POSTERN LABS · SECURITY AUDIT

◇ Postern Labs Products — Security Audit

Owned products: vault/keys/courier/hygiene/quarantine, messenger, Tauri desktop, web apps, enterprise, packaging & scripts

0 CRITICAL | 2 HIGH | 8 MEDIUM | 23 LOW

Scope: `gitlab.com/bloch-sis-group/bloch-sis-project` · 33 findings · Opus 4.8 auditors · 2026-07-15

Nature of this document. Internal, AI-assisted adversarial code review. It is **NOT** a third-party security audit and confers no assurance. Every finding was verified against source (file:line). The system is a disclosed **unaudited, pre-mainnet beta** running a relaxed **k=4** PoW that is openly documented as trivially forgeable — that known posture is not itself a finding. Fix criticals/highs before the corresponding feature is relied upon.

✓ **Remediation status (post-audit).** Every high and medium below has been FIXED in code and adversarially re-verified; the web-app fixes (including the gramota-kyc stored XSS) are DEPLOYED LIVE, and the desktop/crate fixes ship in the downloadable products. Two overclaims caught during fix-verification (a 'TEE-wrapped' key description and an 'audited' vault label) were corrected. Still UNAUDITED — not a substitute for the contracted third-party audit.

Verified sound / no defect found. First-party crates + desktop + apps + scripts audited across 17 scoped slices. The Tauri command surface, the PQ seal engine, and the file parsers were found consistently defensive (input validation, fail-closed, esc()/textContent); no critical.

Findings (33) — ranked by severity

HIGH Receive path badges homeserver-injected plaintext in an E2EE room as InternalE2ee (send path fails closed, receive path does not)

`src/matrix.rs:651`

Scenario. A room classifies InternalE2ee (m.room.encryption present, clean members). A malicious/compromised homeserver injects an unencrypted m.room.message into that room. matrix-sdk delivers it to the `on_text_message` handler with `encryption_info == None`. The handler (lines 651-677) computes `tier = Self::tier_of(&room) = InternalE2ee` (the room is still encrypted) and emits `ReceivedText { was_encrypted: false, tier: InternalE2ee, .. }`. Per the field contract at lines 281-283 ("Fail-closed tier of the room... UI MUST badge the message with it"), the UI badges this attacker-supplied plaintext as "E2EE (Postern)". This is the exact silent-downgrade / plaintext-spoof the SecurityTier wall exists to prevent, and it is triggerable by the untrusted homeserver that is squarely in this module's threat model. Note the asymmetry: the SEND path deliberately guards this (send_text lines 634-636 return MatrixError::SentUnencrypted when encryption_info.is_none()), but the RECEIVE path has no equivalent — an unencrypted event in an encrypted room is presented as trusted internal E2EE.

Fix. In `on_text_message`, reconcile per-message encryption with the room tier: if `encryption_info.is_none()` (was_encrypted == false) but `tier == SecurityTier::InternalE2ee`, downgrade the per-message tier to `SecurityTier::BridgedExternal` (or introduce a dedicated 'unauthenticated / unencrypted-in-encrypted-room' state) so a homeserver-injected plaintext event can never be badged InternalE2ee. Mirror the send path's fail-closed stance on the receive path rather than relying on every UI consumer to cross-check the raw `was_encrypted` flag against `tier`.

HIGH Stored XSS: license-server / D&B response written to innerHTML unescaped under script-src unsafe-inline

`apps/gramota-kyc/index.html:323`

Scenario. gramota-kyc CSP (line 16) is script-src unsafe-inline, so injected inline event handlers execute. The Step-1 handler POSTs /license/review with resolve:true and writes the response into innerHTML with no escaping: line 323-325 interpolates entity.legalName and entity.country (j.dnb.legalName / j.dnb.country, the D&B business-name lookup for the reviewed DUNS), line 320 interpolates j.error, and the Step-4 verdict render (lines 409-414) interpolates j.decision, j.loa, j.assurance and j.methods joined, plus pill(cls,dec). All backend-controlled. An applicant registers a business whose D&B legal name contains an img tag with an onerror handler; when a reviewer resolves that DUNS the payload runs in the reviewer session. connect-src self and img-src self/blob/data block external exfil, but the script can forge the KYC verdict POSTed back to /license/review, read/alter same-origin state, and act as the reviewer. A compromised backend echoing j.error/j.assurance is a more direct trigger.

