Talos Vulnerability Report

TALOS-2020-1219

Prusa Research PrusaSlicer Obj.cpp load_obj() out-of-bounds write vulnerability

APRIL 21, 2021

CVE NUMBER

CVE-2020-28595

Summary

An out-of-bounds write vulnerability exists in the Obj.cpp load_obj() functionality of Prusa Research PrusaSlicer 2.2.0 and Master (commit 4b040b856). A specially crafted obj file can lead to code execution. An attacker can provide a malicious file to trigger this vulnerability.

Tested Versions

Prusa Research PrusaSlicer 2.2.0
Prusa Research PrusaSlicer Master (commit 4b040b856)

Product URLs

https://www.prusa3d.com/prusaslicer/

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-122 - Heap-based Buffer Overflow

Details

Prusa Slicer is an open-source 3-D printer slicing program forked off Slic3r that can convert various 3-D model file formats and can output corresponding 3-D printer-readable Gcode.

One of the input file formats PrusaSlicer can deal with is `.obj` files, the code mainly handling this can be found in `PrusaSlicer/src/libslic3r/Format/OBJ.cpp` and `PrusaSlicer/src/libslic3r/Format/objparser.cpp`.
We now proceed to trace the winding code-path from entry to vulnerability, starting with `load_obj(const char *path, TriangleMesh *meshptr)`.

```
bool load_obj(const char *path, TriangleMesh *meshptr){
    if(meshptr == nullptr) return false;

    // Parse the OBJ file.
    ObjParser::ObjData data;
    if (! ObjParser::objparse(path, data)) {   // [1]
        //    die "Failed to parse $file\n" if !-e $path;
        return false;
    }
```

At [1], our given `.obj` file is parsed and populated into the ObjData structure, which looks like such:

```
struct ObjData {
    // Version of the data structure for load / store in the private binary format.
    int                             version;

    // x, y, z, w
    std::vector<float>              coordinates;
    // u, v, w
    std::vector<float>              textureCoordinates;
    // x, y, z
    std::vector<float>              normals;
    // u, v, w
    std::vector<float>              parameters;

    std::vector<std::string>        mtllibs;
    std::vector<ObjUseMtl>          usemtls;
    std::vector<ObjObject>          objects;
    std::vector<ObjGroup>           groups;
    std::vector<ObjSmoothingGroup>  smoothingGroups;

    // List of faces, delimited by an ObjVertex with all members set to -1.
    std::vector<ObjVertex>          vertices;
};
```

Assuming things are going as intended, all of the data from our input file gets populated with data from our `.obj` file, for example, the following `.obj` file segment would populate two floats and two vertices:

```
f 434//434 488//488 563//563
f 451//451 435//435 436//436
v 55.986176 26.094831 28.978714
v -0.024765 -2.100579 -0.128764
```

Continuing on into `load_obj(const char *path, TriangleMesh *meshptr)`, from where we left off:

```
// Count the faces and verify, that all faces are triangular.
size_t num_faces = 0;
size_t num_quads = 0;
for (size_t i = 0; i < data.vertices.size(); ) {  // [1]
    size_t j = i;
    for (; j < data.vertices.size() && data.vertices[j].coordIdx != -1; ++ j) ;
    if (i == j)
        continue;
    size_t face_vertices = j - i;
    if (face_vertices != 3 && face_vertices != 4) {
        // Non-triangular and non-quad faces are not supported as of now.
        return false;
    }
    if (face_vertices == 4)
        ++ num_quads;   // [2]
    ++ num_faces;       // [3]
    i = j + 1;
}

// Convert ObjData into STL.
TriangleMesh &mesh = *meshptr;
stl_file &stl = mesh.stl;
stl.stats.type = inmemory;
stl.stats.number_of_facets = uint32_t(num_faces + num_quads); // [4]
stl.stats.original_num_facets = int(num_faces + num_quads);
// stl_allocate clears all the allocated data to zero, all normals are set to zeros as well.
stl_allocate(&stl);
```

The most important things to note here are that: [1], we enter a loop whose iteration count is determined by the amount of vertices in our `.obj` file. For each iteration, the `num_quads` variable [2] might go up, and the `num_faces` variable [3] always goes up. At [4], these variables are stored together in the `stl.stats.number_of_facets` variable, which is further used inside `stl_allocate` [5] to determine how many `stl_facet` objects get stored into `stl.facet_start`:

```
void stl_allocate(stl_file *stl)
{
    //  Allocate memory for the entire .STL file.
    stl->facet_start.assign(stl->stats.number_of_facets, stl_facet());
    // Allocate memory for the neighbors list.
    stl->neighbors_start.assign(stl->stats.number_of_facets, stl_neighbors());
}
```

