Signed 32-bit Overflow in PIZ Decoder Leads to OOB Read/Write

High **cary-ilm** published **GHSA-588r-cr5c-w6hf** yesterday

Package

No package listed

Affected versions

3.1.0-3.1.13, 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

`internal_exr_undo_piz()` advances the working wavelet pointer with signed 32-bit arithmetic:

```
wavbuf += nx * ny * wcount;
```

Because `nx`, `ny`, and `wcount` are `int`, a crafted EXR file can make this product overflow and wrap. The next channel then decodes from an incorrect address. The wavelet decode path operates in place, so this yields both out-of-bounds reads and out-of-bounds writes.

Tested on commit `7820b7e`

Technical Details

The vulnerable decode path is:

1. `internal_exr_undo_piz()` sets `wavbuf = decode->scratch_buffer_1`.
2. For each channel, it calls `wav_2D_decode (wavbuf + j, ...)`.
3. It then advances `wavbuf` with `wavbuf += nx * ny * wcount`.

The overflow happens in step 3. Once `wavbuf` is wrapped, the next channel's wavelet decode runs on the wrong address.

In the 14-bit wavelet path, `wdec14_4()` first reads:

- `*px`
- `*p10`
- `*p01`
- `*p11`

and then writes back to the same locations:

- `*px  = ...`
- `*p01 = ...`
- `*p10 = ...`
- `*p11 = ...`

As a result, the bug is not just a crash-only invalid read. It is an out-of-bounds read/write condition.

Reproduction

piz_scanline_redzone.zip

Build `exrcheck` with ASAN and run:

```
❯ ./build-asan/bin/exrcheck /tmp/piz_scanline_redzone.exr
 file /tmp/piz_scanline_redzone.exr
/home/pop/sec/openexr/src/lib/OpenEXRCore/internal_piz.c:373:19: runtime error:
signed integer overflow: 134217724 * 32 cannot be represented in type 'int'
=================================================================
==1711239==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x7bedc3934700 at
pc 0x7bf1f100f498 bp 0x7ffe032d8f00 sp 0x7ffe032d8ef0
READ of size 2 at 0x7bedc3934700 thread T0
    #0 0x7bf1f100f497 in wdec14_4
/home/pop/sec/openexr/src/lib/OpenEXRCore/internal_piz.c:148
    #1 0x7bf1f100f497 in wav_2D_decode
/home/pop/sec/openexr/src/lib/OpenEXRCore/internal_piz.c:403
    #2 0x7bf1f100f497 in internal_exr_undo_piz
/home/pop/sec/openexr/src/lib/OpenEXRCore/internal_piz.c:727
    #3 0x7bf1f115b038 in exr_uncompress_chunk
/home/pop/sec/openexr/src/lib/OpenEXRCore/compression.c:546
    #4 0x7bf1f1161168 in exr_decoding_run
/home/pop/sec/openexr/src/lib/OpenEXRCore/decoding.c:580
    #5 0x7bf1f2a71add in run_decode
/home/pop/sec/openexr/src/lib/OpenEXR/ImfScanLineInputFile.cpp:586
    #6 0x7bf1f2a83dc4 in
Imf_4_0::ScanLineInputFile::Data::readPixels(Imf_4_0::FrameBuffer const&, int, int)
/home/pop/sec/openexr/src/lib/OpenEXR/ImfScanLineInputFile.cpp:500
    #7 0x7bf1f28c6a81 in Imf_4_0::InputFile::Data::readPixels(int, int)
/home/pop/sec/openexr/src/lib/OpenEXR/ImfInputFile.cpp:458
```

```
    #8 0x7bf1f3bfe2dc in readScanline<Imf_4_0::InputPart>
/home/pop/sec/openexr/src/lib/OpenEXRUtil/ImfCheckFile.cpp:239
    #9 0x7bf1f3c05b04 in readMultiPart
/home/pop/sec/openexr/src/lib/OpenEXRUtil/ImfCheckFile.cpp:905
    #10 0x7bf1f3c126fd in runChecks<char const*>
/home/pop/sec/openexr/src/lib/OpenEXRUtil/ImfCheckFile.cpp:1171
    #11 0x7bf1f3c146b9 in Imf_4_0::checkOpenEXRFile(char const*, bool, bool, bool)
/home/pop/sec/openexr/src/lib/OpenEXRUtil/ImfCheckFile.cpp:1835
    #12 0x5d9675fce8f8 in exrCheck(char const*, bool, bool, bool, bool)
/home/pop/sec/openexr/src/bin/exrcheck/main.cpp:96
    #13 0x5d9675fcb2b1 in main /home/pop/sec/openexr/src/bin/exrcheck/main.cpp:164
    #14 0x7bf1efe2a1c9 in __libc_start_call_main
../sysdeps/nptl/libc_start_call_main.h:58
    #15 0x7bf1efe2a28a in __libc_start_main_impl ../csu/libc-start.c:360
    #16 0x5d9675fcc844 in _start (/home/pop/sec/openexr/build-
asan/bin/exrcheck+0xe844) (BuildId: 087c972343a5372940c42c0a2e7bce4a84288aec)

0x7bedc3934700 is located 256 bytes before 8590720784-byte region
[0x7bedc3934800,0x7befc39f4710)
allocated by thread T0 here:
    #0 0x7bf1f40fd9c7 in malloc
../../../../src/libsanitizer/asan/asan_malloc_linux.cpp:69
    #1 0x7bf1f115883e in internal_decode_alloc_buffer
/home/pop/sec/openexr/src/lib/OpenEXRCore/coding.c:256
    #2 0x7bf1f100da97 in internal_exr_undo_piz
/home/pop/sec/openexr/src/lib/OpenEXRCore/internal_piz.c:643
    #3 0x7bf1f115b038 in exr_uncompress_chunk
/home/pop/sec/openexr/src/lib/OpenEXRCore/compression.c:546
    #4 0x7bf1f1161168 in exr_decoding_run
/home/pop/sec/openexr/src/lib/OpenEXRCore/decoding.c:580
    #5 0x7bf1f2a71add in run_decode
/home/pop/sec/openexr/src/lib/OpenEXR/ImfScanLineInputFile.cpp:586
    #6 0x7bf1f2a83dc4 in
Imf_4_0::ScanLineInputFile::Data::readPixels(Imf_4_0::FrameBuffer const&, int, int)
/home/pop/sec/openexr/src/lib/OpenEXR/ImfScanLineInputFile.cpp:500
    #7 0x7bf1f28c6a81 in Imf_4_0::InputFile::Data::readPixels(int, int)
/home/pop/sec/openexr/src/lib/OpenEXR/ImfInputFile.cpp:458
    #8 0x7bf1f3bfe2dc in readScanline<Imf_4_0::InputPart>
/home/pop/sec/openexr/src/lib/OpenEXRUtil/ImfCheckFile.cpp:239
    #9 0x7bf1f3c05b04 in readMultiPart
/home/pop/sec/openexr/src/lib/OpenEXRUtil/ImfCheckFile.cpp:905
    #10 0x7bf1f3c126fd in runChecks<char const*>
/home/pop/sec/openexr/src/lib/OpenEXRUtil/ImfCheckFile.cpp:1171
    #11 0x7bf1f3c146b9 in Imf_4_0::checkOpenEXRFile(char const*, bool, bool, bool)
/home/pop/sec/openexr/src/lib/OpenEXRUtil/ImfCheckFile.cpp:1835
    #12 0x5d9675fce8f8 in exrCheck(char const*, bool, bool, bool, bool)
/home/pop/sec/openexr/src/bin/exrcheck/main.cpp:96
    #13 0x5d9675fcb2b1 in main /home/pop/sec/openexr/src/bin/exrcheck/main.cpp:164
    #14 0x7bf1efe2a1c9 in __libc_start_call_main
../sysdeps/nptl/libc_start_call_main.h:58
    #15 0x7bf1efe2a28a in __libc_start_main_impl ../csu/libc-start.c:360
    #16 0x5d9675fcc844 in _start (/home/pop/sec/openexr/build-
asan/bin/exrcheck+0xe844) (BuildId: 087c972343a5372940c42c0a2e7bce4a84288aec)

SUMMARY: AddressSanitizer: heap-buffer-overflow
/home/pop/sec/openexr/src/lib/OpenEXRCore/internal_piz.c:148 in wdec14_4
```

```
Shadow bytes around the buggy address:
  0x7bedc3934480: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x7bedc3934500: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x7bedc3934580: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x7bedc3934600: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x7bedc3934680: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x7bedc3934700:[fa]fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x7bedc3934780: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x7bedc3934800: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x7bedc3934880: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x7bedc3934900: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x7bedc3934980: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
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
==1711239==ABORTING
```