Fix. Escape every server/D&B-derived string before innerHTML (add an escapeHtml like vedetta/sotto/izbushka and wrap entity.legalName, entity.country, j.error, j.decision, j.loa, j.assurance, j.methods, j.ref, and pill txt), or build nodes with textContent. Ideally also drop unsafe-inline via hashed inline scripts as sibling apps already do.

MEDIUM **Quadratic (O(k·n)) string rewriting defeats the zip-bomb guard with a CPU DoS on an in-budget XML part**

`/Users/tiagoacioli/dev/bloch-protocol/crates/postern-hygiene/src/lib.rs:135`

Scenario. scrub_ooxml admits any single decompressed entry up to MAX_ENTRY = 64 MiB (lib.rs:437,489). meta.xml/content.xml/styles.xml/app.xml are then scrubbed with remove_element (lib.rs:132) — which re-scans from index 0 on every iteration (s.find at lib.rs:136) and shifts the tail on every removal (s.replace_range at lib.rs:144). An attacker crafts a valid ODF whose meta.xml is ~64 MiB of tiny attributed metadata elements (e.g. millions of `<meta:user-defined meta:name="x"> </meta:user-defined>`, or dc:date/dc:creator). meta.xml alone runs 14 remove_element passes (ODF_META_ELEMENTS, lib.rs:548) each doing ~1.7M finds over a ~64 MiB buffer → ~10^14 byte-comparisons, pegging one core for many minutes to hours. blank_attr_values (lib.rs:153) and blank_element_text (lib.rs:217) share the same per-removal O(n) replace_range shift. The decompression cap only bounds MEMORY; it was explicitly added because this scrubs documents 'that can arrive from an untrusted peer' (lib.rs:434-435), and this CPU-time blowup bypasses that intent — a single small package (compresses to well under a MB) stalls the host.

Fix. Rewrite the XML scrub as a single forward pass instead of repeated find+replace_range: either reuse the existing quick-xml event pipeline (as scrub_odf_settings already does) to drop/blank elements in one streaming pass, or build the output with a cursor that advances (like strip_rsid_attrs at lib.rs:184, which is already O(n)) rather than mutating one String in place. Also add an explicit per-part CPU/size ceiling for the string-scrub arms so a pathological part fails closed (TooLarge) rather than spinning.

MEDIUM **approve_change trusts the caller-supplied approver role and never re-reads the approver's live enrollment (asymmetric with the proposer re-validation)**

`crates/postern-consiglio-auth/src/control.rs:170`

Scenario. The proposer side is deliberately re-validated against the store at approve time (proposer_role_now, lines 194-224) precisely because 'the identity captured at propose time is only as good as the proposer's CURRENT standing.' The identical argument applies to the approver, but approver.role is taken verbatim from the `approver: &User` argument (is_admin at line 170, sod_separated_from at 216) and is never looked up in self.enrollments. A compliance-officer who is later demoted to a business role via re-enroll (which does NOT revoke sessions — see the sibling finding) still holds a live Session whose snapshot role is 'compliance-officer'. If the server handler builds `approver` from that session's user (the only natural source — authorize() returns an AccessDecision, not a fresh User), the demoted ex-officer can still satisfy the second-person slot of dual-control, defeating the Brewer-Nash wall the module exists to enforce. Also reachable through store.rs PersistentAuthServer::approve_change (line 406), which forwards the same untrusted approver.

Fix. Re-read the approver from self.enrollments.get(&approver.org_id, &approver.user_id) at approve time (mirroring proposer_role_now) and use that CURRENT role for the is_admin and sod_separated_from checks; refuse fail-closed if the approver is no longer enrolled. Do not trust the role carried in the passed-in User.