Thus, the amount of memory allocated is `sizeof(stl_facet) * (num_faces+num_quads)`, and if we desire further detail we can look at what an `stl_facet` object is:

```
[x.x]> p/x sizeof(stl_facet)
$11 = 0x34

[^~^]> ptype stl_facet
type = struct stl_facet {
    stl_normal normal;
    stl_vertex vertex[3];
    char extra[2];
  public:
    stl_facet rotated(const Eigen::Quaternion<float, 2> &) const;
}
```

To proceed, let us again continue on inside the `load_obj(const char *path, TriangleMesh *meshptr)` function:

```
stl_allocate(&stl);
size_t i_face = 0;
for (size_t i = 0; i < data.vertices.size(); ++ i) {              // [1]
    if (data.vertices[i].coordIdx == -1)
        continue;
    stl_facet &facet = stl.facet_start[i_face ++];                // [2]
    size_t     num_normals = 0;
    stl_normal normal(stl_normal::Zero());
    for (unsigned int v = 0; v < 3; ++ v) {
        const ObjParser::ObjVertex &vertex = data.vertices[i++];  // [3]
            // [...]
        }
    }
    if (data.vertices[i].coordIdx != -1) {                        // [4]
        // This is a quad. Produce the other triangle.
        stl_facet &facet2 = stl.facet_start[i_face++];            // [5]
        facet2.vertex[0] = facet.vertex[0];                       // [6]
        facet2.vertex[1] = facet.vertex[2];
        const ObjParser::ObjVertex &vertex = data.vertices[i++];
            // [...]
        }
```

At [1], we again have a loop whose iteration count is determined by the amount of vertices, and at [2] we can see the `i_face` variable being used to index into the previously allocated `stl.facet_start` array and, more importantly, `i_face` is incremented. [3] is only important because it can increment `i`, which lets us enter the branch at [4]. Thus it's possible to hit the `i_face++` at [2] and the `i_face++` [5] within a single loop of [1].

Recall that the size of `stl.facet_start` is `sizeof(stl_facet) * (num_faces+num_quads)`, and also that `num_faces+num_quads` is at least equal to `data.vertices.size()`, since it can either go up by one or two for each fore mentioned loop iteration. Contrast this with the `i_face` variable, which can go up by two in a single loop, and we can quickly see that it's possible for `i_face` to exceed the amount of actual `stl_facet` objects within `stl.facet_start`, resulting in an out-of-bounds write on the heap.
Dedicated readers might notice and protest that the increment of `i` at [3] can potentially cause `i` to go out of bounds, and those dedicated readers would be decidedly correct, this is in fact a potential out-of-bounds read that allows us to trigger the out-of-bounds write.

To explain another way, PrusaSlicer expects every `stl_facet` to contain 4 vertices entries. If it's a triangle, it still has 4 vertices, it's just that the last `vertices.coordIdx` == -1, whereas a square's last `.coordIdx` != -1. If a given .obj file contains `x` vertices such that `x % 4 == 1` or `x % 4 == 2` then the file is rejected:

```
for (size_t i = 0; i < data.vertices.size(); ) {
    size_t j = i;
    for (; j < data.vertices.size() && data.vertices[j].coordIdx != -1; ++ j) ;
    if (i == j)
        continue;
    size_t face_vertices = j - i;
    if (face_vertices != 3 && face_vertices != 4) {  // rejected here
        // Non-triangular and non-quad faces are not supported as of now.
        return false;
    }
```

But, if `x % 4 == 3`, then we pass the above sanity check, and hit an oob-read that's looking for the fourth vertices in a given facet which can then trigger an oob-write as mentioned before.

