p4-cache — Engineering Maturity Brief

A sales asset for buyers who ask "is this actually production-ready, or is it a hobby project with a marketing site?" The honest answer in numbers and citations.

TL;DR

p4-cache is a ~47K-line Rust workspace (six crates) that has been through three independent code audits and shipped remediation against all three. The workspace enforces compile-time defenses against entire classes of bugs (every unsafe operation requires explicit `unsafe { }` blocks; mutex guards across `.await` boundaries are a compile error). Workspace lints include `cast_sign_loss = "deny"`, `cast_possible_truncation = "deny"`, `cast_possible_wrap = "deny"`, and `must_use_candidate = "warn"`. The repo's quality gate (`scripts/gate.sh`, run on every change) includes `cargo audit` and `cargo deny check`.

This brief exists because most year-1 commercial software has zero audit trail; p4-cache has three. That fact is worth being explicit about.

By the numbers

Metric	Value	Note
Lines of Rust	~47K	Across six workspace crates (see below)
Workspace crates	6	`p4cache` (daemon + tools), `p4cache-trigger` (per-call archive trigger), `shared` (types + protocol), `license` (signing/verification), `license-tool` (vendor CLI), `integrity-stamp` (release-artifact stamping)
Shipped release binaries	2	`p4-cache` (long-resident daemon) and `p4cache-trigger` (statically-linked musl per-call archive trigger, ~587 KiB), plus `SHA256SUMS`. The operator tooling — checkpoint/journal verifier and upload-state reporter — lives inside `p4-cache` as the `verify` and `manifest summary` subcommands rather than as separate binaries. The vendor-side `p4-cache-license` and `p4cache-`

		`integrity-stamp` are build tools, never shipped to customers
Tests	~550	`#[test]` + `#[tokio::test]`
Test files	~42	Unit modules, integration tests, property tests, fuzz harnesses
Independent audits + review passes completed	4	`.audit/` (QA), `.codex-audit/` (revalidation), `.audit-verify/` (verifier binary), `REVIEW_FIXES.md` (2026-05-29 internal full-workspace code-review pass — 9 batches, all 7 HIGH-severity findings closed)
Open CRITICAL findings	0	"Data loss / RCE / auth bypass / secret leak / persisted state corruption / known-exploit CVE in direct dep"
Open HIGH engineering findings	0	All 7 HIGH-severity findings from the most recent (2026-05-29) review pass are closed across Batches 1–9 of `REVIEW_FIXES.md`. Prior HIGHs (`S-004` integrity-failed staging-path RAII, `S-verify-101` verifier argv hardening) remain closed; per-finding entries in `REVIEW_FIXES.md` and `.audit/6-synthesis/REPORT.md` cite the fix commit and the pinning test.
Open HIGH supply-chain advisories	0	`cargo audit` + `cargo deny` are green with zero vulnerabilities. The prior transitive HIGHs cleared on 2026-05-31/06-01: `rustls-webpki` 0.101.x (stale, pruned), `quinn-proto` (`cargo update` → 0.11.14), `rand` (→ 0.8.6/0.9.4), and `protobuf` (eliminated by dropping prometheus's unused `protobuf` feature). The two remaining accepted ignores are both *unmaintained*-status warnings (not vulnerabilities): `rustls-pemfile` (pulled transitively by `object_store`) and `bincode` (dev-only, via `iai-callgrind`).

`TODO`/`FIXME`/`HACK` in production code	0	None; the two former `p4cache/src/license/policy.rs` markers were resolved.
`panic!`/`unimplemented!`/`todo!` in non-test paths	3	All bug-on-impossible-state, not unhandled cases: 2 in the storage backend contract test for post-delete `head_object` divergence, 1 in `p4cache/src/license/load.rs` for an unparseable build-time `P4CACHE_BUILD_DATE` (build.rs ensures this never fires in shipped binaries).
`#[ignore]` tests	0	Every test runs every build
`unsafe` blocks in daemon	~58	Every one carrying a `// SAFETY:` comment; sites cover `dl_iterate_phdr` integrity self-hash, `clock_anchor` defense, license audit-log v2, SCM_RIGHTS fd-passing, and `SO_PEERCRED` plumbing
Stress / fallback shell harnesses	12	`p4_v2_read_stress.sh`, `p4_storage_benchmark.sh`, `p4_cache_read_stress.sh`, `p4_azure_nfs_fallback_test.sh`, `p4_commit_peak_hour_stress.sh`, `p4_read_stress_nocache.sh`, `p4_submit_storage_compare.sh`, `p4_gzip_parity.sh`, plus the restore-tuning sweeps (`p4_restore_chunk_sweep.sh`, `p4_restore_chunk_sweep_nfs.sh`, `p4_restore_threshold_sweep.sh`) and `p4_cache_vs_stock_ops.sh`
`cargo fuzz` targets	2	Trigger request handler (`fuzz_handle_request`), Perforce checkpoint journal parser (`fuzz_checkpoint_parser`)
ADRs documenting major decisions	3	0001 panic strategy, 0002 content-hash schema (the legacy `FileEntry` hash evolution that informed v2's MD5 archive-content verification and the narrow `archive_catalog`/`upload_manifest` `redb` stores), 0003

