Content Mark Survival Audit

ProofMark by Tayyir · Report PM-2026-0001 · Issued 2026-08-29 · Sample report, public

SECTION 01

Executive summary

One image was signed with C2PA Content Credentials and served over four delivery paths. **The credentials survived one of them.**

Assets checked	4 delivery paths, 1 source asset
Credentials intact	1 of 4
Credentials removed	3 of 4
Failure point	The image CDN, on every path — including the one where no transform was requested
Checked	2026-08-29, UTC
Method	c2patool 0.27.16, run against each delivered file

The finding that matters: path 02 asked the CDN for no transformation at all — no resize, no format change. The manifest was still gone. Credential loss here is not a side effect of resizing; the CDN re-encodes by default, and re-encoding discards the manifest.

SECTION 02

Scope and limits

What was checked. Four public URLs, each fetched over HTTPS and inspected with `c2patool 0.27.16`. A path is reported as *credentials present* only when c2patool exits successfully and returns an active manifest.

What was not checked. Whether the signing certificate chains to a publicly trusted root; whether any downstream consumer would accept the credentials; any asset other than the four listed.

Stated limitations of this report.

01 The signing certificate is a self-issued test CA. c2patool reports the origin manifest as *Valid* with the note *signing certificate untrusted*. This report measures manifest **survival**, not certificate trust — the two are independent, and an untrusted manifest that survives still proves the delivery path preserved it.

02 The CDN under test is wsrv.nl, a free public image CDN. A commercial CDN under your own configuration may behave differently — that is precisely why a per-customer audit exists.

03 This is Tayyir's own test asset on a public CDN. **It establishes nothing about any other party's pipeline.**

04 Byte sizes are recorded for completeness. They are not evidence: every verdict comes from c2patool run against the delivered file, never inferred from file size.

SECTION 03

Per-path results

#	DELIVERY PATH	HTTP	BYTES	CREDENTIALS
01	Direct from origin (no CDN)	200	238,584	**INTACT**
02	CDN, no transform (control)	200	159,398	**STRIPPED**

03	CDN, resize to 800px wide	200	41,624	STRIPPED
04	CDN, resize to 800px + WebP	200	16,108	STRIPPED

Every URL, and the exact tool output. Re-run any line below and you get this report's verdict for that path.

01 Direct from origin (no CDN)

```
https://tayyir.com/proof/asset-signed.jpg
$ c2patool <delivered file>
(active manifest returned; issuer Tayyir, alg ES256, validation_state Valid)
```

02 CDN, no transform (control)

```
https://wsrv.nl/?url=tayyir.com/proof/asset-signed.jpg&n=-1
$ c2patool <delivered file>
Error: No claim found
```

03 CDN, resize to 800px wide

```
https://wsrv.nl/?url=tayyir.com/proof/asset-signed.jpg&w=800&n=-1
$ c2patool <delivered file>
Error: No claim found
```

04 CDN, resize to 800px + WebP

```
https://wsrv.nl/?url=tayyir.com/proof/asset-signed.jpg&w=800&output=webp&n=-1
$ c2patool <delivered file>
Error: No claim found
```

SECTION 04

Root cause, and the fix for each case

Where the mark died. The asset left the origin signed (path 01) and arrived stripped on every path that passed through the CDN. The manifest is removed in delivery, not missing at generation. Path 02 isolates the cause: the CDN re-encodes even when no transformation is requested, and the JPEG re-encode does not carry the JUMBF box that holds the manifest.

Three ways to fix it, in the order most teams should try them:

01 **Exclude signed assets from the transform pipeline**

Serve provenance-bearing assets straight from origin, or mark them pass-through. Cheapest fix, and it costs you the CDN's optimisation on those assets only.

02 **Move to a remote manifest**

Store the manifest beside the asset rather than embedded in it, and reference it from the file. The credential then survives a re-encode that would have discarded an embedded manifest. Note the trade-off: a remote manifest is orphaned if the asset is copied away from its URL.

03 **Add a soft binding or a durable content credential**

A watermark-based binding recovers provenance even when the container cannot carry a manifest at all — the correct answer for formats and pipelines where metadata never survives.

SECTION 05

Questionnaire answers, ready to paste

Drafted for the questions procurement actually asks. Edit the bracketed parts; the rest stands as written.

Q. Do you apply machine-readable provenance (C2PA Content Credentials) to synthetic media you publish?

A. Yes. Assets are signed at generation with C2PA Content Credentials [tool and version]. Signing is performed by [system], and the signing certificate is [issuer].

Q. Have you verified that provenance survives your publishing pipeline?

A. Yes. As of [date] we verified [N] published URLs against the delivered file, not the source file. Results and method are recorded in a dated report retained on file, sealed with an RFC 3161 timestamp from a third-party authority.

Q. What happens when provenance does not survive delivery?

A. Each failure is traced to the delivery step responsible and remediated by one of: excluding the asset from transformation, moving to a remote manifest, or applying a durable content credential. Remediation status per asset is recorded in the same report.

SECTION 06

Cryptographic timestamp

This report is sealed with an RFC 3161 timestamp issued by **freetsa.org**, a third-party timestamping authority whose certificate chain is publicly downloadable — which is the point: you verify this without asking me for anything. The timestamp proves the document existed in this exact form at the stated time and has not been altered since. **It does not attest that the findings are correct** — that is what the URLs and commands in section 03 are for, and you can re-run every one of them without involving me.

Verify it yourself:

```
$ curl -O https://freetsa.org/files/cacert.pem
$ curl -O https://freetsa.org/files/tsa.crt
$ openssl ts -verify -data PM-2026-0001.pdf \
    -in PM-2026-0001.pdf.tsr \
    -CAfile cacert.pem -untrusted tsa.crt
Verification: OK
```

The token file `PM-2026-0001.pdf.tsr` is published beside this PDF at tayyir.com. Run the three commands above and openssl will tell you, without my involvement, whether this document is the one that was sealed.

ProofMark by Tayyir · hamza@tayyir.com · tayyir.com
Reads and verifies. Never signs. Never holds your keys.