HTJ2K Signed Integer Overflow in ht_undo_impl()

Moderate **cary-ilm** published **GHSA-r3mr-mx8q-jcw5** 2 days ago

Package

openexr

Affected versions

3.4.0–3.4.9

Patched versions

3.4.10

Description

Summary

A signed integer overflow vulnerability exists in OpenEXR's HTJ2K (High-Throughput JPEG 2000) decompression path. The `ht_undo_impl()` function in `src/lib/OpenEXRCore/internal_ht.cpp` accumulates a bytes-per-line value (`bpl`) using a 32-bit signed integer with no overflow guard. A crafted EXR file with 16,385 FLOAT channels at the HTJ2K maximum width of 32,767 causes `bpl` to overflow `INT_MAX`, producing undefined behavior confirmed by UBSan. On an allocator-permissive host where the required ~64 GB allocation succeeds, the wrapped negative `bpl` value would subsequently be used as a per-scanline pointer advance, which would produce a heap out-of-bounds write. On a memory-constrained host, the allocation fails before `ht_undo_impl()` is entered (see OOM note in Impact section).

This is the **second distinct integer overflow** in `ht_undo_impl()`. CVE-2026-34545 (GHSA-ghfj-fx47-wg97, CVSS 8.4, fixed in v3.4.7) addressed a different overflow in the same function — the `int16_t p` pixel-loop counter at line ~302 that overflows when iterating over channels whose `width` exceeds 32,767. **The CVE-2026-34545 fix did not touch the `int bpl` accumulator at line 211, which is the subject of this advisory.** The `bpl` accumulator was also not addressed by any of the 8 advisories in the 2026-04-05 v3.4.9 release batch (see "Relationship to CVE-2026-34545" section below for the side-by-side comparison).

This finding is structurally identical to **CVE-2026-34588** (PIZ `wcount*nx` overflow in `internal_piz.c`) and should be remediated with the same pattern. The CVE-2026-34588 fix did not touch `internal_ht.cpp` — confirmed via code review of the v3.4.9 codebase (see "Relationship to CVE-2026-34588" section below).

Vulnerability class: CWE-190 (Integer Overflow or Wraparound); CWE-680 (Integer Overflow to Buffer Overflow) as secondary; CWE-787 (Out-of-Bounds Write) as allocator-dependent downstream consequence.

Affected Versions

Version range	Status
3.4.0 – 3.4.8	Expected vulnerable (not directly tested — see note below)
3.4.9	**Confirmed vulnerable** (UBSan on Ubuntu 22.04, clang 14)
< 3.4.0	Not affected (HTJ2K compression not present)

Version range note: Only 3.4.9 was directly tested. The `int bpl` declaration has been present in `internal_ht.cpp` since the file was introduced with HTJ2K support in OpenEXR 3.4.0 (PR #2291 — OpenJPH integration). Based on code review, the vulnerable accumulation loop is unchanged from introduction through 3.4.9. Versions 3.4.0–3.4.8 are expected to be affected but were not directly tested.

HTJ2K support was introduced in OpenEXR 3.4.0 via PR #2291 (OpenJPH integration). That same PR introduced the maximum HTJ2K image width of 32,767, which sets the overflow threshold within reach of the channel-count range accepted by the decoder.

Vulnerability Details

File: `src/lib/OpenEXRCore/internal_ht.cpp`
Function: `ht_undo_impl()`
Line: 211 (declaration), 213–220 (accumulation loop), 215 (overflow point)

Vulnerable code (3.4.9):

```
// internal_ht.cpp:211
int  bpl       = 0;
bool is_planar = false;
for (int16_t c = 0; c < decode->channel_count; c++)
{
    bpl +=                                              // line 215 — overflow here
        decode->channels[c].bytes_per_element * decode->channels[c].width;
    ...
}
cs.set_planar (is_planar);
```

```
cs.create ();
// ... later ...
line_pixels += bpl;                                    // line 319 — on allocator-permi
                                                       // host, negative stride would p
                                                       // heap OOB write (not demonstra
```

Root cause

`bpl` is declared as `int` (signed 32-bit). With:

- `channel_count = 16385`
- `bytes_per_element = 4` (FLOAT)
- `width = 32767`

The correct sum is `16385 * 4 * 32767 = 2,147,549,180`, which exceeds `INT_MAX = 2,147,483,647` by 65,533 bytes. The final accumulation step adds `131,068` to the partial sum `2,147,418,112`, wrapping to a negative value. This is confirmed undefined behavior under the C++ standard.

