Talos Vulnerability Report

TALOS-2020-1105

Pixar OpenUSD binary file format index type values information leak vulnerability

NOVEMBER 12, 2020

CVE NUMBER

CVE-2020-13498,CVE-2020-13496,CVE-2020-13497

SUMMARY

An exploitable vulnerability exists in the way Pixar OpenUSD 20.05 handles parses certain encoded types. A specially crafted malformed file can trigger an arbitrary 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, the victim needs to access an attacker-provided malformed file.

CONFIRMED VULNERABLE VERSIONS

The versions below were either tested or verified to be vulnerable by Talos or confirmed to be vulnerable by the vendor.

Pixar OpenUSD 20.05
Apple macOS Catalina 10.15.3

PRODUCT URLS

OpenUSD - https://openusd.org macOS - https://apple.com

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-125 - Out-of-bounds Read

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 `FIELDS` contains two arrays, second one labeled as reps. Compressed in binary, this array holds an array of 64 bit integers which encode references to extra data. Encoding consists of three top bits specifying whether the value is inline, compressed or array. Inlined values are directly represented in the first 48 bits, otherwise those 48 bits represent an offset into the file where the actual value can be found. Besides three most significant bits, 5 are unused and next eight encode one of 54 different value types (ints, floats, matrices, vectors , special types…). The immediate inlined value for certain types simply represets an index into other structures, so extracting these values boils down to retrieving a value at the specified index. For example, types such as `String` `TfToken` and `SdfPath` are read like so:

```
string Read(string *) { return crate->GetString(Read<StringIndex>()); }
TfToken Read(TfToken *) { return crate->GetToken(Read<TokenIndex>()); }
SdfPath Read(SdfPath *) { return crate->GetPath(Read<PathIndex>()); }
```

CVE-2020-13496 - TfToken Type Index Out Of Bounds Read

In the previously quoted code, getting the specific `TfToken` is done by calling `GetToken` with an index. This index is retrieved by reading the value inlined in `ValueRep`, the `Read<TokenIndex>` templated methond does just that. Method `GetToken` in turn uses that index to access the specific token:

```
inline TfToken const &
GetToken(TokenIndex i) const { return _tokens[i.value]; }
```

The `_tokens` array is allocated and populated when parsing the `TOKENS` section of the file and will therefore have a predetermined and known size.
No check is performed to ensure the index is less than or equal to the number of actuall tokens in the array. If a large index value is specified , out of bounds memory can be referenced, which could lead to process' sensitive information disclosure or further memory corruption.

CVE-2020-13497 - String Type Index Out Of Bounds Read

Similarly to previous vulnerability, the reference to `String` type is retrieved by calling `GetString` with a trusted index. The index is again simply read from a file via `Read<StringIndex>`. Inside `GetString` the string is retrieved

```
inline std::string const &
GetString(StringIndex i) const {
    return GetToken(_strings[i.value]).GetString();
}
```

Again, the `_strings` array is allocated and populated when initially parsing the `STRINGS` section. No check is performed here to ensure the index is less than or equal to number of items in `_strings` array and a large enough index value can lead to out of bounds memory access which can potentially be abused to leak sensitive process information or cause further memory corruption.

CVE-2020-13498 - SdfPath Type Index Out Of Bounds Read

Lastly, the `SdfPath` is also an indexed type affected by a similar vulnerability. `SdfPath` type is read as follows:

```
SdfPath Read(SdfPath *) { return crate->GetPath(Read<PathIndex>()); }
```

In the above, a path index is used to index into `PATHS` array. The `GetPath` method is as follows:

```
inline SdfPath const &
GetPath(PathIndex i) const { return _paths[i.value]; }
```

Referenced `_paths` array is reconstructed from the contents of the `PATHS` section during initial parsing. An arbitrary index value in this field can result in out of boudns memory access because the lack of bounds checking.

USD files are usually distributed as `usdz` archives which can contains multiple distinct `usd` files each referencing each other, with careful file layout , this sort of vulnerability could be abused to probe the memory layout and influence which files are successfully loaded. Especially, `String` and `SdfPath` values are used to represent references so reading out of bounds values, and having possible predictions as names of `usdc` files in the archive could lead to further abuse of these vulnerabilities. This could be abused to achieve information leak and defeat exploitation mitigations such as ASLR.