MEDIUM **Key rotation / re-enrollment does not revoke the user's existing sessions, so a rotated (compromised) wallet key leaves the thief's sessions alive**

`crates/postern-consiglio-auth/src/control.rs:229`

Scenario. The dual-control Enroll commit calls self.enrollments_mut().enroll(e.clone()) (control.rs:229) and reports rotated=true when it replaces a prior enrollment, but never revokes that user's live sessions; the same is true of any direct enrollments_mut().enroll(). authorize() (server.rs:320-353) validates a session only by (a) session id live and (b) (org,user) still enrolled and (c) current role/clearance — it never checks that the session was minted under the enrollment's CURRENT public_key. So when alice's wallet key is stolen and remediated by rotating it (enroll a new pubkey for acme/alice), an attacker holding a session id minted under the compromised key keeps full access for up to the session TTL (default 3600s), because (org,user) is still enrolled and the role is unchanged. Rotation-in-place, the natural 'rotate my key' operation, is silently non-revoking; only the offboard path revokes sessions.

Fix. On any enroll that replaces an existing enrollment (previous.is_some()), call revoke_sessions_for(org,user) so a key rotation kills sessions minted under the old key. Alternatively stamp each enrollment with a key/epoch id, copy it into the Session at mint time, and have authorize() refuse a session whose epoch != the current enrollment's.

MEDIUM **Audit trail records declared capabilities, not actual task activity — 'glowing skull' overstates every run**

`/Users/tiagoacioli/dev/bloch-protocol/crates/yagabona/src/scheduler.rs:280`

Scenario. On a Fired run, poll() builds the trail Record from the GRANT, not from what the Task did: paths_touched is set to the full list of grant.write_paths (scheduler.rs:280-285) and network_used to grant.network.destination() (scheduler.rs:299), unconditionally. A NoopTask (or any task that writes nothing / makes no network call) still produces a record claiming it touched every writable path and used the network destination. The Record doc comments assert the opposite — trail.rs:57 'Concrete paths touched by the action' and trail.rs:59 'Network destination actually used'. Because the hash-chained trail is the crate's sole accountability mechanism, a systematically inaccurate record undermines the very property it exists to provide: a tamper-evident record that misrepresents behavior gives false assurance, and it cannot record a task that exceeded its declared network (there is no runtime network enforcement in-crate to compare against).

Fix. Have Task::run report the paths it actually opened and network it actually used (e.g. instrument ScopedFs to accumulate resolved paths and thread that back into the Record), and populate paths_touched/network_used from the observed activity. At minimum, rename/redocument the fields as 'granted_write_paths'/'granted_network' so the trail does not assert facts it did not observe.

MEDIUM **QEMU -drive/-qmp/-serial option injection via unescaped comma in caller-supplied disk_path**

`/Users/tiagoacioli/dev/bloch-protocol/desktop/src-tauri/src/terem.rs:396`

Scenario. terem_create(disk_path) accepts any path (create_impl, terem.rs:312-347) with no comma check; disk_path and its with_extension()-derived qmp_sock/serial_log are later interpolated raw into three comma-delimited QEMU option strings in start_impl: `file={},if=virtio,format=qcow2` (line 396), `unix:{},server=on,wait=off` (line 402), `file:{}` (line 404). QEMU parses these options on commas (a literal comma must be escaped by doubling, which this code never does). A caller passing e.g. disk_path="/tmp/g,file=/dev/sda,readonly=on" (qemu-img happily creates a file with a comma in its name, and guest.disk_path.is_file() then passes at line 385) yields `-drive file=/tmp/g,file=/dev/sda,readonly=on,if=virtio,format=qcow2`, letting the caller inject arbitrary -drive sub-parameters (a second file= pointing at a host block device, cache/aio/werror/snapshot flags, etc.). The trailing `,format=qcow2` limits full raw block-device passthrough, but this still hands the guest a host-device-backed drive and drives QEMU's large option-parser attack surface — directly eroding the module's own headline guarantee (terem.rs:56-59) that 'no guest ever gets a raw host block-device handle', achieved here only by the QMP allowlist while the argv path is left unescaped.

Fix. Escape commas per QEMU's rule (replace ',' with ',,') when building the -drive/-qmp/-serial option strings, or reject disk_path values containing ',' (and other QEMU option metacharacters) in create_impl before storing them. Prefer the -blockdev/-object modern syntax or pass the path via a QMP-free absolute-path allowlist under app_data_dir so the guest disk can never be aimed at a host device node.

