Talos Vulnerability Report

TALOS-2020-1103

Pixar OpenUSD Binary File Format Token Strings Information Leak Vulnerability

NOVEMBER 12, 2020

CVE NUMBER

CVE-2020-13494

Summary

A heap overflow vulnerability exists in the Pixar OpenUSD 20.05 parsing of compressed string tokens in binary USD files. A specially crafted malformed file can trigger a heap overflow which can result in out of bounds memory access which could lead to information disclosure. This vulnerability could be used to bypass mitigations and aid further exploitation. To trigger this vulnerability, victim needs to access an attacker-provided malformed file.

Tested Versions

Pixar OpenUSD 20.05
Apple macOS Catalina 10.15.3

Product URLs

https://openusd.org

CVSSv3 Score

4.3 - CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:N/A:N

CWE

CWE-122 - Heap-based Buffer Overflow

Details

OpenUSD stands for open Universal Scene Descriptor and is a software suite by Pixar that facilitates, among other things, interchange of arbitrary 3-D scenes that may be composed of many elemental assets.

Most notably, USD and its backing file format `usd` are used on Apple iOS and macOS as part of ModelIO framework in support of SceneKit and ARKit for sharing and displaying 3D scenes in, for example, augmented reality applications. On macOS, these files are automatically rendered to generate thumbnails, while on iOS they can be shared via iMessage and opened with user interaction.

USD binary file format consists of a header pointing to a table of contents that in turn points to individual sections that comprise the whole file. Section `TOKENS` contains an LZ4 compressed buffer containing an array of C-style strings. Tokens section consists of a 64bit `numTokens` value specifying total value of strings, a 64 bit uncompressed size value specifying the expected size of uncompressed buffer, a 64 bit compressed size of the following LZ4 buffer.
The following code is invoked when parsing this section:

```
auto numTokens = reader.template Read<uint64_t>();          [1]

RawDataPtr chars;

Version fileVer(_boot);
if (fileVer < Version(0,4,0)) {
    // XXX: To support pread(), we need to read the whole thing into memory
    // to make tokens out of it.  This is a pessimization vs mmap, from
    // which we can just construct from the chars directly.
    auto tokensNumBytes = reader.template Read<uint64_t>();
    chars.reset(new char[tokensNumBytes]);
    reader.ReadContiguous(chars.get(), tokensNumBytes);
} else {
    // Compressed token data.
    uint64_t uncompressedSize = reader.template Read<uint64_t>();         [2]
    uint64_t compressedSize = reader.template Read<uint64_t>();             [3]
    chars.reset(new char[uncompressedSize]);
    RawDataPtr compressed(new char[compressedSize]);
    reader.ReadContiguous(compressed.get(), compressedSize);              [4]
    TfFastCompression::DecompressFromBuffer(
        compressed.get(), chars.get(), compressedSize, uncompressedSize); [5]
}

// Now we read that many null-terminated strings into _tokens.
char const *p = chars.get();
_tokens.clear();
_tokens.resize(numTokens);

WorkArenaDispatcher wd;
struct MakeToken {
    void operator()() const { (*tokens)[index] = TfToken(str); }
    vector<TfToken> *tokens;
    size_t index;
    char const *str;
};
for (size_t i = 0; i != numTokens; ++i) {          [6]
    MakeToken mt { &_tokens, i, p };
    wd.Run(mt);
    p += strlen(p) + 1; [7]
}
```

At [1] in the above code, a total number of string tokens is read. At [2] and [3], expected uncompressed size and size of compressed buffer are read from the file. Compressed buffer is read at [4] with correct size and decompression is invoked with expected uncompressed size at [5]. If the buffer is successfully decompressed, processing continues to [6] where at least `numTokens` will be decoded into a `_tokens` vector. Null terminated strings are extracted at [7], however, no check is performed to ensure the string parsing doesn't continue past the end of the decompressed buffer. Supplying a value of `numTokens` greater than the actual number of null terminated strings in the decompressed buffers will continue to access memory beyond the end of the buffer into adjacent heap memory. Strings from adjacent memory will be copied into the `_tokens` vector until `numTokens` are reached or an access violation occurs causing an application to crash. Data read from out of bounds memory could contain process sensitive information such as heap or process metadata, which can then be referenced by other parts of the file and influence file parsing further. This could be abused to achieve information leak and defeat exploitation mitigations such as ASLR.