Crash Information

ASAN crash:

```
=================================================================
==108654==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60c000042184 at pc 0x7f3221ebff3e bp 0x7ffd9277e120 sp 0x7ffd9277e110
AddressSanitizer:DEADLYSIGNAL
READ of size 4 at 0x60c000042184 thread T0
    #0 0x7f3221ebff3d in
pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::GetToken(pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::TokenIndex) const
./USD-20.05/pxr/usd/usd/crateFile.h:651
    #1 0x7f3221ebff3d in pxrInternal_v0_20__pxrReserved__::Usd_CrateDataImpl::_PopulateFromCrateFile()::{lambda()#5}::operator()() ./USD-
20.05/pxr/usd/usd/crateData.cpp:866
    #2 0x7f3221ebff3d in void std::__invoke_impl<void, pxrInternal_v0_20__pxrReserved__::Usd_CrateDataImpl::_PopulateFromCrateFile()::
{lambda()#5}&>(std::__invoke_other, pxrInternal_v0_20__pxrReserved__::Usd_CrateDataImpl::_PopulateFromCrateFile()::{lambda()#5}&)
/usr/include/c++/9/bits/invoke.h:60
    #3 0x7f3221ebff3d in std::__invoke_result<pxrInternal_v0_20__pxrReserved__::Usd_CrateDataImpl::_PopulateFromCrateFile()::
{lambda()#5}&>::type std::__invoke<pxrInternal_v0_20__pxrReserved__::Usd_CrateDataImpl::_PopulateFromCrateFile()::{lambda()#5}&>
(std::__invoke_result&, (pxrInternal_v0_20__pxrReserved__::Usd_CrateDataImpl::_PopulateFromCrateFile()::{lambda()#5}&)...)
/usr/include/c++/9/bits/invoke.h:95
    #4 0x7f3221ebff3d in void std::_Bind<pxrInternal_v0_20__pxrReserved__::Usd_CrateDataImpl::_PopulateFromCrateFile()::{lambda()#5}
()>::__call<void>(std::tuple<>&&, std::_Index_tuple<>) /usr/include/c++/9/functional:400
    #5 0x7f3221ebff3d in void std::_Bind<pxrInternal_v0_20__pxrReserved__::Usd_CrateDataImpl::_PopulateFromCrateFile()::{lambda()#5}
()>::operator()<, void>() /usr/include/c++/9/functional:484
    #6 0x7f3221ebff3d in
pxrInternal_v0_20__pxrReserved__::WorkDispatcher::_InvokerTask<std::_Bind<pxrInternal_v0_20__pxrReserved__::Usd_CrateDataImpl::_PopulateFrom
CrateFile()::{lambda()#5} ()> >::execute() ./USD-20.05/pxr/base/work/dispatcher.h:145
    #7 0x7f3229479ea4 in tbb::internal::custom_scheduler<tbb::internal::IntelSchedulerTraits>::local_wait_for_all(tbb::task&, tbb::task*)
../../src/tbb/custom_scheduler.h:501
    #8 0x7f32297d8b70 in tbb::task::wait_for_all() ./USD-20.05/build/include/tbb/task.h:760
    #9 0x7f32297d8b70 in pxrInternal_v0_20__pxrReserved__::WorkDispatcher::Wait() ./USD-20.05/pxr/base/work/dispatcher.cpp:52
    #10 0x7f32297d4af0 in void std::__invoke_impl<void, void (pxrInternal_v0_20__pxrReserved__::WorkDispatcher::* const&)(),
pxrInternal_v0_20__pxrReserved__::WorkDispatcher* const&>(std::__invoke_memfun_deref, void
(pxrInternal_v0_20__pxrReserved__::WorkDispatcher::* const&)(), pxrInternal_v0_20__pxrReserved__::WorkDispatcher* const&)
/usr/include/c++/9/bits/invoke.h:73
    #11 0x7f32297d4af0 in std::__invoke_result<void (pxrInternal_v0_20__pxrReserved__::WorkDispatcher::* const&)(),
pxrInternal_v0_20__pxrReserved__::WorkDispatcher* const&>::type std::__invoke<void (pxrInternal_v0_20__pxrReserved__::WorkDispatcher::*
const&)(), pxrInternal_v0_20__pxrReserved__::WorkDispatcher* const&>(void (pxrInternal_v0_20__pxrReserved__::WorkDispatcher::* const&)(),
pxrInternal_v0_20__pxrReserved__::WorkDispatcher* const&) /usr/include/c++/9/bits/invoke.h:95
    #12 0x7f32297d4af0 in void std::_Bind<void (pxrInternal_v0_20__pxrReserved__::WorkDispatcher::*
(pxrInternal_v0_20__pxrReserved__::WorkDispatcher*))()>::__call_c<void, , 0ul>(std::tuple<>&&, std::_Index_tuple<0ul>) const
/usr/include/c++/9/functional:410
    #13 0x7f32297d4af0 in void std::_Bind<void (pxrInternal_v0_20__pxrReserved__::WorkDispatcher::*
(pxrInternal_v0_20__pxrReserved__::WorkDispatcher*))()>::operator()<, void>() const /usr/include/c++/9/functional:495
    #14 0x7f32297d4af0 in tbb::interface7::internal::delegated_function<std::_Bind<void (pxrInternal_v0_20__pxrReserved__::WorkDispatcher::*
(pxrInternal_v0_20__pxrReserved__::WorkDispatcher*))()> const>::operator()() const ./USD-20.05/build/include/tbb/task_arena.h:62
    #15 0x7f3229474eac in tbb::interface7::internal::task_arena_base::internal_execute(tbb::interface7::internal::delegate_base&) const
../../src/tbb/arena.cpp:811