MEDIUM **set -x traces the tokenized GitLab clone URL into the S3-uploaded build log (credential leak beyond the documented metadata exposure)**

`/Users/tiagoacioli/dev/bloch-protocol/scripts/ec2-userdata-mobile.sh:40`

Scenario. Line 40 enables `set -x`; line 41 redirects all output through `tee -a /var/log/postern-mobile.log`. When line 61 runs `git clone ... "$GITLAB_REPO_URL" repo`, bash's xtrace prints the fully-expanded command — `+ git clone ... https://oauth2:<TOKEN>@gitlab.com/...` — into that log. Line 87 then does `aws s3 cp /var/log/postern-mobile.log s3://$S3_BUCKET/$S3_PREFIX/build.log`, and the same trace is streamed to the EC2 console (readable via get-console-output). The script's header only warns that user-data is readable at 169.254.169.254; it does NOT warn that the token is additionally copied verbatim into a persisted S3 object (which may have broader/misconfigured read access) and the console. Anyone who can read the bucket or console recovers the deploy token — the 'revoke after build' mitigation assumes an exposure surface that is actually larger than documented.

Fix. Do not use `set -x` while a secret-bearing URL is in the environment (or `set +x` around the clone), and/or pass the token via a git credential helper / `GIT_ASKPASS` instead of embedding it in the remote URL. Alternatively scrub the token from the log before upload (e.g. `sed 's#oauth2:[^@]*@#oauth2:REDACTED@#g'`).

MEDIUM **Remote SHA256SUMS is merged and re-signed WITHOUT verifying its existing signature — laundered hashes for other artifacts get a valid production signature**

`/Users/tiagoacioli/dev/bloch-protocol/os/publish-iso.sh:313`

Scenario. On a real publish, line 313 does `r2 get --key $PREFIX/SHA256SUMS` to fetch the current remote manifest, then lines 326-331 drop only the line for the artifact being published now and re-append/sort, and lines 346-359 sign the WHOLE merged file with the production minisign secret key (self-checking only that the signature verifies against its own pubkey). The pre-existing lines for OTHER images are never signature-verified before merging and are never re-hashed against on-disk bytes this run. An attacker with write access to R2 (the destination the signature is meant to protect) — or who can tamper the fetched object — plants a malicious hash for e.g. postern-os-desktop-x86_64.iso into the remote SHA256SUMS. The next time the operator publishes ANY image, this script keeps that malicious line and signs it with the real key, producing a validly-signed SHA256SUMS endorsing an attacker-chosen hash. End users who verify the signature then trust the forged desktop hash. The signature is the only defense against a compromised R2, and this defeats it for previously-published artifacts.

Fix. Before merging, fetch SHA256SUMS.minisig and run `minisign -Vm SHA256SUMS -P $PUBKEY` on the remote manifest; refuse to proceed if it does not verify. Or only sign lines the operator hashed/verified this run (do not carry forward unverified remote entries).

MEDIUM **Permissive CORS contradicts the threat model and defeats write-auth on the default localhost node via drive-by requests**

`src/rpc/mod.rs:148`

Scenario. THREAT_MODEL.md:352 states the posture is "No CORS headers" and frames browser drive-by as a --rpc-public-only concern, but the router actually installs `CorsLayer::permissive()` (Allow-Origin reflected, all methods/headers). A node operator running the DEFAULT config (bind 127.0.0.1, api_key set, require_auth_for_writes=true) visits any malicious web page. The page issues `fetch('http://127.0.0.1:16210/', {method:'POST', headers:{'content-type':'application/json'}, body:...})`. The preflight passes (permissive), the POST is delivered, the node sees source IP 127.0.0.1 -> `is_trusted_local_ip` returns true (auth.rs:157) -> `authorize_with_trust` returns Allow BEFORE the api-key check, so `require_auth_for_writes`+`api_key` are bypassed. The attacker can drive `sendrawtransaction`/`submitblock` and, because permissive CORS lets the page READ the response, exfiltrate `getbalance`/`getutxos`/`listtransactions`/`getpeers` wallet-and-topology data for the operator's own node — none of which requires knowing the API key. The threat model's stated mitigation ("don't run RPC public") does not apply because this hits the localhost-only default bind.