Crash Information

```
=================================================================
==897228==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x606000010860 at pc 0x000000579ce2 bp 0x7ffdd8b355f0 sp 0x7ffdd8b355e8
WRITE of size 4 at 0x606000010860 thread T0
    #0 0x579ce1 in Eigen::internal::assign_op<float, float>::assignCoeff(float&, float const&) const
/boop/assorted_fuzzing/prusaslicer/./PrusaSlicer/src/eigen/Eigen/src/Core/functors/AssignmentFunctors.h:24:102
    #1 0x7f65faed2d8e in Eigen::internal::generic_dense_assignment_kernel<Eigen::internal::evaluator<Eigen::Matrix<float, 3, 1, 2, 3, 1> >,
Eigen::internal::evaluator<Eigen::Matrix<float, 3, 1, 2, 3, 1> >, Eigen::internal::assign_op<float, float>, 0>::assignCoeff(long, long)
/usr/include/eigen3/Eigen/src
/Core/AssignEvaluator.h:631:15
    #2 0x7f65faed2ba4 in Eigen::internal::generic_dense_assignment_kernel<Eigen::internal::evaluator<Eigen::Matrix<float, 3, 1, 2, 3, 1> >,
Eigen::internal::evaluator<Eigen::Matrix<float, 3, 1, 2, 3, 1> >, Eigen::internal::assign_op<float, float>,
0>::assignCoeffByOuterInner(long, long) /usr/include/eige
n3/Eigen/src/Core/AssignEvaluator.h:645:5
    #3 0x7f65faed2aee in
Eigen::internal::copy_using_evaluator_DefaultTraversal_CompleteUnrolling<Eigen::internal::generic_dense_assignment_kernel<Eigen::internal::e
valuator<Eigen::Matrix<float, 3, 1, 2, 3, 1> >, Eigen::internal::evaluator<Eigen::Matrix<float, 3, 1, 2, 3, 1> >, Eigen::internal::assign_op
<float, float>, 0>, 0, 3>::run(Eigen::internal::generic_dense_assignment_kernel<Eigen::internal::evaluator<Eigen::Matrix<float, 3, 1, 2, 3,
1> >, Eigen::internal::evaluator<Eigen::Matrix<float, 3, 1, 2, 3, 1> >, Eigen::internal::assign_op<float, float>, 0>&)
/usr/include/eigen3/Eigen/src/Core/AssignEvalu
ator.h:206:12
    #4 0x7f65faed2a5d in
Eigen::internal::dense_assignment_loop<Eigen::internal::generic_dense_assignment_kernel<Eigen::internal::evaluator<Eigen::Matrix<float, 3,
1, 2, 3, 1> >, Eigen::internal::evaluator<Eigen::Matrix<float, 3, 1, 2, 3, 1> >, Eigen::internal::assign_op<float, float>, 0>, 3,
2>::run(Eig
en::internal::generic_dense_assignment_kernel<Eigen::internal::evaluator<Eigen::Matrix<float, 3, 1, 2, 3, 1> >,
Eigen::internal::evaluator<Eigen::Matrix<float, 3, 1, 2, 3, 1> >, Eigen::internal::assign_op<float, float>, 0>&)
/usr/include/eigen3/Eigen/src/Core/AssignEvaluator.h:435:5
    #5 0x7f65faed25d3 in void Eigen::internal::call_dense_assignment_loop<Eigen::Matrix<float, 3, 1, 2, 3, 1>, Eigen::Matrix<float, 3, 1, 2,
3, 1>, Eigen::internal::assign_op<float, float> >(Eigen::Matrix<float, 3, 1, 2, 3, 1>&, Eigen::Matrix<float, 3, 1, 2, 3, 1> const&,
Eigen::internal::assign_op<float
, float> const&) /usr/include/eigen3/Eigen/src/Core/AssignEvaluator.h:741:3
    #6 0x7f65faed23a4 in Eigen::internal::Assignment<Eigen::Matrix<float, 3, 1, 2, 3, 1>, Eigen::Matrix<float, 3, 1, 2, 3, 1>,
Eigen::internal::assign_op<float, float>, Eigen::internal::Dense2Dense, void>::run(Eigen::Matrix<float, 3, 1, 2, 3, 1>&,
Eigen::Matrix<float, 3, 1, 2, 3, 1> const&, Eigen::intern
al::assign_op<float, float> const&) /usr/include/eigen3/Eigen/src/Core/AssignEvaluator.h:879:5
    #7 0x7f65faed231c in void Eigen::internal::call_assignment_no_alias<Eigen::Matrix<float, 3, 1, 2, 3, 1>, Eigen::Matrix<float, 3, 1, 2,