To prove this is both READ and WRITE, we can also `memcheck` against non-ASAN release build:

```
valgrind --tool=memcheck --leak-check=no --track-origins=no \
  --error-limit=no --num-callers=20 \
  ./build-relwithdebinfo/bin/exrcheck /tmp/piz_scanline_redzone.exr
```

Observed result:

- `Invalid read of size 2` at internal_piz.c:150
- `Invalid write of size 2` at internal_piz.c:171

This confirms the bug is an OOB read/write, not only a read-first crash.

Redzone-Oriented File

- width: `67108862`
- height: `32`
- channel A: `FLOAT`, sampling `1 x 1`

- channel B: `HALF`, sampling `33554431 x 16`

This makes:

```
width * 32 * 2 = 4294967168
```

which wraps signed 32-bit arithmetic to `-128`.

That places the next `wavbuf` access just before the allocated buffer, producing a clean heap-overflow report.

Impact

A crafted EXR file can trigger out-of-bounds memory access during PIZ decompression. The primitive includes both invalid reads and invalid writes. Depending on allocator layout and surrounding memory, this could lead to process crash, memory corruption, or potentially stronger exploitation outcomes.

Recommended Fix

- compute channel span in 64-bit arithmetic
- reject any overflow in `nx * ny * wcount`
- validate cumulative per-channel decoded footprint against `outsz` before wavelet decode
- fail decompression if channel-derived layout does not exactly fit the decompression buffer

---

Found by: Quang Luong of Calif.io

Severity

High 8.6 / 10

CVSS v4 base metrics

Exploitability Metrics

Attack Vector	Local
Attack Complexity	Low
Attack Requirements	None
Privileges Required	None
User interaction	None

Vulnerable System Impact Metrics

Confidentiality	High

Integrity	High
Availability	High
Subsequent System Impact Metrics	
Confidentiality	None
Integrity	None
Availability	None

Learn more about base metrics

CVSS:4.0/AV:L/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N

CVE ID

CVE-2026-34588

Weaknesses

- CWE-125
- CWE-190
- CWE-787

Credits

quangIO Reporter