    #16 0x7f32297d1b9a in void tbb::interface7::task_arena::execute<std::_Bind<void (pxrInternal_v0_20__pxrReserved__::WorkDispatcher::*
(pxrInternal_v0_20__pxrReserved__::WorkDispatcher*))()> >(std::_Bind<void (pxrInternal_v0_20__pxrReserved__::WorkDispatcher::*
(pxrInternal_v0_20__pxrReserved__::WorkDispatcher*))()> const&) ./USD-20.05/build/include/tbb/task_arena.h:272
    #17 0x7f32297d1b9a in pxrInternal_v0_20__pxrReserved__::WorkArenaDispatcher::Wait() ./USD-20.05/pxr/base/work/arenaDispatcher.cpp:100
    #18 0x7f3221f20e82 in pxrInternal_v0_20__pxrReserved__::Usd_CrateDataImpl::_PopulateFromCrateFile() ./USD-
20.05/pxr/usd/usd/crateData.cpp:872
    #19 0x7f3221e708b7 in pxrInternal_v0_20__pxrReserved__::Usd_CrateDataImpl::Open(std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&) ./USD-20.05/pxr/usd/usd/crateData.cpp:200
    #20 0x7f3221e708b7 in pxrInternal_v0_20__pxrReserved__::Usd_CrateData::Open(std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&) ./USD-20.05/pxr/usd/usd/crateData.cpp:1205
    #21 0x7f3223b5247b 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 ./USD-
20.05/pxr/usd/usd/usdcFileFormat.cpp:95
    #22 0x7f322c545f4b 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) ./USD-
20.05/pxr/usd/sdf/layer.cpp:1045
    #23 0x7f322c6109ed 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) ./USD-
20.05/pxr/usd/sdf/layer.cpp:3072
    #24 0x7f322c5e1a4f 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&) ./USD-
20.05/pxr/usd/sdf/layer.cpp:819
    #25 0x55585e80dbaa in main ./USD-20.05/pxr/usd/bin/sdfdump/sdfdump.cpp:522
    #26 0x7f3229f3d0b2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x270b2)
    #27 0x55585e8186bd in _start (./USD-20.05/build/bin/sdfdump+0x2a6bd)

0x60c000042184 is located 4 bytes to the right of 128-byte region [0x60c000042100,0x60c000042180)
allocated by thread T0 here:
    #0 0x7f322d1d7947 in operator new(unsigned long) (/lib/x86_64-linux-gnu/libasan.so.5+0x10f947)
    #1 0x7f3222217906 in __gnu_cxx::new_allocator<pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::Field>::allocate(unsigned
long, void const*) /usr/include/c++/9/ext/new_allocator.h:114
    #2 0x7f3222217906 in std::allocator_traits<std::allocator<pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::Field>
>::allocate(std::allocator<pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::Field>&, unsigned long)
/usr/include/c++/9/bits/alloc_traits.h:444
    #3 0x7f3222217906 in std::_Vector_base<pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::Field,
std::allocator<pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::Field> >::_M_allocate(unsigned long)
/usr/include/c++/9/bits/stl_vector.h:343
    #4 0x7f3222217906 in std::vector<pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::Field,
std::allocator<pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::Field> >::_M_default_append(unsigned long)
/usr/include/c++/9/bits/vector.tcc:635
    #5 0x7f3223bc36a2  (./USD-20.05/build/lib/libusd.so+0x228a6a2)

SUMMARY: AddressSanitizer: heap-buffer-overflow ./USD-20.05/pxr/usd/usd/crateFile.h:651 in
pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::CrateFile::GetToken(pxrInternal_v0_20__pxrReserved__::Usd_CrateFile::TokenIndex) const
Shadow bytes around the buggy address:
  0x0c18800003e0: fd fd fd fd fd fd fd fa fa fa fa fa fa fa fa fa
  0x0c18800003f0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c1880000400: fa fa fa fa fa fa fa fa 00 00 00 00 00 00 00 00
  0x0c1880000410: 00 00 00 00 00 00 00 00 fa fa fa fa fa fa fa fa
  0x0c1880000420: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c1880000430:[fa]fa fa fa fa fa fa fa 00 00 00 00 00 00 00 00
  0x0c1880000440: 00 00 00 00 00 00 00 00 fa fa fa fa fa fa fa fa
  0x0c1880000450: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c1880000460: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c1880000470: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c1880000480: 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
==108654==ABORTING
```

TIMELINE

2020-07-01 - Vendor Disclosure

2020-11-12 - Public Release

CREDIT

Discovered by Aleksandar Nikolic of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2020-1032

NEXT REPORT TALOS-2020-1104