Signed integer overflow in generic_unpack() when parsing EXR files with crafted negative dataWindow.min.x

Moderate **cary-ilm** published **GHSA-v76p-4qvv-vh4g** yesterday

Package

openexr

Affected versions	Patched versions
3.4.0-3.4.8	3.4.9

Description

Summary

A missing bounds check on the dataWindow attribute in EXR file headers allows an attacker to trigger a signed integer overflow in generic_unpack(). By setting dataWindow.min.x to a large negative value, OpenEXRCore computes an enormous image width, which is later used in a signed integer multiplication that overflows, causing the process to terminate with SIGILL via UBSan.

Details

When parsing a scanline EXR file, the image width is computed from the dataWindow attribute read directly from the file header:

width = dataWindow.max.x - dataWindow.min.x + 1

No validation is performed on dataWindow.min.x. By setting it to a large negative value (e.g. -1,073,741,804), the resulting width becomes approximately 1 billion pixels.

This enormous width value is stored in the channel info struct and later used in generic_unpack() at unpack.c:1278:

```
// unpack.c:1278
// w = 1,073,742,002 (int), bpc = 2 (int)
srcbuffer += w * bpc;
// 1,073,742,002 × 2 = 2,147,484,004 > INT32_MAX (2,147,483,647)
// → signed integer overflow → UBSan triggers SIGILL
```

The crash manifests differently from the OOM case because on systems with memory overcommit enabled, the large allocation (~6 GB) succeeds, allowing execution to reach the unpack stage where the overflow occurs.

Crash details observed under GDB:

- dataWindow: min.x = -1,073,741,804, max.x = 197
- chunk.width = 1,073,742,002
- chunk.unpacked_size = 6,442,452,012 (~6 GB)
- crash at generic_unpack() unpack.c:1278

PoC

1. Take any valid single-part scanline EXR file with NONE compression and two channels (HALF + FLOAT).

2. Modify the dataWindow.min.x field in the file header to a large negative value (e.g. 0xC0000014 = -1,073,741,804).

3. Feed the crafted file to any application using OpenEXRCore to parse EXR files.

Reproducing with the fuzzer harness:

```
$ clang -fsanitize=address,undefined -o harness fuzz_exr_pipeline.c -lopenexr
$ ./harness < poc.exr
```

Expected output:

```
Program received signal SIGILL, Illegal instruction.
generic_unpack (decode=...) at unpack.c:1278
1278 srcbuffer += w * bpc;
```

Stack trace:

```
#0 generic_unpack unpack.c:1278
#1 exr_decoding_run decoding.c:664
#2 decode_scanline_part harness.c:295
#3 main
```

Impact

Any application using OpenEXRCore to parse untrusted EXR files is affected. An attacker can craft a malicious EXR file and cause the target process to terminate due to a signed integer overflow, resulting in a denial of service. No authentication or special privileges are required — supplying the crafted file is sufficient to trigger the crash.

Affected software includes but is not limited to image editors, 3D renderers, compositing tools, and media processing pipelines that rely on OpenEXRCore for EXR file parsing.

Severity

Moderate 6.5 / 10

CVSS v3 base metrics

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

Learn more about base metrics

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

CVE ID

CVE-2026-34378

Weaknesses

- CWE-20
- CWE-190

Credits

 **pwn2woot** Reporter