Heap information disclosure in PXR24 decompression via unchecked decompressed size (undo_pxr24_impl)

High **cary-ilm** published **GHSA-vc68-257w-m432** last week

Package

openexr

Affected versions

3.2.0-3.2.6, 3.3.0-3.3.8, 3.4.0–3.4.7

Patched versions

3.2.7, 3.3.9, 3.4.8

Description

Summary

The PXR24 decompression function undo_pxr24_impl in OpenEXR (internal_pxr24.c) ignores the actual decompressed size (outSize) returned by exr_uncompress_buffer() and instead reads from the scratch buffer based solely on the expected size (uncompressed_size) derived from the header metadata.

Additionally, exr_uncompress_buffer() (compression.c:202) treats LIBDEFLATE_SHORT_OUTPUT (where the compressed stream decompresses to fewer bytes than expected) as a successful result rather than an error.

When these two issues are combined, an attacker can craft a PXR24 EXR file containing a valid but truncated zlib stream. As a result, the decoder reads uninitialized heap memory and incorporates it into the output pixel data.

Details

This issue occurs due to the combination of two flaws.

1. compression.c:202–205 — LIBDEFLATE_SHORT_OUTPUT treated as success

```
else if (res == LIBDEFLATE_SHORT_OUTPUT)
{
```

```
    /* TODO: is this an error? */
    return EXR_ERR_SUCCESS;
}
```

libdeflate_zlib_decompress_ex() returns LIBDEFLATE_SHORT_OUTPUT when the compressed stream is successfully decompressed but the resulting output size is smaller than the provided output buffer size. In this case, the actual number of decompressed bytes is written to actual_out. However, the function does not treat this condition as an error and instead returns success.

2. internal_pxr24.c:279–287 — outSize return value ignored

```
rstat = exr_uncompress_buffer(
    decode->context, compressed_data, comp_buf_size,
    scratch_data, scratch_size, &outSize);   // outSize = actual bytes written

if (rstat != EXR_ERR_SUCCESS) return rstat;

// outSize is never referenced afterwards.
// The loop below reads the entire scratch_data buffer based on
// uncompressed_size (the header-derived expected size).
for (int y = 0; y < decode->chunk.height; ++y) { ... }
```

After exr_uncompress_buffer() returns success, the code does not verify whether the actual decompressed size (outSize) matches the expected size (uncompressed_size). The subsequent byte-plane reconstruction loop reads from the scratch buffer up to uncompressed_size bytes. As a result, the region between outSize and uncompressed_size consists of uninitialized heap memory, which is then read by the decoder.

Affected component

- src/lib/OpenEXRCore/internal_pxr24.c — undo_pxr24_impl() (line 261–399)
- src/lib/OpenEXRCore/compression.c — exr_uncompress_buffer() (line 202–205)

PoC

Please refer to the atta
poc.zip
ched archive file and proceed after extracting it.

1. git clone https://github.com/AcademySoftwareFoundation/openexr.git
2. mv poc openexr/
3. cd openexr
4. docker build -f poc/Dockerfile -t pxr24-poc .
5. docker run --rm pxr24-poc

```
~/study/poc/openexr/openexr main* 22s ❯ docker run --rm pxr24-poc
decode   : OK
expected : 4096 B  |  actual decompressed : 256 B
leaked   : 3840 / 3840 bytes non-zero in uninitialised region

hex @ offset 256 (heap leak):
be be be be 7c 7d 7d 7d 3a 3c 3c 3c f8 fa fa fa b6 b9 b9 b9 74 78 78 78 32 37 37 37 f0 f5 f5 f5

VULNERABILITY CONFIRMED — uninitialised heap data in pixel output
```

Impact

- Sensitive information from heap memory may be leaked through the decoded pixel data (information disclosure).
  Trigger Condition: Occurs under default settings; simply reading a malicious EXR file is sufficient to trigger the issue, without any user interaction.

Severity

High

CVE ID

CVE-2026-34543

Weaknesses

▶ CWE-908