		peer-cred allowlists (all under `docs/ADR/`)
Direct Rust dependencies	~44	Reviewed; transitive advisories tracked in `rust/.cargo/audit.toml`
Rust edition	2024	Workspace-pinned
MSRV	1.88	Pinned in workspace `Cargo.toml`; let-chains in `if`/`while` are the current floor feature

Compile-time defenses

The workspace `[lints]` section enforces, *at compile time*:

Lint	Level	What it prevents
`unsafe_op_in_unsafe_fn`	deny	Rust 2024 idiom: every unsafe operation inside an `unsafe fn` body must sit inside an explicit `unsafe { ... }` block. Catches UB the outer signature used to hide.
`await_holding_lock`	deny	Holding a `MutexGuard` or `RwLockGuard` across an `.await` is a compile error. Prevents deadlock-under-load that would otherwise be a runtime bug.
`cast_sign_loss`	deny	Catches silent `i32 as u32` / `i64 as u64` / `isize as usize` sign-flip casts. Surviving same-width sign changes go through `.cast_unsigned()` as an explicit intent marker.
`cast_possible_truncation`	deny	Catches silent narrowing casts. Surviving sites use the `wrapping_as_*` helpers in `p4cache_shared` (intentional wrap) or `u32::try_from(x).expect("…")` (panic-on-bug at a boundary).
`cast_possible_wrap`	deny	Catches silent `u32 as i32` / `u64 as i64` wrap-around casts. Surviving same-width sign changes are made explicit at the call site.

`must_use_candidate`	warn	Catches new public functions that should advertise their return value as `#[must_use]`. The `CacheConfig` accessor wall carries a local `#[allow(...)]` so the lint stays useful for new methods without spamming on obvious getters.

`panic = "unwind"` is set in both `[profile.release]` and `[profile.dev]` so the per-connection `catch_unwind` guard in the trigger listener can convert a handler panic into a structured ERROR log without killing the daemon. The daemon is unwind-safe (no FFI boundaries on the hot path; tokio + redb are unwind-safe; `Drop` impls release locks/fds correctly).

These aren't documentation. They're enforced. A change that violates any of them fails the build and the quality gate.

Quality gates

`scripts/gate.sh` — the repo's gate, run on every change across the Linux / macOS / Windows dev machines — includes:

- `cargo fmt --check`
- `cargo clippy --release --all-targets --all-features -- -D warnings`
- `cargo test` (full suite against a live p4d when available)
- `cargo audit` (advisory scan)
- `cargo deny check advisories bans licenses sources` (supply-chain policy)
- MSRV floor build (`--locked` on exactly Rust 1.88)
- Binary size budget on the shipped artifact shapes (24 MiB daemon / 1 MiB musl trigger)
- `--full` adds: the 75% core coverage floor, `cargo fuzz` smoke runs on both fuzz targets, the deterministic callgrind perf gate, gzip/layout parity against a real p4d, and advisory-ignore hygiene
- On Windows the same script runs the cfg(windows) clippy wall plus the Windows unit + integration test set

Release artifacts are still built by a pipeline (`.github/workflows/release.yml`, tag-triggered): the two shipped binaries, `SHA256SUMS`, and a signed build-provenance attestation.

A change that introduces a new transitive advisory, an unapproved license, a banned dependency, or a fuzz crash fails the gate. The advisory ignore list in `rust/.cargo/audit.toml` documents each entry with a per-advisory reachability rationale (why it's not exploitable in this build).

