Talos Vulnerability Report

TALOS-2020-1218

Prusa Research PrusaSlicer _3MF_Importer::_handle_end_model() use-after-free vulnerability

APRIL 21, 2020

CVE NUMBER

CVE-2020-28594

Summary

A use-after-free vulnerability exists in the _3MF_Importer::_handle_end_model() functionality of Prusa Research PrusaSlicer 2.2.0 and Master (commit 4b040b856). A specially crafted 3MF 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-416 - Use After Free

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 `.3mf` files, a type of `.zip` file that has a specific directory structure within, most importantly the `3D/` directory that should contain `*.3dmodel` files.
To simplify this advisory, we first present an example `.3dmodel` file and then proceed into the relevant code-paths, found within in `PrusaSlicer/src/libslic3r/Format/3mf.cpp`, that are taken as a result of our particular `.3dmodel` file. To proceed, the following are the contents for `sample.3dmodel`:

```
<?xml version="1.0" encoding="UTF-8"?>
<model unit="millimeter" xml:lang="en-US" xmlns="[...]">    // [1]
  <metadata name="Designer">^_^</metadata>
  <resources>
    <color id="0" value="#79BC53" />
    <color id="1" value="#FF4D96" />
    <material id="2" name="Green" colorid="0" type="ABS" />
    <material id="3" name="Coral" colorid="1" type="ABS" />
    <object id="4" type="model" materialid="2">             // [2]
    <mesh>
        <vertices>
          <vertex x="137.251999" y="144.088013" z="37.256001" />
          <vertex x="139.956009" y="145.754013" z="37.008003" />
        </vertices>
        <triangles>
          <triangle v1="0" v2="1" v3="2" />
          <triangle v1="6" v2="5" v3="7" />
          <triangle v1="1" v2="0" v3="19" />
        </triangles>
      </mesh>
      </object>
   </resources>
  <build>
    <ited objectid="8" transform="25.400000 0.000000 0.000000 0.000000" />
    <ited objectid="9" transform="25.400000 0.000000 0.000000 0.000000" />
  </build>
</model>   // [3]
```

While this example has been truncated and it's unknown if it'd output anything intelligible, it still has the basic properties that we care about: at [1] we obviously have a `<model>` defined, wherein all other definitions are placed. At [2], we observe an `<object>` tag, which generally contains actual coordinates of points and shapes and vertices, the important data, and finally at [3] we end our current model definition. While [3] is among the most basic lines, let us examine the resultant function that is called:

```
bool _3MF_Importer::_handle_end_model()
{
    // deletes all non-built or non-instanced objects
    for (const IdToModelObjectMap::value_type& object : m_objects)      // [1]
    {
        ModelObject *model_object = m_model->objects[object.second];    // [2]
        if ((model_object != nullptr) && (model_object->instances.size() == 0))
            m_model->delete_object(model_object);                       // [3]
    }
// [...]
```

As kindly noted by the comment, once done with a given `<model>` we delete all the non-built/non-instanced objects. To do this we first walk the `m_objects` vector [1], which is a mapping of `objectid` to order of the objects within `m_model->objects`, and grab the appropriate index of `m_model->objects` at [2]. Finally, assuming it's non-null and its size is 0x0, then it's deleted at [3]. While this is admittedly rather abstract and vague, displaying example `m_objects` and `m_model->objects` structures quickly clarifies, so let us look at what goes on with `sample.3dmodel`:

```
[>_>]#p/x m_model->objects
$1 = std::vector of length 1, capacity 1 = {0x61600008be80}

[^~^]#p/x m_objects
$2 = std::map with 1 element = {[0x4] = 0x0}
```

The above structures are created due to the `<object id="4" type="model" materialid="2">` line within `sample.3dmodel`. Likewise, if our `sample.3dmodel` file contained another `<object>` entry with an `id=10`, our structures would look as such:

```
//  <object id="4" type="model" materialid="2">
//  <object id="10" type="model" materialid="3">

[>_>]#p/x m_model->objects
$1 = std::vector of length 2, capacity 2 = {0x61600008be80, 0x61600008c780,}

[^~^]#p/x m_objects
$2 = std::map with 2 element = {[0x4] = 0x0, [0xa] = 0x1}
```