3, 1>, Eigen::internal::assign_op<float, float> >(Eigen::Matrix<float, 3, 1, 2, 3, 1>&, Eigen::Matrix<float, 3, 1, 2, 3, 1> const&,
Eigen::internal::assign_op<float,
float> const&) /usr/include/eigen3/Eigen/src/Core/AssignEvaluator.h:836:3
    #8 0x7f65faed2290 in void Eigen::internal::call_assignment<Eigen::Matrix<float, 3, 1, 2, 3, 1>, Eigen::Matrix<float, 3, 1, 2, 3, 1>,
Eigen::internal::assign_op<float, float> >(Eigen::Matrix<float, 3, 1, 2, 3, 1>&, Eigen::Matrix<float, 3, 1, 2, 3, 1> const&,
Eigen::internal::assign_op<float, float> co
nst&, Eigen::internal::enable_if<!(evaluator_assume_aliasing<Eigen::Matrix<float, 3, 1, 2, 3, 1> >::value), void*>::type)
/usr/include/eigen3/Eigen/src/Core/AssignEvaluator.h:804:3
    #9 0x7f65faed21b2 in void Eigen::internal::call_assignment<Eigen::Matrix<float, 3, 1, 2, 3, 1>, Eigen::Matrix<float, 3, 1, 2, 3, 1> >
(Eigen::Matrix<float, 3, 1, 2, 3, 1>&, Eigen::Matrix<float, 3, 1, 2, 3, 1> const&)
/usr/include/eigen3/Eigen/src/Core/AssignEvaluator.h:782:3
    #10 0x7f65faed2046 in Eigen::Matrix<float, 3, 1, 2, 3, 1>& Eigen::PlainObjectBase<Eigen::Matrix<float, 3, 1, 2, 3, 1>
>::_set<Eigen::Matrix<float, 3, 1, 2, 3, 1> >(Eigen::DenseBase<Eigen::Matrix<float, 3, 1, 2, 3, 1> > const&)
/usr/include/eigen3/Eigen/src/Core/PlainObjectBase.h:714:7
    #11 0x7f65faea9327 in Eigen::Matrix<float, 3, 1, 2, 3, 1>::operator=(Eigen::Matrix<float, 3, 1, 2, 3, 1> const&)
/usr/include/eigen3/Eigen/src/Core/Matrix.h:208:20
    #12 0x7f65fb2021e7 in Slic3r::load_obj(char const*, Slic3r::TriangleMesh*)
/boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/OBJ.cpp:83:30
    #13 0x567aaf in LLVMFuzzerTestOneInput /boop/assorted_fuzzing/prusaslicer/./fuzz_obj_harness.cpp:82:20
    #14 0x46ddd1 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long)
(/boop/assorted_fuzzing/prusaslicer/obj_fuzzdir/fuzzobj.bin+0x46ddd1)
    #15 0x459542 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long)
(/boop/assorted_fuzzing/prusaslicer/obj_fuzzdir/fuzzobj.bin+0x459542)
    #16 0x45eff6 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long))
(/boop/assorted_fuzzing/prusaslicer/obj_fuzzdir/fuzzobj.bin+0x45eff6)
    #17 0x487cb2 in main (/boop/assorted_fuzzing/prusaslicer/obj_fuzzdir/fuzzobj.bin+0x487cb2)
    #18 0x7f65f6e720b2 in __libc_start_main /build/glibc-ZN95T4/glibc-2.31/csu/../csu/libc-start.c:308:16
    #19 0x433c0d in _start (/boop/assorted_fuzzing/prusaslicer/obj_fuzzdir/fuzzobj.bin+0x433c0d)

0x606000010860 is located 12 bytes to the right of 52-byte region [0x606000010820,0x606000010854)
allocated by thread T0 here:
    #0 0x56308d in operator new(unsigned long) (/boop/assorted_fuzzing/prusaslicer/obj_fuzzdir/fuzzobj.bin+0x56308d)
    #1 0x7f65fb47a238 in __gnu_cxx::new_allocator<stl_facet>::allocate(unsigned long, void const*) /usr/bin/../lib/gcc/x86_64-linux-
gnu/9/../../../../include/c++/9/ext/new_allocator.h:114:27
    #2 0x7f65fb47a168 in std::allocator_traits<std::allocator<stl_facet> >::allocate(std::allocator<stl_facet>&, unsigned long)
/usr/bin/../lib/gcc/x86_64-linux-gnu/9/../../../../include/c++/9/bits/alloc_traits.h:444:20
    #3 0x7f65fb47a0cf in std::_Vector_base<stl_facet, std::allocator<stl_facet> >::_M_allocate(unsigned long) /usr/bin/../lib/gcc/x86_64-
linux-gnu/9/../../../../include/c++/9/bits/stl_vector.h:343:20