Audit history in detail

Audit pass 1 — full-workspace QA audit (.audit/)

Six phases: inventory, quality, architecture, security, performance, tech debt.

Inventory: 23K LoC at audit time, dep graph mapped, workspace topology documented.

Quality (40 findings): Mostly per-crate cleanups. Notable: 729 pedantic clippy warnings catalogued (most cosmetic); ~85 `unwrap`/`expect` sites in production paths reviewed (mostly bounds-guarded but flagged for hardening). One flaky test (pre-existing, lock-file collision).

Architecture (17 findings): Several large modules identified as refactoring targets. Storage backend duplication of `temp + fsync + rename + assert_regular + fsync_parent` plumbing identified.

Security (25 findings): Transitive vulns in `rustls-webpki`, `aws-lc-sys`, `quinn-proto`. A large number of `unsafe` blocks lacked `// SAFETY:` comments. No fuzz targets. No deny.toml.

Performance (12 findings): 52 MiB default build; no benchmarks; double-pass disk I/O on restore (write-then-hash).

Tech debt (10 findings): Zero `TODO`/`FIXME`/`HACK` markers. Three `#[allow]`s, all justified. Four unused direct deps.

Audit pass 2 — codex re-audit (.codex-audit/)

Re-validation after pass-1 remediation landed. Deeper storage-SDK retry/perf review across S3, Azure, GCS. Doc and packaging drift review. Reproduced the automated gate spot-checks.

HIGH findings at the time of the pass — all now closed:

- `S-004`: Integrity-failed restores left rejected bytes at the live depot path. Closed by the staging-path RAII landing in the restore pipeline — the cache write only atomic-renames onto the live archive path after the content-hash check passes.
- `S-006`: cargo-audit gate was red on active transitive advisories. Closed by tightening `rust/.cargo/audit.toml` to per-advisory ignore entries with documented rationale, plus a `cargo update` for the items where an upstream fix existed (`rustls-webpki` to 0.103.13, `aws-lc-sys` to 0.40.0). Those former `S-001`/`S-003` items have since cleared entirely; the only remaining accepted ignores today are the unmaintained-status `rustls-pemfile` and the dev-only `bincode` (see the supply-chain advisory row above), and a nightly CI `advisory-hygiene` job fails if any accepted ignore goes stale.

Plus MEDIUMs clustered into argv hardening, logging unification, resource limits, OsString-clean argv, live-shelf cap, and architecture lift — all closed or partially closed and tracked in `.codex-audit/6-synthesis/REPORT.md`.

Audit pass 3 — verify-binary audit (.audit-verify/)

Targeted verification audit of the checkpoint verifier (now the `p4-cache verify` subcommand). Inventory complete; manual review of CLI argv handling, checkpoint/journal parser, source enumeration, state-dir lifecycle, backend exists-checking, verify runtime.

One HIGH finding (`S-verify-101`): depot path bytes flowed into `p4 verify` argv unfiltered. **Closed.** Two complementary defenses landed: `storage_row_from_fields` now

rejects rows whose `depot_path` does not start with `//` (with a structured WARN), and a parser test (`skips_row_whose_depot_path_does_not_start_with_doubled_slash`) pins the validator. The argv-`--`-terminator approach was tried and reverted after `p4 verify` rejected it; the `//` prefix validator is the defense in the shipping code.

Plus MEDIUMs, including secondary-error-swallowing in `BackendSet::PrimaryAndSecondary`, OsString-vs-String argv parsing, sequential backend probes, and unbounded `live_shelves` HashSet on long-lived servers.

Review pass 4 — full-workspace internal code review (REVIEW_FIXES.md, 2026-05-29)

A fan-out review with adversarial per-finding verification ran across the workspace, organized into 9 batches with atomic per-batch commits. Batches 7–9 alone produced **23 confirmed findings** (2 HIGH, 6 MEDIUM, 15 LOW), with 7 additional candidates verified as false positives and dropped. Across all batches the review recorded **7 HIGH-severity findings**. Severity is operator-impact: HIGH = data loss / privilege escalation / license bypass / daemon crash / audit-compliance gap.