UBSan output (confirmed via direct harness, Docker lab, Ubuntu 22.04, clang 14):

```
/openexr-src/src/lib/OpenEXRCore/internal_ht.cpp:215:13: runtime error:
signed integer overflow: 2147418112 + 131068 cannot be represented in type 'int'
```

In a non-sanitized release build (`-DNDEBUG` disables `assert()`), the wrapped negative `bpl` value would be used at line 319 as `line_pixels += bpl`, advancing the output buffer pointer backward by approximately 2 GB per scanline. **This out-of-bounds write was not directly observed** — it is a code-review inference conditional on the ~64 GB uncompressed buffer allocation succeeding. On a memory-constrained host, the allocation fails before this point (see Impact section).

Taint path

```
Attacker EXR file (crafted chlist: channel_count=16385, FLOAT, width=32767)
  → parse_header.c (no channel count cap)
  → channel_list.c (no num_channels upper bound — only rejects <= 0)
  → decode->channel_count / decode->channels[c].bytes_per_element / decode-
>channels[c].width
  → internal_ht.cpp:213–220 bpl accumulation (signed int, no overflow guard)
  → UBSan: signed integer overflow at line 215 (confirmed)
  → [allocator-permissive host only] internal_ht.cpp:319 line_pixels += bpl
    → heap OOB write in uncompressed_data (not demonstrated)
```

Channel count validation (channel_list.c / validation.c)

Channel count validation in `channel_list.c` and `validation.c` (reviewed by static analysis on 3.4.9) imposes no upper bound on the number of channels in a file. The sole bounds check in `validate_channel_list()` (validation.c:413) is:

```
if (channels->num_channels <= 0)
    return f->report_error(f, EXR_ERR_FILE_BAD_HEADER, "At least one channel requi...);
```

No `MAX_CHANNEL` constant or equivalent cap exists on the decode dispatch path. A 16,385-channel chlist passes this check. **Note:** This was confirmed by static code review of the 3.4.9 source. The 16,385-channel file was not observed to complete the full header validation path dynamically (the exrcheck reproducer hits an allocator OOM before that point — see OOM note below); however, the static review of the relevant validation functions is unambiguous.

Relationship to CVE-2026-34588

CVE-2026-34588 is the same class of defect in the PIZ decompressor. The fix applied in `internal_piz.c` at lines 572 and 730:

```
wcount = (int)(curc->bytes_per_element / 2);
if (wcount > 0 && nx > INT_MAX / wcount)
    return EXR_ERR_CORRUPT_CHUNK;
```

The CVE-2026-34588 fix (PIZ `wcount*nx`) did not touch `internal_ht.cpp` — confirmed via code review of the v3.4.9 source. The single commit that comprises the v3.4.9 security release (`b5fa98a` in the local research repo) added `internal_ht.cpp` as a new file with the overflow-unguarded `int bpl` accumulator intact. No equivalent guard was applied to the HTJ2K path. The two bugs are structurally identical but independently present.

Relationship to CVE-2026-34545

CVE-2026-34545 (GHSA-ghfj-fx47-wg97, CVSS 8.4 High, fixed in v3.4.7 via PR #2291) is also an integer overflow in `ht_undo_impl()` in `internal_ht.cpp`. The CHANGES.md entry for v3.4.7 describes it as "Fix integer overflow in htj2k decode with width > 32767."