Since `m_objects` is a map, it's also worth noting that if we have a `.3dmodel` file with three objects inside, like such:

```
<object id="8" type="model" materialid="4">
<object id="4" type="model" materialid="2">
<object id="10" type="model" materialid="3">
```

The resulting `m_objects` structure would then look like such:

```
 [^~^]#p/x m_objects
 $2 = std::map with 2 element = {[0x4] = 0x1, [0x8] = 0x0, [0xa] = 0x2}
```

To emphasize further: the format of the map is `<object id> => <ordering in which the objects were found>` Since the actual order in the map is determined by the `objectid`, which is read from the `.3dmodel` file, we naturally control the ordering of the elements within. Thus, let us explore the possibility of a given set of objects within a model as such:

```
 <object id="11" type="model" materialid="3">
 <object id="8" type="model" materialid="4">
 <object id="10" type="model" materialid="2">
 <object id="6" type="model" materialid="3">
 <object id="7" type="model" materialid="2">
```

Assuming all of the given objects above were not built or instantiated, when we hit our `</model>` tag, the `m_objects` map and `m_model->objects` vector might look like such:

```
[x.x]#p/x m_objects
$3 = std::map with 5 elements = {[0x6] = 0x3, [0x7] = 0x4, [0x8] = 0x1, [0xa] = 0x2, [0xb] = 0x0}

<(^.^)>#p/x m_model->objects
$6 = std::vector of length 5, capacity 8 = {0x61600008c480, 0x61600008c780, 0x61600008ca80, 0x61600008be80, 0x61600008cd80}
```

Thus, let us walk through the iterations of the loop presented again below:

```
bool _3MF_Importer::_handle_end_model()
   {
    // deletes all non-built or non-instanced objects
   for (const IdToModelObjectMap::value_type& object : m_objects) //[1]
         {
             ModelObject *model_object = m_model->objects[object.second]; //[2]
             if ((model_object != nullptr) && (model_object->instances.size() == 0))
                 m_model->delete_object(model_object);   //[3]
         }
       //[...]
```

The first iteration pulls out `object([0x6]=>0x3)` at [1], which causes us to grab `m_model->objects[3]` (0x61600008be80) and delete it at [3]. Again displaying `m_objects` and `m_model->objects` after this:

```
[x.x]#p/x m_objects
$3 = std::map with 5 elements = {[0x6] = 0x3, [0x7] = 0x4, [0x8] = 0x1, [0xa] = 0x2, [0xb] = 0x0}

<(^.^)>#p/x m_model->objects
$6 = std::vector of length 4, capacity 8 = {0x61600008c480, 0x61600008c780, 0x61600008ca80, 0x61600008cd80}
```

Expectedly, we see the second index, `0x61600008be80`, removed from the `m_model->objects` vector, but no change to `m_objects`. Upon the next loop we grab `object([0x7]=>0x3)`, and free index 0x3 from the `m_model->objects` vector. Displaying `m_objects` and `m_model->objects` again:

```
[x.x]#p/x m_objects
$3 = std::map with 5 elements = {[0x6] = 0x3, [0x7] = 0x4, [0x8] = 0x1, [0xa] = 0x2, [0xb] = 0x0}

[o_O]#p/x m_model->objects
$6 = std::vector of length 3, capacity 8 = {0x61600008c480, 0x61600008c780, 0x61600008ca80}
```

If the vulnerability is not quite made plain yet, the next iteration finds `object([0x8]=>0x1)`, deleting index 0x1 (`0x61600008c780`), and we're left with:

```
[x.x]#p/x m_objects
$3 = std::map with 5 elements = {[0x6] = 0x3, [0x7] = 0x4, [0x8] = 0x1, [0xa] = 0x2, [0xb] = 0x0}

[o_O]#p/x m_model->objects
$6 = std::vector of length 2, capacity 8 = {0x61600008c480, 0x61600008ca80}
```

Upon the next iteration, `object([0xa]=>0x2)` causes us to grab an out-of-bounds index from `m_model->objects`, which contains a pointer to an already deleted `model_object`, resulting in a double-free and potential code execution.