HIGH-finding closure status (all 7 closed):

- **Archive layout path traversal (`archive_layout.rs:84`, Batch 2).** `archive_path` validated `depot_path` segments but passed `lbr_rev` directly into the revision filename component; slash, `..`, backslash, NUL, or absolute-like content could escape it. Closed.
- **`p4 storage` false-OK parse (`p4_storage.rs:442`, Batch 3).** An invalid `lbrRefCount` was silently coerced to `None` then defaulted to `0`, so malformed output could be read as safe-to-delete. Closed.
- **Checkpoint parser false-OK on schema mismatch (`parser.rs:321`, Batch 5).** `storage_row_from_fields` treated unexpected `db.rev*` field counts as `Ok(None)` after a warning, so corrupted/schema-changed rows could be skipped while the verifier still exited OK. Closed.
- **Checkpoint parser truncated-input false-OK (`parser.rs:659`, Batch 5).** On a `MAX_LINE_BYTES` cap hit, `send_storage_rows` logged, broke, and returned success, letting a truncated scan be recorded complete. Closed.
- **Checkpoint state-file durability (`state.rs:962`, Batch 5).** `append_specs_to_path` flushed target files but did not fsync them before the source-completion state was fsynced, so a crash could lose missing-target files while preserving the completed-source marker. Closed.
- **Capability-drop fail-closed misfire (`caps.rs:74`, Batch 7).** `PR_CAPBSET_DROP` needs `CAP_SETPCAP`, which the shipped unprivileged systemd unit doesn't grant, forcing `exit(1)`. Reordered so the effective/permitted capset clear (the real drop) stays fatal while the bounding-set drop is best-effort, since `NO_NEW_PRIVS` plus the capset clear already remove every elevation path. Closed.
- **License clock-rewind bypass (`clock_anchor.rs:138`, Batch 8).** The anchor overwrote its rollback baseline with the current `now` each start, so sub-tolerance rollbacks across many restarts walked the clock back without limit. Now a monotonic high-water mark `max(prior, now)`, with a cumulative-rollback regression test. Closed (applied with explicit user approval).

Remaining work (no HIGH): all batch findings were addressed; the single item originally marked **Deferred** — the `shared`-crate feature-gate (LOW, build-hygiene) — was implemented in the follow-up pass (2026-05-29), along with three other deferred items

now done (large trigger-write streaming, `object_store` runtime credential redaction, secondary-backend restore fallback). Per-batch dispositions, the rejected-false-positive list, and the verification log live in `REVIEW_FIXES.md`; "build + `cargo test -p p4cache` after every batch, no red tree" is the standing rule.

Remediation pattern

Each audit pass produces a synthesis report. The QA audit synthesis lives at `.audit/6-synthesis/REPORT.md`; review-pass findings and their per-batch closures are tracked in `REVIEW_FIXES.md` at the repo root. Highlights of what landed: supply chain (cargo update + deny.toml + gate); build size (a single `object_store` backend compiles all clouds — per-cloud builds retired — with the daemon at 13.4 MiB and the trigger at ~587 KiB after LTO for the shipped statically-linked musl build, ~470 KiB for the glibc-dynamic dev build); criterion benchmarks; SAFETY comment pass; storage trait contract test; fuzz harness; property tests; ADRs under `docs/ADR/`.

This is not unusual for a mature commercial product. It *is* unusual for a year-1 product.

What the audits did not find

Documented because their absence is informative:

- **No data-loss CRITICAL.** No bug, in any audit, where p4-cache could destroy customer depot data without operator action.
- **No RCE.** No bug, in any audit, where untrusted input becomes code execution.
- **No auth bypass.** The peer-credential allowlist mechanism (ADR-0003) was reviewed across both subsequent audits without bypass findings.
- **No secret leak.** `Redacted<T>` wrapper applied to all credential types; verified in audits 2 and 3.
- **No persisted state corruption path.** The two narrow `redb` stores — `archive_catalog` (a local digest cache of `(depot_path, lbr_rev) -> {md5, gzipped}`) and `upload_manifest` (the durable Dirty/Uploading/Clean async-upload state machine) — use redb's single-writer single-tx model, atomic at the storage layer; their record codecs are forward- and backward-tested. db.storage stays the replicated, authoritative digest source of truth.

The closest the audits came to CRITICAL was `S-001`/`S-002`/`S-003` in pass 1 — transitive vulns in `rustls-webpki`/`aws-lc-sys`/`quinn-proto`. None exploited under the threat model (all required attacker-on-the-wire). All resolved by `cargo update`.

