Talos Vulnerability Report

TALOS-2020-1220

Prusa Research PrusaSlicer Objparser::objparse() stack-based buffer overflow vulnerability

APRIL 21, 2021

CVE NUMBER

CVE-2020-28596

Summary

A stack-based buffer overflow vulnerability exists in the Objparser::objparse() 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-121 - Stack-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 code-path from entry to vulnerability:

```
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], we provide a path to a `.obj` file, which is then processed into the `ObjParser::ObjData` object, which we will examine the structure of when needed. Continuing into `ObjParser::objparse`:

```
bool objparse(const char *path, ObjData &data)
{
    FILE *pFile = boost::nowide::fopen(path, "rt");
    if (pFile == 0)
        return false;

    try {
        char buf[65536 * 2];  // [1]
        size_t len = 0;
        size_t lenPrev = 0;
        while ((len = ::fread(buf + lenPrev, 1, 65536, pFile)) != 0) { // [2]
            len += lenPrev;
            size_t lastLine = 0;
            for (size_t i = 0; i < len; ++ i)
                if (buf[i] == '\r' || buf[i] == '\n') {
                    buf[i] = 0;
                    char *c = buf + lastLine;
                    while (*c == ' ' || *c == '\t')
                        ++ c;
                    obj_parseline(c, data);
                    lastLine = i + 1;
                }
            lenPrev = len - lastLine; //
            memmove(buf, buf + lastLine, lenPrev);
        }
    }
    catch (std::bad_alloc&) {
        printf("Out of memory\r\n");
    }
    ::fclose(pFile);

    // printf("vertices: %d\r\n", data.vertices.size() / 4);
    // printf("coords: %d\r\n", data.coordinates.size());
    return true;
}
```

At [2] we can clearly see the call to `fread` reading into a fixed-size stack buffer at [1] inside of a while loop that is completely at the input file's behest, resulting in a stack-based buffer overflow that can lead to arbitrary code execution.

Crash Information

```
=================================================================
==593876==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7fff5b94fd80 at pc 0x0000004d3521 bp 0x7fff5b92fc70 sp 0x7fff5b92f438
WRITE of size 12928 at 0x7fff5b94fd80 thread T0
    #0 0x4d3520 in fread (/root/boop/assorted_fuzzing/prusaslicer/obj_fuzzdir/fuzzobj.bin+0x4d3520)
    #1 0x7f2f2ceb7a94 in ObjParser::objparse(char const*, ObjParser::ObjData&)
boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/objparser.cpp:332:17

Address 0x7fff5b94fd80 is located in stack of thread T0 at offset 131104 in frame
    #0 0x7f2f2ceb78af in ObjParser::objparse(char const*, ObjParser::ObjData&)
/boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/objparser.cpp:323

  This frame has 1 object(s):
    [32, 131104) 'buf' (line 329) <== Memory access at offset 131104 overflows this variable
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/prusaslicer/obj_fuzzdir/fuzzobj.bin+0x4d3520) in fread
Shadow bytes around the buggy address:
  0x10006b721f60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10006b721f70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10006b721f80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10006b721f90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10006b721fa0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x10006b721fb0:[f3]f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3
  0x10006b721fc0: f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3
  0x10006b721fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10006b721fe0: 00 00 00 00 f1 f1 f1 f1 00 00 00 00 00 00 00 00
  0x10006b721ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10006b722000: 00 00 00 00 00 00 00 00 f2 f2 f2 f2 f2 f2 f2 f2
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
==593876==ABORTING
```

Timeline

2020-12-14 - Vendor disclosure

2021-01-14 - Vendor patched

2021-04-21 - Public release

CREDIT

Discovered by Lilith >_> of Cisco Talos.