Crash Information

This vulnerability has been tested on latest version of macOS Catalina 10.15.3 but the crash is more easily apparent with AddressSanitizer:

```
=================================================================
==107873==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60d00000110a at pc 0x7f560612ba6d bp 0x7ffdc0827870 sp 0x7ffdc0827018
READ of size 1 at 0x60d00000110a thread T0
    #0 0x7f560612ba6c  (/lib/x86_64-linux-gnu/libasan.so.5+0x67a6c)
    #1 0x7f55fb2166ea in void
pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::_ReadTokens<pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::_Reader<
pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::_MmapStream<pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::_FileMapping*> > >
(pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::_Reader<pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::_MmapStream<pxrInterna
l_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::_FileMapping*> >) ./release/USD-20.05/pxr/usd/usd/crateFile.cpp:3305
    #2 0x7f55fb2d78fe in void
pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::_ReadStructuralSections<pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFi
le::_Reader<pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::_MmapStream<pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::_FileMa
pping*> > >
(pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::_Reader<pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::_MmapStream<pxrInterna
l_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::_FileMapping*> >, long) ./release/USD-20.05/pxr/usd/usd/crateFile.cpp:3034
    #3 0x7f55fafdf25b in pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::_InitMMap() ./release/USD-
20.05/pxr/usd/usd/crateFile.cpp:2140
    #4 0x7f55fafe7a3a in pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::CrateFile(std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char> > const&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >
const&, boost::intrusive_ptr<pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::_FileMapping>,
std::shared_ptr<pxrInternal_v0_20__pxrReserved__::ArAsset> const&) ./release/USD-20.05/pxr/usd/usd/crateFile.cpp:2104
    #5 0x7f55fafebdf6 in pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::Open(std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char> > const&) ./release/USD-20.05/pxr/usd/usd/crateFile.cpp:2051
    #6 0x7f55fae7081b in pxrInternal_v0_20__pxrReserved__::Usd_CrateDataImpl::Open(std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&) ./release/USD-20.05/pxr/usd/usd/crateData.cpp:198
    #7 0x7f55fae7081b in pxrInternal_v0_20__pxrReserved__::Usd_CrateData::Open(std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&) ./release/USD-20.05/pxr/usd/usd/crateData.cpp:1205
    #8 0x7f55fcb5247b in pxrInternal_v0_20__pxrReserved__::UsdUsdcFileFormat::Read(pxrInternal_v0_20__pxrReserved__::SdfLayer*,
std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, bool) const ./release/USD-
20.05/pxr/usd/usd/usdcFileFormat.cpp:95
    #9 0x7f5605541f4b in pxrInternal_v0_20__pxrReserved__::SdfLayer::_Read(std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, bool) ./release/USD-
20.05/pxr/usd/sdf/layer.cpp:1045
    #10 0x7f560560c9ed in pxrInternal_v0_20__pxrReserved__::TfRefPtr<pxrInternal_v0_20__pxrReserved__::SdfLayer>
pxrInternal_v0_20__pxrReserved__::SdfLayer::_OpenLayerAndUnlockRegistry<tbb::queuing_rw_mutex::scoped_lock>
(tbb::queuing_rw_mutex::scoped_lock&, pxrInternal_v0_20__pxrReserved__::SdfLayer::_FindOrOpenLayerInfo const&, bool) ./release/USD-
20.05/pxr/usd/sdf/layer.cpp:3072
    #11 0x7f56055dda4f in pxrInternal_v0_20__pxrReserved__::SdfLayer::FindOrOpen(std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&, std::map<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >,
std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::less<std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char> > >, std::allocator<std::pair<std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > > > > const&) ./release/USD-
20.05/pxr/usd/sdf/layer.cpp:819
    #12 0x562eb06bcbaa in main ./release/USD-20.05/pxr/usd/bin/sdfdump/sdfdump.cpp:522
    #13 0x7f5602f390b2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x270b2)
    #14 0x562eb06c76bd in _start (./release/USD-20.05/build/bin/sdfdump+0x2a6bd)

0x60d00000110a is located 0 bytes to the right of 138-byte region [0x60d000001080,0x60d00000110a)
allocated by thread T0 here:
    #0 0x7f56061d3b47 in operator new[](unsigned long) (/lib/x86_64-linux-gnu/libasan.so.5+0x10fb47)
    #1 0x7f55fb215d16 in void
pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::_ReadTokens<pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::_Reader<
pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::_MmapStream<pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::_FileMapping*> > >
(pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::_Reader<pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::_MmapStream<pxrInterna
l_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::_FileMapping*> >) ./release/USD-20.05/pxr/usd/usd/crateFile.cpp:3283

SUMMARY: AddressSanitizer: heap-buffer-overflow (/lib/x86_64-linux-gnu/libasan.so.5+0x67a6c)
Shadow bytes around the buggy address:
  0x0c1a7fff81d0: fd fd fd fa fa fa fa fa fa fa fa fa fd fd fd fd
  0x0c1a7fff81e0: fd fd fd fd fd fd fd fd fd fd fd fd fd fa fa fa
  0x0c1a7fff81f0: fa fa fa fa fa fa 00 00 00 00 00 00 00 00 00 00
  0x0c1a7fff8200: 00 00 00 00 00 00 00 fa fa fa fa fa fa fa fa fa
  0x0c1a7fff8210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c1a7fff8220: 00[02]fa fa fa fa fa fa fa fa fd fd fd fd fd fd
  0x0c1a7fff8230: fd fd fd fd fd fd fd fd fd fd fd fa fa fa fa fa
  0x0c1a7fff8240: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c1a7fff8250: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c1a7fff8260: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c1a7fff8270: 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
==107873==ABORTING
```

Timeline

2020-07-01 - Vendor Disclosure
2020-07-16 - Talos tested and confirmed fix with latest beta of macOS Catalina 10.15.6
2020-11-12 - Public Release

CREDIT

Discovered by Aleksandar Nikolic of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2020-1104

NEXT REPORT TALOS-2020-1101

© 2022 Cisco Systems, Inc. and/or its affiliates. All rights reserved. View our Privacy Policy.