What the audits are still working on

Honest list, current to the latest review-pass execution log:

High-priority, bounded: *None.* All 7 HIGH-severity findings from the 2026-05-29 internal review pass are closed across Batches 1–9 of `REVIEW_FIXES.md`. Prior-audit HIGHs (`S-004`, `Q-p4cache-006`, `S-verify-101`) also closed and pinned by regression tests.

Supply-chain HIGH: *None remaining.* The prior transitive HIGHs cleared on 2026-05-31/06-01:

- `S-001` — `rustls-webpki` 0.101.x: the legacy `aws-smithy-http-client` ladder that pinned it is gone; the entry was pruned as stale.
- `S-003` — `quinn-proto`: bumped 0.11.13 → 0.11.14 via `cargo update`; the QUIC-DoS advisory no longer fires.
- `protobuf` (uncontrolled recursion): eliminated by dropping `prometheus`'s unused `protobuf` exposition feature (`default-features = false`) — it left the dependency graph entirely.

The only accepted exceptions today are two *unmaintained*-status warnings (not vulnerabilities; no patched version exists): `rustls-pemfile`, pulled transitively by `object_store`, and `bincode`, dev-only via `iai-callgrind`. A nightly CI `advisory-hygiene` job fails if any accepted ignore goes stale, so the list cannot silently drift.

MED + LOW findings (from `REVIEW_FIXES.md`): all addressed across Batches 1–9, by area:

- Storage: async cleanup of temp files + expected-size check, per-chunk `to_vec` avoidance for multi-GB downloads.
- Checkpoint verifier: parser send-channel root-cause, runtime flush-on-error guard, runtime tracker fixes, checkpoint canonicalize bail, `prune_orphan_source_states` valid-filenames, shelf/CRLF validation, `progress.rs` rotation race + writer exit, runtime drop on submit error, sources strict suffix.
- Access log: `MSG_TRUNC` recv handling, spool re-validate on read, ES index allowlist, chmod-failure surfacing, 32-bit batch panic floor, Postgres `.expect` → `Transient` classification.
- Config + lifecycle + CLI (Batch 7) plus the LOW cleanup tail. The four originally-**Deferred** items — including the `shared`-crate feature-gate — were implemented in the 2026-05-29 follow-up pass.

Low-priority, hygiene:

- Remaining `cast_possible_truncation` allow-sites — `cast_possible_truncation` is a deny lint, so these carry a local `#[allow(...)]`; most are FFI-shaped (`socklen_t = u32`); replace with explicit `try_from` or `wrapping_as_*` helpers as touched.
- Duplicate crate versions in the dep graph — natural consequence of the cloud SDK matrix exposed through `object_store`; not a bug.

Recently landed engineering hygiene:

- Shared `download_parallel_to_temp` driver in `storage/mod.rs` — single driver for the parallel range-GET path.
- Shared `pwrite_chunk_at` and `plan_ranges` helpers.
- `write_atomic_with` helper in `checkpoint_verify/state.rs` — consolidates the temp/rename/parent-fsync skeleton shared between `write_json_atomic` and `write_specs_atomic`.

The remediation pattern from prior audits shows these typically close within weeks of someone picking them up. None block customer deployment; all are visible and tracked.

How to read this brief

If you're a *technical evaluator*, every claim here is verifiable in the source repo. The audit synthesis reports are in `.audit/6-synthesis/REPORT.md`, `.codex-audit/6-synthesis/REPORT.md`, and `.audit-verify/6-synthesis/REPORT.md` respectively. Per-phase findings are in the numbered subdirectories. ADRs are in `docs/ADR/`. The lint policy is in `rust/Cargo.toml` under `[workspace.lints]`. The quality gate is `scripts/gate.sh`; the release pipeline is `.github/workflows/release.yml`.

If you're a *commercial buyer*, the headline is: **this is software built by someone who treats it like software people pay for, not like a side project.** The audit infrastructure is the evidence. The remediation track record is the evidence. The compile-time defenses are the evidence.

Most products at this stage don't have this story to tell. The ones that do are the ones their customers stop worrying about.

Document control

- Version 1.0
- Maintained against repo HEAD
- Numbers verified by source grep at the date of last update
- Audit report citations link to the actual files in the source tree