CVE-2026-34545 and this finding are in the **same function** but involve **different variables at different lines with different overflow conditions**:

	CVE-2026-34545	This advisory (EXR-HTJ2K-2026-001)
Variable	`int16_t p` (pixel loop counter)	`int bpl` (bytes-per-line accumulator)
Location	`internal_ht.cpp` lines ~302 and ~312	`internal_ht.cpp` line 211 (decl), line 215 (overflow)
Overflow condition	`int16_t` counter wraps at 32,767 when iterating over pixels in a	`int` accumulator wraps at INT_MAX when summing `bytes_per_element *`

	CVE-2026-34545	This advisory (EXR-HTJ2K-2026-001)
	wide channel	`width` across channels
Trigger	Single channel, `width ≥ 32768`	16,385 FLOAT channels, `width = 32767`
Fixed in	v3.4.7	Not fixed in any release through v3.4.9

Side-by-side code locations:

```
// internal_ht.cpp:211 — THIS ADVISORY — int bpl accumulator (NOT fixed by CVE-202 4
int  bpl       = 0;
bool is_planar = false;
for (int16_t c = 0; c < decode->channel_count; c++)
{
    bpl +=                                          // line 215 — overflow confirmed vi
        decode->channels[c].bytes_per_element * decode->channels[c].width;
    ...
}
// ... later ...
line_pixels += bpl;                                 // line 319 — OOB write on allocato

// internal_ht.cpp:~302 — CVE-2026-34545 — int16_t p pixel loop counter (fixed in v3.4.7
for (int32_t p = 0; p < decode->channels[file_c].width;   // post-fix: int32_t p
     p++)                                                  // pre-fix: int16_t p — wraps
{
    *channel_pixels++ = cur_line->i32[p];
}
```

The v3.4.7 fix (PR #2291) changed `int16_t p` to `int32_t p` at the pixel loop counter to prevent the per-pixel iteration counter from wrapping when `width > INT16_MAX`. It did not touch the `int bpl` accumulator at line 211. The current v3.4.9 source confirms this: line 211 still reads `int bpl = 0` with no overflow guard, and lines ~302/~312 show `int32_t p` as the (already-fixed) loop counter.

This finding is not a duplicate of CVE-2026-34545. The vendor's v3.4.7 audit that produced the CVE-2026-34545 fix was focused on the pixel-width overflow; the `bpl` multi-channel accumulation overflow at line 215 was a sibling defect in the same function that was not identified or addressed at that time, and remains present in v3.4.9.

Proof of Concept

Reachability note: `internal_exr_undo_ht()` — the C wrapper for `ht_undo_impl()` — is called from the public decode dispatch at `src/lib/OpenEXRCore/compression.c:466` for both `EXR_COMPRESSION_HTJ2K256` and `EXR_COMPRESSION_HTJ2K32` compression types. The harness calls `internal_exr_undo_ht()` directly as a minimization to bypass the unrelated ~64 GB allocation that obscures UBSan observation on the public path. The `decode->channels[]` state the harness sets up is identical to what the real call site populates from the file's chlist — no upstream validation in `channel_list.c` or `validation.c` caps channel count to a value that would block the trigger (see Channel count validation section above).

Primary reproducer: The enclosed `Dockerfile.fuzz` + `harness_bpl_overflow.cpp`. This is the definitive PoC that allows the vendor to observe the UBSan signed-integer-overflow at `internal_ht.cpp:215` directly.

Build and run instructions (harness — recommended for UBSan observation):

```
# Build with UBSan (see enclosed Dockerfile.fuzz for full environment)
clang++ -fsanitize=undefined -fno-sanitize-recover=undefined \
        -I/openexr-src/src/lib/OpenEXRCore \
        harness_bpl_overflow.cpp -o harness_bpl_overflow \
        -lOpenEXRCore -lOpenEXR
./harness_bpl_overflow htj2k_bpl_overflow.exr
```

Expected UBSan output:

```
runtime error: signed integer overflow: 2147418112 + 131068 cannot be represented
type 'int'
  at internal_ht.cpp:215:13
```

Alternate reproducer (exrcheck path — demonstrates OOM, not UBSan hit):

```
exrcheck -c htj2k_bpl_overflow.exr
```

Important: Running `exrcheck -c` on the attached file produces an ASan out-of-memory abort (~64 GB allocation request) rather than the UBSan signed-integer-overflow hit. The OOM occurs *before* `ht_undo_impl()` is entered, at the point where the decoder allocates the uncompressed output buffer. To observe the signed-overflow UB directly, use the enclosed `harness_bpl_overflow.cpp` compiled against the same UBSan build.