Crash Information

```
==1124150==ERROR: AddressSanitizer: heap-use-after-free on address 0x61600008cad8 at pc 0x7ffff105df3d bp 0x7fffffff7190 sp 0x7fffffff7188
READ of size 8 at 0x61600008cad8 thread T0
[Detaching after fork from child process 1124836]
    #0 0x7ffff105df3c in std::vector<Slic3r::ModelInstance*, std::allocator<Slic3r::ModelInstance*> >::size() const
/usr/bin/../lib/gcc/x86_64-linux-gnu/9/../../../../include/c++/9/bits/stl_vector.h:916:40
    #1 0x7ffff1059e3e in Slic3r::_3MF_Importer::_handle_end_model()
/boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/3mf.cpp:1384:71
    #2 0x7ffff10595ed in Slic3r::_3MF_Importer::_handle_end_model_xml_element(char const*)
/boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/3mf.cpp:1296:19
    #3 0x7ffff10517b8 in Slic3r::_3MF_Importer::_handle_end_model_xml_element(void*, char const*)
/boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/3mf.cpp:1948:23
    #4 0x7fffeb5759d9  (/lib/x86_64-linux-gnu/libexpat.so.1+0xb9d9)
    #5 0x7fffeb5766af  (/lib/x86_64-linux-gnu/libexpat.so.1+0xc6af)
    #6 0x7fffeb573b82  (/lib/x86_64-linux-gnu/libexpat.so.1+0x9b82)
    #7 0x7fffeb57504d  (/lib/x86_64-linux-gnu/libexpat.so.1+0xb04d)
    #8 0x7fffeb578dbf in XML_ParseBuffer (/lib/x86_64-linux-gnu/libexpat.so.1+0xedbf)
    #9 0x7ffff108aad6 in Slic3r::_3MF_Importer::_extract_model_from_archive(mz_zip_archive&, mz_zip_archive_file_stat
const&)::$_0::operator()(void*, unsigned long, void const*, unsigned long) const /boop/
assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/3mf.cpp:780:22
    #10 0x7ffff108a794 in Slic3r::_3MF_Importer::_extract_model_from_archive(mz_zip_archive&, mz_zip_archive_file_stat
const&)::$_0::__invoke(void*, unsigned long, void const*, unsigned long) /boop/assorte
d_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/3mf.cpp:778:85
    #11 0x7ffff4e49e5f in mz_zip_reader_extract_to_callback /boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/miniz/miniz.c:4742:25
    #12 0x7ffff4e4aade in mz_zip_reader_extract_file_to_callback /boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/miniz/miniz.c:4795:12
    #13 0x7ffff103dd5e in Slic3r::_3MF_Importer::_extract_model_from_archive(mz_zip_archive&, mz_zip_archive_file_stat const&)
/boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/3mf.cpp:77
8:19
    #14 0x7ffff103a19b in Slic3r::_3MF_Importer::_load_model_from_file(std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&, Slic3r::Model&, Slic3r::DynamicPrintConfig&) /
boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/3mf.cpp:600:30
    #15 0x7ffff1038cfe in Slic3r::_3MF_Importer::load_model_from_file(std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&, Slic3r::Model&, Slic3r::DynamicPrintConfig&, bool) /
root/boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/3mf.cpp:553:16
    #16 0x7ffff1082b47 in Slic3r::load_3mf(char const*, Slic3r::DynamicPrintConfig*, Slic3r::Model*, bool)
/boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/3mf.cpp:2854:29
    #17 0x565aa6 in LLVMFuzzerTestOneInput /boop/assorted_fuzzing/prusaslicer/./fuzz_3mf_harness.cpp:82:20
    #18 0x46be11 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long)
(/boop/assorted_fuzzing/prusaslicer/3mf_fuzzdir/fuzz3mf.bin+0x46be11)
    #19 0x457582 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long)
(/boop/assorted_fuzzing/prusaslicer/3mf_fuzzdir/fuzz3mf.bin+0x457582)
    #20 0x45d036 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long))
