Talos Vulnerability Report

TALOS-2020-1213

# Slic3r libslic3r Obj File TriangleMesh::TriangleMesh() out-of-bounds read vulnerability

FEBRUARY 24, 2021

### CVE NUMBER

CVE-2020-28590

### Summary

An out-of-bounds read vulnerability exists in the Obj File TriangleMesh::TriangleMesh() functionality of Slic3r libslic3r 1.3.0 and Master Commit 92abbc42. A specially crafted obj file could lead to information disclosure. An attacker can provide a malicious file to trigger this vulnerability.

### Tested Versions

Slic3r libslic3r 1.3.0
Slic3r libslic3r Master Commit 92abbc42

### Product URLs

http://slic3r.org

### CVSSv3 Score

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

### CWE

CWE-20 - Improper Input Validation

### Details

Slic3r is an open-source 3-D printing toolbox, mainly utilized for translating assorted 3-D printing model file types into machine code for a specific printer. Slic3r uses libslic3er to do most of the non-GUI-based heavy lifting like reading various file formats, converting formats and outputting appropriate gcode for selected 3-D printer settings.

When reading in a `.obj` file for the purposes of conversion or display, libslic3r uses tiny_object_loader.h to load a given filestream and turn it into `shape_t` and `material_t` object vectors for further processing. For each `shape_t` read in, a set of vertices are created, and then from these vertices, a set of facets. If any given facet has more than three vertices, the facet is split up into triangles, and then a `TriangleMesh` object is created from all of these triangles. This process can be seen in code below:

```
// Loop over shapes and add a volume for each one.
for (std::vector<tinyobj::shape_t>::const_iterator shape = shapes.begin();
    shape != shapes.end(); ++shape) {

    Pointf3s points;
    std::vector<Point3> facets;

    // Read vertices.
    assert((attrib.vertices.size() % 3) == 0);
    for (size_t v = 0; v < attrib.vertices.size(); v += 3) {
        points.push_back(Pointf3(
            attrib.vertices[v],
            attrib.vertices[v+1],
            attrib.vertices[v+2]
        ));
    }

    // Loop over facets of the current shape.
    for (size_t f = 0; f < shape->mesh.num_face_vertices.size(); ++f) {
        // tiny_obj_loader should triangulate any facet with more than 3 vertices
        assert((shape->mesh.num_face_vertices[f] % 3) == 0);

        facets.push_back(Point3(
            shape->mesh.indices[f*3+0].vertex_index,
            shape->mesh.indices[f*3+1].vertex_index,
            shape->mesh.indices[f*3+2].vertex_index
        ));
    }

    TriangleMesh mesh(points, facets);              // [1]
    mesh.check_topology();
    ModelVolume* volume = object->add_volume(mesh);
    volume->name        = object->name;
}
```

For our purposes, we only really care about this resultant `TriangleMesh` object, and so we investigate the constructor function `TriangleMesh::TriangleMesh(const Pointf3* points, const Point3* facets, size_t n_facets) : repaired(false)`:

```
TriangleMesh::TriangleMesh(const Pointf3* points, const Point3* facets, size_t n_facets)
    : repaired(false) {
stl_initialize(&this->stl);
stl_file &stl = this->stl;
stl.error = 0;
stl.stats.type = inmemory;

// count facets and allocate memory
stl.stats.number_of_facets = n_facets;
stl.stats.original_num_facets = stl.stats.number_of_facets;
stl_allocate(&stl);

for (int i = 0; i < stl.stats.number_of_facets; i++) {   // [1]
    stl_facet facet;
    facet.normal.x = 0;
    facet.normal.y = 0;
    facet.normal.z = 0;

    const Pointf3& ref_f1 = points[facets[i].x];         //  [2]
    facet.vertex[0].x = ref_f1.x;
    facet.vertex[0].y = ref_f1.y;
    facet.vertex[0].z = ref_f1.z;

    const Pointf3& ref_f2 = points[facets[i].y];         //  [3]
    facet.vertex[1].x = ref_f2.x;
    facet.vertex[1].y = ref_f2.y;
    facet.vertex[1].z = ref_f2.z;

    const Pointf3& ref_f3 = points[facets[i].z];         //  [4]
    facet.vertex[2].x = ref_f3.x;
    facet.vertex[2].y = ref_f3.y;
    facet.vertex[2].z = ref_f3.z;

    facet.extra[0] = 0;
    facet.extra[1] = 0;

    stl.facet_start[i] = facet;
}
```