Attachment: `htj2k_bpl_overflow.exr` (459 KB — crafted file, see attached)
Attachment: `harness_bpl_overflow.cpp` (direct harness source — see attached)
Attachment: `Dockerfile.fuzz` (build environment — available on request)

Impact

Confirmed (direct harness, UBSan build): Signed integer overflow (CWE-190) in the `bpl` accumulator at `internal_ht.cpp:215`. This is undefined behavior under the C++ standard and will produce implementation-defined results on compilers that exploit signed overflow for optimization (GCC/Clang with `-O2` or higher can elide overflow checks under this assumption).

Allocator-dependent (not demonstrated): On a host where the ~64 GB allocation for the uncompressed output buffer succeeds (no cgroup/ulimit/overcommit restrictions), the wrapped negative `bpl` value would be used as a per-scanline pointer advance at line 319, writing outside the allocated buffer. Whether this is exploitable for memory corruption beyond a process crash depends on heap layout and allocator behavior and was not demonstrated in this engagement. All OOB-write impact claims are conditional on this allocation succeeding.

OOM note — not a claimed security impact: On a memory-constrained host, `exrcheck` aborts at the `malloc(~64GB)` call before entering `ht_undo_impl()`. Per OpenEXR's own SECURITY.md:
"Memory allocation failures caused by large image dimensions declared in file headers are not considered security vulnerabilities when the allocation size is proportional to the declared image dimensions." The 64 GB allocation is proportional to the declared dimensions (16385 channels × 32767 width × 4 bytes × scan height), so this OOM does not constitute a security-relevant DoS under OpenEXR's own policy. The claimed vulnerability is the integer overflow UB itself — not the OOM.

CVSS 3.1 (demonstrated impact):
`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L` — **Score: 5.3 (Medium)**

Rationale:

- **AV:N** — Any service that auto-decodes EXR files (e.g., OpenImageIO-linked asset ingest, thumbnailing pipelines) can receive a crafted file from the network.
- **AC:L** — No race condition or special configuration required; integer overflow is deterministic given the parameter values.
- **PR:N / UI:N** — File processing in a server context requires no privileges or user interaction.
- **S:U** — Impact is contained within the decoding process.
- **C:N / I:N** — No confidentiality or integrity impact demonstrated.
- **A:L** — On a realistic production host (with memory limits), the decoder encounters UB (signed overflow) on a crafted file; in a release build with compiler optimizations, this produces degraded/incorrect output or a process fault. The OOM-only outcome on memory-constrained hosts is explicitly excluded per OpenEXR's policy.

Ceiling vector (allocator-permissive host, OOB write reached):
`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:H` — **Score: 7.3 (High)**

This ceiling score reflects a host where the ~64 GB allocation succeeds and the OOB write at line 319 is reachable. This scenario was not demonstrated. It is provided as an upper bound for vendor remediation prioritization.

Note on patch validation: The proposed fix (see Suggested Fix section) was validated by code review only. Dynamic validation of the patched binary was blocked: the patched Docker image did not include the `harness_bpl_overflow.cpp` harness files, and both the vulnerable and patched `exrcheck` runs hit the same allocator OOM. The patched binary was never observed executing past the allocation gate. The fix is correct by code review (mirrors the CVE-2026-34588 pattern exactly), but dynamic patch validation has not been performed.

Suggested Fix

Minimal fix (mirrors CVE-2026-34588 pattern in `internal_piz.c`):

In `src/lib/OpenEXRCore/internal_ht.cpp`, change line 211:

```
// Before (vulnerable):
int  bpl       = 0;

// After (fixed):
int64_t bpl    = 0;
```

And add a post-loop guard before `cs.set_planar()` (line ~221):

```
if (bpl > (int64_t)INT_MAX) return EXR_ERR_CORRUPT_CHUNK;
cs.set_planar (is_planar);
```

This requires adding `#include <climits>` to the file (or using `std::numeric_limits<int32_t>::max()` from the existing `#include <limits>`).

A more comprehensive fix would also add a channel-count cap in `channel_list.c` / `validation.c` to reject files with unreasonably large numbers of channels, consistent with what OIIO already does via its `max_channels` limit.

##Attachment

Proof.zip
Proof.zip

Severity

Moderate 5.3 / 10

CVSS v3 base metrics

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

Learn more about base metrics

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

CVE ID

CVE-2026-39886

Weaknesses

▶ CWE-190

Credits


stanleytobias Reporter


Medoedus Reporter