(/boop/assorted_fuzzing/prusaslicer/3mf_fuzzdir/fuzz3mf.bin+0x45d036)
    #21 0x485cf2 in main (/boop/assorted_fuzzing/prusaslicer/3mf_fuzzdir/fuzz3mf.bin+0x485cf2)
    #22 0x7fffece160b2 in __libc_start_main /build/glibc-ZN95T4/glibc-2.31/csu/../csu/libc-start.c:308:16
    #23 0x431c4d in _start (/boop/assorted_fuzzing/prusaslicer/3mf_fuzzdir/fuzz3mf.bin+0x431c4d)

0x61600008cad8 is located 88 bytes inside of 600-byte region [0x61600008ca80,0x61600008ccd8)
freed by thread T0 here:
    #0 0x56192d in operator delete(void*) (/boop/assorted_fuzzing/prusaslicer/3mf_fuzzdir/fuzz3mf.bin+0x56192d)
    #1 0x7ffff13cf9ef in Slic3r::Model::delete_object(Slic3r::ModelObject*)
/boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Model.cpp:232:17
    #2 0x7ffff1059ebf in Slic3r::_3MF_Importer::_handle_end_model()
/boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/3mf.cpp:1385:26
    #3 0x7ffff10595ed in Slic3r::_3MF_Importer::_handle_end_model_xml_element(char const*)
/boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/3mf.cpp:1296:19
    #4 0x7ffff10517b8 in Slic3r::_3MF_Importer::_handle_end_model_xml_element(void*, char const*)
/boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/3mf.cpp:1948:23
    #5 0x7fffeb5759d9  (/lib/x86_64-linux-gnu/libexpat.so.1+0xb9d9)

previously allocated by thread T0 here:
    #0 0x5610cd in operator new(unsigned long) (/boop/assorted_fuzzing/prusaslicer/3mf_fuzzdir/fuzz3mf.bin+0x5610cd)
    #1 0x7ffff13cc783 in Slic3r::Model::add_object() /boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Model.cpp:177:32
    #2 0x7ffff1057116 in Slic3r::_3MF_Importer::_handle_start_object(char const**, unsigned int)
/boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/3mf.cpp:1426:45
    #3 0x7ffff10565a3 in Slic3r::_3MF_Importer::_handle_start_model_xml_element(char const*, char const**)
/boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/3mf.cpp:1262:19
    #4 0x7ffff1051718 in Slic3r::_3MF_Importer::_handle_start_model_xml_element(void*, char const*, char const**)
/boop/assorted_fuzzing/prusaslicer/PrusaSlicer/src/libslic3r/Format/3mf.cpp:1941:23
    #5 0x7fffeb5756b8  (/lib/x86_64-linux-gnu/libexpat.so.1+0xb6b8)

SUMMARY: AddressSanitizer: heap-use-after-free /usr/bin/../lib/gcc/x86_64-linux-gnu/9/../../../../include/c++/9/bits/stl_vector.h:916:40 in
std::vector<Slic3r::ModelInstance*, std::allocator<Slic3r::ModelInstan
ce*> >::size() const
Shadow bytes around the buggy address:
  0x0c2c80009900: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c2c80009910: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c2c80009920: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c2c80009930: 00 00 00 00 00 00 00 00 00 00 00 00 fa fa fa fa
  0x0c2c80009940: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x0c2c80009950: fd fd fd fd fd fd fd fd fd fd fd[fd]fd fd fd fd
  0x0c2c80009960: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c2c80009970: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c2c80009980: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c2c80009990: fd fd fd fd fd fd fd fd fd fd fd fa fa fa fa fa
  0x0c2c800099a0: 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
==1124150==ABORTING

================================================================================
```

Timeline

2020-12-14 - Vendor Disclosure
2021-04-21 - Public Release

CREDIT

Discovered by Lilith >_> of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2020-1052

NEXT REPORT TALOS-2020-1223