    #4 0x7f65fb479f1b in std::_Vector_base<stl_facet, std::allocator<stl_facet> >::_M_create_storage(unsigned long)
/usr/bin/../lib/gcc/x86_64-linux-gnu/9/../../../../include/c++/9/bits/stl_vector.h:358:33
    #5 0x7f65fb479b0f in std::_Vector_base<stl_facet, std::allocator<stl_facet> >::_Vector_base(unsigned long, std::allocator<stl_facet>
const&) /usr/bin/../lib/gcc/x86_64-linux-gnu/9/../../../../include/c++/9/bits/stl_vector.h:302:9
    #6 0x7f65fee5cb37 in std::vector<stl_facet, std::allocator<stl_facet> >::vector(unsigned long, stl_facet const&,
std::allocator<stl_facet> const&) /usr/bin/../lib/gcc/x86_64-linux-gnu/9/../../../../include/c++/9/bits/stl_vector.h:521:9
    #7 0x7f65fee5c75e in std::vector<stl_facet, std::allocator<stl_facet> >::_M_fill_assign(unsigned long, stl_facet const&)
/usr/bin/../lib/gcc/x86_64-linux-gnu/9/../../../../include/c++/9/bits/vector.tcc:262:11
    #8 0x7f65fee5b404 in std::vector<stl_facet, std::allocator<stl_facet> >::assign(unsigned long, stl_facet const&)
/usr/bin/../lib/gcc/x86_64-linux-gnu/9/../../../../include/c++/9/bits/stl_vector.h:747:9
    #9 0x7f65fee59998 in stl_allocate(stl_file*) /boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/admesh/stlinit.cpp:248:21
    #10 0x7f65fb201607 in Slic3r::load_obj(char const*, Slic3r::TriangleMesh*)
/boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/OBJ.cpp:55:5
    #11 0x567aaf in LLVMFuzzerTestOneInput /boop/assorted_fuzzing/prusaslicer/./fuzz_obj_harness.cpp:82:20
    #12 0x46ddd1 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long)
(/boop/assorted_fuzzing/prusaslicer/obj_fuzzdir/fuzzobj.bin+0x46ddd1)
    #13 0x459542 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long)
(/boop/assorted_fuzzing/prusaslicer/obj_fuzzdir/fuzzobj.bin+0x459542)
    #14 0x45eff6 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long))
(/boop/assorted_fuzzing/prusaslicer/obj_fuzzdir/fuzzobj.bin+0x45eff6)
    #15 0x487cb2 in main (/boop/assorted_fuzzing/prusaslicer/obj_fuzzdir/fuzzobj.bin+0x487cb2)
    #16 0x7f65f6e720b2 in __libc_start_main /build/glibc-ZN95T4/glibc-2.31/csu/../csu/libc-start.c:308:16

SUMMARY: AddressSanitizer: heap-buffer-overflow
/boop/assorted_fuzzing/prusaslicer/./PrusaSlicer/src/eigen/Eigen/src/Core/functors/AssignmentFunctors.h:24:102 in
Eigen::internal::assign_op<float, float>::assignCoeff(float&, float const&) const
Shadow bytes around the buggy address:
  0x0c0c7fffa0b0: 00 00 00 00 00 00 00 00 fa fa fa fa 00 00 00 00
  0x0c0c7fffa0c0: 00 00 00 00 fa fa fa fa 00 00 00 00 00 00 00 00
  0x0c0c7fffa0d0: fa fa fa fa fd fd fd fd fd fd fd fd fa fa fa fa
  0x0c0c7fffa0e0: 00 00 00 00 00 00 00 00 fa fa fa fa fd fd fd fd
  0x0c0c7fffa0f0: fd fd fd fd fa fa fa fa fd fd fd fd fd fd fd fd
=>0x0c0c7fffa100: fa fa fa fa 00 00 00 00 00 00 04 fa[fa]fa fa fa
  0x0c0c7fffa110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c0c7fffa120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c0c7fffa130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c0c7fffa140: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c0c7fffa150: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
```

Timeline

2020-12-14 - Vendor disclosure
2021-01-14 - Vendor patched
2021-04-21 - Public release

CREDIT

Discovered by Lilith >_> of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2020-1160

NEXT REPORT TALOS-2020-1220
