Talos Vulnerability Report

TALOS-2020-1215

Slic3r libslic3r AMF File AMFParserContext::endElement() out-of-bounds read vulnerability

FEBRUARY 24, 2020

CVE NUMBER

CVE-2020-28591

Summary

An out-of-bounds read vulnerability exists in the AMF File AMFParserContext::endElement() functionality of Slic3r libslic3r 1.3.0 and Master Commit 92abbc42. A specially crafted AMF file can 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 3-D printing model file types into machine code for any printer. Slic3r uses libslic3er to do most of the non-GUI-based heavy lifting: reading various file formats, converting formats, and outputting appropriate gcode for the 3-D printer's given settings.

When reading in an `.amf` file (which is a specific XML schema), libSlic3r utilizes an `AMFParserContext` object (`libslic3r/IO/AMF.cpp`) for the in-depth parsing and population of data structures thereof. The main AMF-specific functionality is within the `AMFParserContext::startElement` and `AMFParserContext::endElement` callback functions, the latter of which is where our vulnerability lays. But first, an example `.amf` file:

```
<?xml version="1.0" encoding="utf-8"?>
<amf unit="inch" version="1.1">
  <metadata type="name">Split Pyramid</metadata>
  <metadata tyze="author">John Smith</metadata>
  <object id="1">
<mesh>
  <vertices>
    <vertex><coordinates><x>2</x><y>0</y><z>0</z></coordinates></vertex>
    <vertex><coordinates><x>2.5</x><y>0.5</y><z>0</z></coordinates></vertex>
  </vertices>
  <volume materialid="2">
    <metadata type="name">Hard side</metadata>
    <triangle><v1>2</v1><v2>1</v2><v3>2</v3></triangle>
    <triangle><v1>0</v1><v2>0</v2><v3>2</v3></triangle>
  </volume>
</mesh>
  </object>
  <material id="1">
    <metadata type=<g>0.9</g><b>0n9</b><a>0.5</a></color>
  </material>
```

For each node in this XML file, various code paths are taken within the forementioned `AMFParserContext::endElement` function:

```
void AMFParserContext::endElement(const char *name)
{
    switch (m_path.back()) {

    // Constellation transformation:
    case NODE_TYPE_DELTAX:
        assert(m_instance);
        m_instance->deltax = float(atof(m_value[0].c_str()));
        m_instance->deltax_set = true;
        m_value[0].clear();
        break;
    case NODE_TYPE_DELTAY:
        assert(m_instance);
        m_instance->deltay = float(atof(m_value[0].c_str()));
        m_instance->deltay_set = true;
        m_value[0].clear();
        break;
    [...]
```

For current writeup purposes, we only specifically examine two different AMF element types:

```
// Faces of the current volume:
case NODE_TYPE_TRIANGLE:                                // [1]
    assert(m_object && m_volume);
    m_volume_facets.push_back(atoi(m_value[0].c_str())); // [2]
    m_volume_facets.push_back(atoi(m_value[1].c_str()));
    m_volume_facets.push_back(atoi(m_value[2].c_str()));
    m_value[0].clear();
    m_value[1].clear();
    m_value[2].clear();
    break;

// Closing the current volume. Create an STL from m_volume_facets pointing to m_object_vertices.
case NODE_TYPE_VOLUME: // 12
{
    assert(m_object && m_volume);
    stl_file &stl = m_volume->mesh.stl;
    stl.stats.type = inmemory;
    stl.stats.number_of_facets = int(m_volume_facets.size() / 3);
    stl.stats.original_num_facets = stl.stats.number_of_facets;
    stl_allocate(&stl);
    for (size_t i = 0; i < m_volume_facets.size();) {
        stl_facet &facet = stl.facet_start[i/3];
        for (unsigned int v = 0; v < 3; ++ v)
            memcpy(&facet.vertex[v].x, &m_object_vertices[m_volume_facets[i ++] * 3], 3 * sizeof(float)); . // [3]
    }
    stl_get_size(&stl);
    m_volume->mesh.repair();
    m_volume_facets.clear();
    m_volume = NULL;
    break;
}
```

At [1], when we parse a given triangle XML node (e.g. `<triangle>212</triangle>`), all three vertexes are copied into the m_volume_facets vector [2]. This same vector is then read when populating a new set of facets for a new stl_file object at [3]. Since the values inside of the std::vector m_volume_facets` vector are completely under the input file's control, it follows that we can then cause the `memcpy` at [3] to index anywhere from 0x0 to 0xFFFFFFFF, resulting in our new `stl_file` object being populated with out-of-bounds data, resulting in a potential info leak.

