Signed integer overflow (undefined behavior) in undo_pxr24_impl may allow bounds-check bypass in PXR24 decompression

Moderate **cary-ilm** published **GHSA-q3v8-hw4m-59w5** yesterday

Package

openexr

Affected versions

3.2.0-3.2.6, 3.3.0-3.3.8, 3.4.0-3.4.8

Patched versions

3.2.7, 3.3.9, 3.4.9

Description

Summary

A signed integer overflow exists in `undo_pxr24_impl()` in `src/lib/OpenEXRCore/internal_pxr24.c` at line 377. The expression `(uint64_t)(w * 3)` computes `w * 3` as a signed 32-bit integer before casting to `uint64_t`. When `w` is large, this multiplication constitutes undefined behavior under the C standard. On tested builds (clang/gcc without sanitizers), two's-complement wraparound commonly occurs, and for specific values of `w` the wrapped result is a small positive integer, which may allow the subsequent bounds check to pass incorrectly. If the check is bypassed, the decoding loop proceeds to write pixel data through `dout`, potentially extending far beyond the allocated output buffer.

The integer overflow itself is confirmed and reproducible. Full exploitation (end-to-end bounds-check bypass followed by observable heap corruption) has not yet been demonstrated, as it requires a crafted `w` value to survive all upstream EXR header and chunk-size validations.

Details

Vulnerable file: `src/lib/OpenEXRCore/internal_pxr24.c`, lines 377 and 389
Function: `undo_pxr24_impl()`

```
/* internal_pxr24.c:377 — primary issue: bounds check */
if (nDec + (uint64_t) (w * 3) > outSize)
    return EXR_ERR_CORRUPT_CHUNK;
//          ^^^^^^^^
//  w * 3 evaluated as signed int32 before cast.
//  For w = 1431655766 (0x55555556):
//    w * 3 = 4,294,967,298 → int32 overflow → wraps to 2
//    (uint64_t)(2) = 2
//    check: 0 + 2 > outSize  →  passes for any outSize > 2
//    loop then executes w iterations, writing 4*w bytes through dout

for (int x = 0; x < w; ++x)
{
    uint32_t diff = ( ((uint32_t)(*(ptr[0]++)) << 24) |
                      ((uint32_t)(*(ptr[1]++)) << 16) |
                      ((uint32_t)(*(ptr[2]++)) <<  8) );
    pixel += diff;
    unaligned_store32(dout, pixel);   // ← potential OOB write if check bypassed
    ++dout;
}

/* internal_pxr24.c:389 — secondary issue: accounting corruption */
nDec += (uint64_t) (w * 3);
//  same overflow; corrupts the decoded-byte counter for subsequent iterations
```

Section 1 — Confirmed: signed integer overflow (UB)

UBSan flags the multiplication as undefined behavior. Crash log reproduced on OpenEXR **v3.4.7** and on `main` at commit `c48f23ce`:

```
/src/lib/OpenEXRCore/internal_pxr24.c:377:46:
runtime error: signed integer overflow: 736549864 * 3 cannot be represented in type
'int'
    #0 in undo_pxr24_impl          internal_pxr24.c:377:46
    #1 in internal_exr_undo_pxr24  internal_pxr24.c:419:12
    #2 in exr_uncompress_chunk      compression.c:542:14
    #3 in exr_decoding_run          decoding.c:580:14

SUMMARY: UndefinedBehaviorSanitizer: undefined-behavior
         internal_pxr24.c:377:46
```

Note: for `w = 736549864`, the two's-complement result of `w * 3` is a large negative value, which when cast to `uint64_t` produces a value far exceeding `outSize`, so the check correctly rejects the chunk in this case. The UBSan report proves undefined behavior occurs; whether the specific wraparound value causes a bypass depends on the exact `w` supplied.

Section 2 — Theoretical: bounds-check bypass on two's-complement builds

For `w = 1431655766` (0x55555556), tested clang and gcc builds without sanitizers exhibit two's-complement wraparound, producing:

```
w * 3 (int32 wraparound) = 2
(uint64_t)(2)            = 2
```

In this case the check `nDec + 2 > outSize` passes for any `outSize > 2`, which is true under normal conditions. The loop then executes `w` iterations and writes `4 * w` bytes through `dout`, potentially extending far beyond the destination buffer. Whether `w = 1431655766` can reach this code path through all prior EXR header and chunk-size validations has not been confirmed.

Suggested fix:

```
// Before (vulnerable)
if (nDec + (uint64_t)(w * 3) > outSize) ...
nDec += (uint64_t)(w * 3);

// After (safe — cast before multiply)
if (nDec + (uint64_t)w * 3 > outSize) ...
nDec += (uint64_t)w * 3;
```

The same pattern must be fixed on both lines (377 and 389). Identical expressions exist for the HALF and UINT pixel-type branches in the same function and should be audited for the same issue.

PoC

Requirements: OpenEXR v3.4.7, clang with `-fsanitize=address,undefined`

```
git clone https://github.com/AcademySoftwareFoundation/openexr.git
cd openexr && git checkout v3.4.7

cmake -B build \
  -DCMAKE_C_FLAGS="-fsanitize=address,undefined -g -O1" \
  -DCMAKE_CXX_FLAGS="-fsanitize=address,undefined -g -O1" \
  -DBUILD_SHARED_LIBS=OFF -DBUILD_TESTING=OFF
cmake --build build --target OpenEXRCore -j$(nproc)

clang -fsanitize=address,undefined -g -O1 \
  -I openexr/src/lib/OpenEXRCore \
  -I build/cmake/OpenEXRCore -I build/cmake \
  -I build/_deps/imath-build/config \
  poc.c build/src/lib/OpenEXRCore/libOpenEXRCore-3_4.a \
  build/external/OpenJPH/src/core/libopenjph.a \
  -lz -lm -lstdc++ -lubsan -o poc
```

```
ASAN_OPTIONS="halt_on_error=1:print_stacktrace=1" \
UBSAN_OPTIONS="print_stacktrace=1:halt_on_error=1" \
./poc crash.exr
```

Trigger conditions:

- EXR file using PXR24 compression
- Channel with `FLOAT` pixel type
- Crafted `dataWindow` producing a channel `width` value where `w * 3` overflows signed int32

Impact

Any application that decodes untrusted PXR24-compressed EXR files is affected.

- **Confirmed — Denial of Service**: immediate crash under sanitized builds. Reproducible with the attached PoC on v3.4.7.
- **Theoretical — Heap Out-of-Bounds Write**: if a crafted `w` value survives EXR parsing validation and satisfies the wraparound condition, the bounds check is bypassed and the decoder writes `4 * w` bytes through `dout`, potentially extending beyond the allocated buffer. The written data is derived from attacker-controlled EXR pixel content. This could lead to heap corruption with possible code execution impact depending on allocator behavior and memory layout. Full exploitation has not been demonstrated.

Confirmed affected versions: v3.4.7 and `main` at commit `c48f23ce`.

Severity

Moderate 5.9 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	High
Privileges required	None
User interaction	Required
Scope	Unchanged
Confidentiality	None
Integrity	Low
Availability	High

Learn more about base metrics

CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:N/I:L/A:H

CVE ID

CVE-2026-34380

Weaknesses

- CWE-190
- CWE-787

Credits

pwn2woot Reporter