet|endloop"`, are completely ignored, lines matching at [4], `"outerloop"`, reset the `i` variable, but that's about it. The only regex that is actually read in is at [5], `"\\s*vertex\\s+([^\\s]+)\\s+([^\\s]+)\\s+([^\\s]+)"`. To give an example:

```
  facet normal  1.000000e+00  0.000000e+00 -0.000000e+00
    outer loop
      vertex   2.000000e+01  2.000000e+01  0.000000e+00
      vertex   2.000000e+01  2.000000e+01  2.000000e+01
      vertex   2.000000e+01  0.000000e+00  2.000000e+01
    endloop
  endfacet
```

To proceed, let us now examine the code hit when the `ex_vertices` regex is hit:

```
if (boost::regex_search(line, results, ex_vertices)) {  // "\\s*vertex\\s+([^\\s]+)\\s+([^\\s]+)\\s+([^\\s]+)"
    try {
        for (int v=0; v<3; ++v) {
            vdata[i][v] = boost::lexical_cast<double>(results[v+1]); // [1]
        }
    }
    catch (const boost::bad_lexical_cast &blc) {
        LOG(message_group::Warning,Location::NONE,"","Can't parse vertex line '%1$s', import() at line %2$d",line,loc.firstLine());
        i = 10;
        continue;
    }
    if (++i == 3) { // [2]
        p->append_poly();
        p->append_vertex(vdata[0][0], vdata[0][1], vdata[0][2]);
        p->append_vertex(vdata[1][0], vdata[1][1], vdata[1][2]);
        p->append_vertex(vdata[2][0], vdata[2][1], vdata[2][2]);
    }
}
```

Each of the vertex numbers are populated into the `vdata` variable at [1], and if we have three vertexes read in (forming a triangle) at [2], we append these vertexes into the `PolySet *p` object. Interestingly, the only thing that resets the `i` variable is, as mentioned before, when we hit the `"outer loop"` regex:

```
    if (boost::regex_search(line, ex_outer)) {  // "outer loop"
        i = 0;
        continue;
    }
```

Thus, if our .stl file has more than three vertexes in a given `outer loop` tag, `i` keeps incrementing, and if we look again at `double vdata[3][3];`, we quickly realize that this is an arbitrary stack-based buffer overflow, resulting in potential code execution.

Crash Information

```
==2559056==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffee6cc0ec8 at pc 0x7f2417b8938f bp 0x7ffee6cc0a30 sp 0x7ffee6cc0a28
WRITE of size 8 at 0x7ffee6cc0ec8 thread T0
    #0 0x7f2417b8938e in import_stl(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, Location const&)
//boop/assorted_fuzzing/openscad/openscad-openscad-2020.12-RC2/openscad-openscad-2020.12-RC2/src/import_stl.cc:114:19
    #1 0x55bb6c in LLVMFuzzerTestOneInput //boop/assorted_fuzzing/openscad/openscad-openscad-2020.12-RC2/./fuzz_stl_harness.cpp:71:21
    #2 0x461ae1 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) (//boop/assorted_fuzzing/openscad/openscad-openscad-
2020.12-RC2/stl_fuzzdir/stl_harness.bin+0x461ae1)
    #3 0x44d252 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long) (//boop/assorted_fuzzing/openscad/openscad-openscad-
2020.12-RC2/stl_fuzzdir/stl_harness.bin+0x44d252)
    #4 0x452d06 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long))
(//boop/assorted_fuzzing/openscad/openscad-openscad-2020.12-RC2/stl_fuzzdir/stl_harness.bin+0x452d06)
    #5 0x47b9c2 in main (//boop/assorted_fuzzing/openscad/openscad-openscad-2020.12-RC2/stl_fuzzdir/stl_harness.bin+0x47b9c2)
    #6 0x7f24159380b2 in __libc_start_main /build/glibc-ZN95T4/glibc-2.31/csu/../csu/libc-start.c:308:16
    #7 0x42791d in _start (//boop/assorted_fuzzing/openscad/openscad-openscad-2020.12-RC2/stl_fuzzdir/stl_harness.bin+0x42791d)

Address 0x7ffee6cc0ec8 is located in stack of thread T0 at offset 1160 in frame
    #0 0x7f2417b87daf in import_stl(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, Location const&)
//boop/assorted_fuzzing/openscad/openscad-openscad-2020.12-RC2/openscad-openscad-2020.12-RC2/src/import_stl.cc:63

  This frame has 25 object(s):
    [32, 36) 'agg.tmp'
    [48, 568) 'f' (line 67)
    [704, 708) 'ref.tmp' (line 69)
    [720, 752) 'ref.tmp7' (line 69)
    [784, 785) 'ref.tmp8' (line 69)
    [800, 804) 'ref.tmp11' (line 69)
    [816, 832) 'ex_sfe' (line 73)
    [848, 864) 'ex_outer' (line 74)
    [880, 896) 'ex_vertex' (line 75)
    [912, 928) 'ex_vertices' (line 76)
    [944, 960) 'file_size' (line 79)
    [976, 992) 'agg.tmp30'
    [1008, 1012) 'facenum' (line 82)
    [1024, 1040) 'agg.tmp56'
    [1056, 1061) 'data' (line 93)
    [1088, 1160) 'vdata' (line 97) <== Memory access at offset 1160 overflows this variable
    [1200, 1232) 'line' (line 98)
    [1264, 1272) 'ref.tmp93' (line 102)
    [1296, 1376) 'results' (line 110)
    [1408, 1409) 'ref.tmp106' (line 110)
    [1424, 1428) 'ref.tmp125' (line 118)
    [1440, 1472) 'ref.tmp126' (line 118)
    [1504, 1505) 'ref.tmp127' (line 118)
    [1520, 1524) 'ref.tmp130' (line 118)
    [1536, 1588) 'facet' (line 135)
HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow //boop/assorted_fuzzing/openscad/openscad-openscad-2020.12-RC2/openscad-openscad-2020.12-
RC2/src/import_stl.cc:114:19 in import_stl(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, Location
const&)
Shadow bytes around the buggy address:
  0x10005cd90180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 f2
  0x10005cd90190: f2 f2 f2 f2 f2 f2 f2 f2 f2 f2 f2 f2 f2 f2 f2 f2
  0x10005cd901a0: f8 f2 f8 f8 f8 f8 f2 f2 f2 f2 f8 f2 f8 f2 00 00
  0x10005cd901b0: f2 f2 00 00 f2 f2 00 00 f2 f2 00 00 f2 f2 00 00
  0x10005cd901c0: f2 f2 00 00 f2 f2 f8 f2 00 00 f2 f2 05 f2 f2 f2
=>0x10005cd901d0: 00 00 00 00 00 00 00 00 00 00[f2]f2 f2 f2 f2 00 00
  0x10005cd901e0: 00 00 f2 f2 f2 f2 f8 f2 f2 f2 00 00 00 00 00 00
  0x10005cd901f0: 00 00 00 00 f2 f2 f2 f2 f8 f2 f8 f2 f8 f8 f8 f8
  0x10005cd90200: f2 f2 f2 f2 f8 f2 f8 f2 f8 f8 f8 f8 f8 f8 f8 f3
  0x10005cd90210: f3 f3 f3 f3 00 00 00 00 00 00 00 00 00 00 00 00
  0x10005cd90220: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
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
```

Timeline

2021-01-08 - Vendor Disclosure

2021-01-31 - Vendor Patched

2021-02-23 - Public Release

CREDIT

Discovered by Lilith >_> of Cisco Talos.