At [1] we start our loop upon all the facets, and then at [2], [3], and [4] we start gathering all the co-ordinates for our current triangle. We now provide example objects to clarify this process:

```
[^~^]> p/x points
$2 = std::vector of length 22, capacity 32 =
{{<Slic3r::Pointf> = {x = 0x0, y = 0x2}, z = 0x2},
{<Slic3r::Pointf> = {x = 0x0, y = 0x0}, z = 0x0},
{<Slic3r::Pointf> = {x = 0x2, y = 0x0}, z = 0x0},
[...]

[~.~]> p/x facets
$3 = std::vector of length 8, capacity 8 = {
{<Slic3r::Point> = {x = 0x0, y = 0x1}, z = 0x2},
{<Slic3r::Point> = {x = 0x0, y = 0x2}, z = 0x3},
{<Slic3r::Point> = {x = 0x5, y = 0x6}, z = 0x7},
[...]
```

Unfortunately it should be emphasized that all these co-ordinates for both the `facet` and `points` variables come directly from the input stream (i.e. the file input), and there’s no input validation on these values. Thus it becomes possible to have a facet that might look like: `{<Slic3r::Point> = {x = 0xd, y = 0xf}, z = 0xffffffffffe17b91}`, and so when we read ` const Pointf3& ref_f3 = points[facets[i].z]; it quite naturally results in a quite powerful read-what-where vulnerability, as this would result in points[0xffffffffffe17b91]`.

Utilizing this data would require further work, but since an attacker could theoretically read as much memory as they wanted, anywhere they wanted, and since this code is used as part of a library, this issue could result in an information disclosure.

### Crash Information

```
=================================================================
==637998==ERROR: AddressSanitizer: SEGV on unknown address 0x616ffd239d18 (pc 0x7f8e7b980b8d bp 0x7fffbdad04f0 sp 0x7fffbdad0180 T0)
==637998==The signal is caused by a READ memory access.

***********************************************************************************
rax        : 0xc2dffa473a3                      | r13        : 0xd0
rbx        : 0x0                                | r14        : 0xffffffffffa472b3
rcx        : 0x3fd7b125c3f2c0                   | r15        : 0x616ffd239d18
rdx        : 0x1e50                             | rip[L]     : 0x7faf624b8860
rsi        : 0x0                                | eflags     : 0x10246
rdi        : 0x1                                | cs         : 0x33
rbp[S]     : 0x7ffd5a5ccdb0                     | ss         : 0x2b
rsp[S]     : 0x7ffd5a5cccf0                     | ds         : 0x0
r8         : 0x5c9000                           | es         : 0x0
r9         : 0x3fd7b125c3f280                   | fs         : 0x0
r10        : 0x50                               | gs         : 0x0
r11        : 0x522c01                           | fs_base    : 0x7faf5ef67c40
r12        : 0x1                                | gs_base    : 0x0
***********************************************************************************
0x7faf624b884c : mov    rax,r15
0x7faf624b884f : shr    rax,0x3
0x7faf624b8853 : cmp    BYTE PTR [rax+0x7fff8000],0x0
0x7faf624b885a : jne    0x7faf624b930a <Slic3r::TriangleMesh::TriangleMesh(Slic3r::Pointf3 const*, Slic3r::Point3 const*, unsigned
long)+4634>
=>0x7faf624b8860 : movsd  xmm0,QWORD PTR [r15]
0x7faf624b8865 : movsd  QWORD PTR [rbp-0x60],xmm0
0x7faf624b886a : test   rbx,rbx
0x7faf624b886d : jne    0x7faf624b8f86 <Slic3r::TriangleMesh::TriangleMesh(Slic3r::Pointf3 const*, Slic3r::Point3 const*, unsigned
long)+3734>
0x7faf624b8873 : add    BYTE PTR [rip+0x12e04b4],0x1        # 0x7faf63798d2e
0x7faf624b887a : test   r12b,r12b
***********************************************************************************
#0  0x00007faf624b8860 in Slic3r::TriangleMesh::TriangleMesh (this=?, points=?, facets=?, n_facets=?) at/TriangleMesh.cpp:61
#1  0x00007faf62378d84 in Slic3r::IO::OBJ::read (input_file=..., model=?) at/IO.cpp:146
#2  0x0000000000556ded in LLVMFuzzerTestOneInput (Data=0x618000000480 "v 0.000000 2.000000 2.000000\nv 0000000 0.000000 0.000000\nv
2.000000000 0.000000\n\200 0.000000 0.000000 2.000000\nv 0.000000 2.000000 2.000000\nf -4 -3 -2 -1\n\nv 0.000000 0.000000 2.000001\nv 0.0000
00 0.00000"..., Size=785) at/fuzz_obj_harness.cpp:81
#3  0x000000000045d012 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) () at/exception_ptr.hpp:145
#4  0x0000000000448783 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long) () at/exception_ptr.hpp:145
#5  0x000000000044e237 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long)) () at/exception_ptr.hpp:145
#6  0x0000000000476ef3 in main () at/exception_ptr.hpp:145
***********************************************************************************
```

### Timeline

2020-12-21 - Vendor Disclosure

2021-02-21 2021-02-24 - Public Release

### CREDIT

Discovered by Lilith >_> of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT

TALOS-2020-1167

NEXT REPORT

TALOS-2020-1225