Talos Vulnerability Report

TALOS-2020-1224

Openscad import_stl.cc:import_stl() out-of-bounds stack write vulnerability

FEBRUARY 23, 2021

CVE NUMBER

CVE-2020-28600

Summary

An out-of-bounds write 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-119 - Improper Restriction of Operations within the Bounds of a Memory Buffer

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) { // [2]
        LOG(message_group::Warning,Location::NONE,"","Can't parse vertex line '%1$s', import() at line %2$d",line,loc.firstLine());
        i = 10;     // [3]
        continue;
    }
    if (++i == 3) { // [4]
        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 [4], we append these vertexes into the `PolySet *p` object.

A curious thing happens though if we have a given vertex co-ordinate that raises an error from `boost::lexical_cast<double>` [2], the `i` variable is set to 10 at [3]. Thus, once we hit the next line in the file that hits the `ex_vertices` regex, the line at [1] will write a user controlled value to `vdata[10][v]`, the consequences thereof depending on the compiler.

At least for our testing build, this overwrote a pointer inside a `boost::shared_ptr`, allowing us to control exactly what happened during the `boost::shared_ptr`'s destructor.

Crash Information

```
********************************************************************************
rax        : 0x4034000000000034                | r13[S]     : 0x7ffd8634d370
rbx[S]     : 0x7ffd8634d600                    | r14[S]     : 0x7ffd8634d310
rcx        : 0x4034000000000008                | r15[S]     : 0x7ffd8634d330
rdx        : 0x1                               | rip[L]     : 0x7f6aaebad51f
rsi        : 0x806800000000001                 | eflags     : 0x10202
rdi        : 0x4034000000000008                | cs         : 0x33
rbp[S]     : 0x7ffd8634ceb0                    | ss         : 0x2b
rsp[S]     : 0x7ffd8634ce90                    | ds         : 0x0
r8         : 0x5c4000                          | es         : 0x0
r9         : 0x21                              | fs         : 0x0
r10        : 0x7ffd85b50000                    | gs         : 0x0
r11        : 0xffffff01                        | fs_base    : 0x7f6aa7583e00
r12[S]     : 0x7ffd8634d350                    | gs_base    : 0x0
********************************************************************************
0x7f6aaebad50e : mov    DWORD PTR [rbp-0xc],0x1
0x7f6aaebad515 : mov    edx,DWORD PTR [rbp-0xc]
0x7f6aaebad518 : mov    rsi,rcx
0x7f6aaebad51b : shr    rsi,0x3
=>0x7f6aaebad51f : mov    al,BYTE PTR [rsi+0x7fff8000]
0x7f6aaebad525 : cmp    al,0x0
0x7f6aaebad527 : mov    QWORD PTR [rbp-0x18],rcx
0x7f6aaebad52b : mov    DWORD PTR [rbp-0x1c],edx
0x7f6aaebad52e : mov    BYTE PTR [rbp-0x1d],al
0x7f6aaebad531 : je     0x7f6aaebad560 <boost::detail::atomic_decrement(int _Atomic*)+112>
********************************************************************************
#0  0x00007f6aaebad51f in boost::detail::atomic_decrement(int _Atomic*) (pw=0x4034000000000008) at/sp_counted_base_clang.hpp:38
#1  0x00007f6aaebad426 in boost::detail::sp_counted_base::release (this=0x4034000000000000) at/sp_counted_base_clang.hpp:118
#2  0x00007f6aaebacc7a in boost::detail::shared_count::~shared_count (this=0x7ffd8634d4f0) at/shared_count.hpp:427
#3  0x00007f6aaee27a8f in boost::shared_ptr<boost::re_detail_107100::named_subexpressions>::~shared_ptr (this=0x7ffd8634d4e8)
at/shared_ptr.hpp:341
#4  0x00007f6aaed88db6 in boost::match_results<__gnu_cxx::__normal_iterator<char const*, std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char> > >, std::allocator<boost::sub_match<__gnu_cxx::__normal_iterator<char const*,
std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > > > >::~match_results (this=0x7ffd8634d4b0)
at/match_results.hpp:119
#5  0x00007f6aaf1c7aa2 in import_stl (filename= at/import_stl.cc:129
#6  0x000000000055bb6d in LLVMFuzzerTestOneInput (Data=0x61f000000e80 "solid STL generated by MeshLab\n  facet normal  0.000000e+00
-0.000000e+00 -1.000000e+00\n    outer loop\n      vertex   2.000000e+01  2.000000e+01  0.000000e+00\n      vertex   2.000000e+01
0.000000e+00"..., Size=3342) at/fuzz_stl_harness.cpp:71
#7  0x0000000000461ae2 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) () at/optional.hpp:99
#8  0x000000000044d253 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long) () at/optional.hpp:99
#9  0x0000000000452d07 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long)) () at/optional.hpp:99
#10 0x000000000047b9c3 in main () at/optional.hpp:99
********************************************************************************
[^_^] 2020_12_16_12_42_26_078463 - Got a crash! SIGSEGV, 0x7f6aaebad51f
```

Timeline

2021-01-11 - Vendor Disclosure

2021-02-23 - Public Release

CREDIT

Discovered by Lilith >_> of Cisco Talos.