Crash Information

```
==1052134==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x606000001850 at pc 0x00000052afca bp 0x7ffcbbc9f890 sp 0x7ffcbbc9f058
READ of size 12 at 0x606000001850 thread T0
    #0 0x52afc9 in __asan_memcpy (/boop/assorted_fuzzing/slic3er/amf_fuzz_dir/sorted_crashes_v1/libfuzzer_amf_harness.bin+0x52afc9)
    #1 0x7fe7aacfc0c2 in Slic3r::IO::AMFParserContext::endElement(char const*)
/boop/assorted_fuzzing/slic3er/Slic3r/xs/src/libslic3r/IO/AMF.cpp:367:17
    #2 0x7fe7aacffedc in Slic3r::IO::AMFParserContext::endElement(void*, char const*)
/boop/assorted_fuzzing/slic3er/Slic3r/xs/src/libslic3r/IO/AMF.cpp:46:14
    #3 0x7fe7ab5c7fb9 in doContent(XML_ParserStruct*, int, encoding const*, char const*, char const*, char const**, unsigned char)
/boop/assorted_fuzzing/slic3er/Slic3r/xs/src/expat/xmlparse.c:2573:11
    #4 0x7fe7ab5b773d in contentProcessor(XML_ParserStruct*, char const*, char const*, char const**)
/boop/assorted_fuzzing/slic3er/Slic3r/xs/src/expat/xmlparse.c:2146:27
    #5 0x7fe7ab5a7041 in doProlog(XML_ParserStruct*, encoding const*, char const*, char const*, int, char const*, char const**, unsigned
char) /boop/assorted_fuzzing/slic3er/Slic3r/xs/src/expat/xmlparse.c:4059:14
    #6 0x7fe7ab5a2a3c in prologProcessor(XML_ParserStruct*, char const*, char const*, char const**)
/boop/assorted_fuzzing/slic3er/Slic3r/xs/src/expat/xmlparse.c:3782:10
    #7 0x7fe7ab5a23e3 in prologInitProcessor(XML_ParserStruct*, char const*, char const*, char const**)
/boop/assorted_fuzzing/slic3er/Slic3r/xs/src/expat/xmlparse.c:3599:10
    #8 0x7fe7ab59e7d9 in XML_ParseBuffer /boop/assorted_fuzzing/slic3er/Slic3r/xs/src/expat/xmlparse.c:1677:15
    #9 0x7fe7ab59c9b1 in XML_Parse /boop/assorted_fuzzing/slic3er/Slic3r/xs/src/expat/xmlparse.c:1643:14
    #10 0x7fe7aacff6f5 in Slic3r::IO::AMF::read(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >,
Slic3r::Model*) /boop/assorted_fuzzing/slic3er/Slic3r/xs/src/libslic3r/IO/AMF.cpp:478:13
    #11 0x55fdec in LLVMFuzzerTestOneInput /boop/assorted_fuzzing/slic3er/./fuzz_amf_harness.cpp:81:9
    #12 0x466011 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long)
(/boop/assorted_fuzzing/slic3er/amf_fuzz_dir/sorted_crashes_v1/libfuzzer_amf_harness.bin+0x466011)
    #13 0x451782 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long)
(/boop/assorted_fuzzing/slic3er/amf_fuzz_dir/sorted_crashes_v1/libfuzzer_amf_harness.bin+0x451782)
    #14 0x457236 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long))
(/boop/assorted_fuzzing/slic3er/amf_fuzz_dir/sorted_crashes_v1/libfuzzer_amf_harness.bin+0x457236)
    #15 0x47fef2 in main (/boop/assorted_fuzzing/slic3er/amf_fuzz_dir/sorted_crashes_v1/libfuzzer_amf_harness.bin+0x47fef2)

Address 0x606000001850 is a wild pointer.
SUMMARY: AddressSanitizer: heap-buffer-overflow
(/boop/assorted_fuzzing/slic3er/amf_fuzz_dir/sorted_crashes_v1/libfuzzer_amf_harness.bin+0x52afc9) in __asan_memcpy
Shadow bytes around the buggy address:
  0x0c0c7fff82b0: 00 00 00 00 00 00 00 00 fa fa fa fa 00 00 00 00
  0x0c0c7fff82c0: 00 00 00 00 fa fa fa fa 00 00 00 00 00 00 00 00
  0x0c0c7fff82d0: fa fa fa fa fd fd fd fd fd fd fd fd fa fa fa fa
  0x0c0c7fff82e0: 00 00 00 00 00 00 00 00 fa fa fa fa fd fd fd fd
  0x0c0c7fff82f0: fd fd fd fd fa fa fa fa 00 00 00 00 00 00 00 00
=>0x0c0c7fff8300: fa fa fa fa fa fa fa fa fa fa[fa]fa fa fa fa fa
  0x0c0c7fff8310: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c0c7fff8320: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c0c7fff8330: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c0c7fff8340: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c0c7fff8350: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
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
==1052134==ABORTING
================================================================================
```

Timeline

2020-12-21 - Vendor Disclosure
2021-02-24 - Public Release

CREDIT

Discovered by Lilith >_> of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2019-0955

NEXT REPORT TALOS-2020-1221