Fix. Do not use `CorsLayer::permissive()` for a JSON-RPC endpoint. Default to no CORS (omit the layer, letting the same-origin policy block cross-origin reads as the threat model assumes), or restrict `allow_origin` to an explicit operator-configured allowlist and never reflect arbitrary origins. Separately, reconcile THREAT_MODEL.md:352 ("No CORS headers") with the code. If a browser explorer genuinely needs CORS, gate it behind an opt-in config flag and document that it removes the localhost trust boundary.

LOW **xref offsets assume <10 GB output; wider offsets corrupt the fixed-width cross-reference table**

`/Users/tiagoacioli/dev/bloch-protocol/crates/postern-quarantine/src/pdf.rs:214`

Scenario. An injected renderer returns many large rasters (permitted: up to MAX_PAGES=10000 pages, each up to MAX_DIMENSION^2*3 ≈ 805 MB, with no aggregate cap). Once the running byte offset of any object reaches 10^10 (~9.3 GB), `format!("{offset:010}")` emits 11+ digits instead of the PDF-mandated fixed 10-digit field. The classic xref table entries stop being 20 bytes wide, so `startxref`/xref parsing is misaligned and the produced PDF is malformed (strict viewers fail to open it; lenient ones fall back to full-file reconstruction). Not a memory-safety issue, a correctness one in a module documented as shipping and tested.

Fix. Either enforce an aggregate output-size cap that keeps every offset < 10^10, or stop relying on a fixed 10-digit classic xref: switch to a PDF 1.5 cross-reference stream, or assert/return QError::TooLarge when out.len() would exceed the 10-digit offset range before writing the xref.

LOW **rebuild_pdf_from_pages has no aggregate size bound — page-count cap and per-page cap multiply**

`/Users/tiagoacioli/dev/bloch-protocol/crates/postern-quarantine/src/pdf.rs:133`

Scenario. MAX_PAGES (10000) and per-axis MAX_DIMENSION (16384) are each enforced, but their product is not: a compromised or buggy injected renderer that returns thousands of near-max rasters forces rebuild_pdf_from_pages to allocate `out` holding the sum of all rasters verbatim (10000 × ~805 MB ≈ multi-TB), OOM-killing the host process. This is an in-process DoS on the rebuild step itself, distinct from the disclosed 'render is unsandboxed' caveat (the rebuild step is the part claimed to ship safely).

Fix. Add a total-output/aggregate-pixel cap (e.g. sum of width*height*3 across pages, or a MAX_TOTAL_RGB_BYTES constant) checked in the validation loop at pdf.rs:144-146, returning QError::TooLarge before emitting; this also subsumes the xref-offset overflow above if set below ~9 GB.

LOW **Whitespace-literal matching lets rsid/attributed metadata survive when a producer separates attributes with tab/newline instead of a space**

`/Users/tiagoacioli/dev/bloch-protocol/crates/postern-hygiene/src/lib.rs:185`

Scenario. strip_rsid_attrs matches the literal NEEDLE = " w:rsid" (a single ASCII space; lib.rs:185), and remove_element only recognizes `<tag>` or `<tag ` (space) openers (lib.rs:133). XML permits any whitespace (\t \n \r) between an element name and its attributes and between attributes. A document serialized with newline/tab attribute separators — e.g. `<w:r\n w:rsidR="00AB12CD">` or `<meta:user-defined\n meta:name="secret">…` — is not matched: the inline w:rsid* fingerprints (editing machine/session) and attributed metadata elements are copied through UNSCRUBBED, yet scrub reports success and the CLI 'check' can report the file clean. This contradicts the crate's explicit 'every w:rsid* attribute stripped' / fail-closed claims. Real-world Office/LibreOffice emit single spaces, so live exposure is limited to non-standard or pretty-printed serializations.

Fix. Match on any XML whitespace, not a literal space: in strip_rsid_attrs scan for the token `w:rsid` and verify the preceding byte is is_ascii_whitespace() (the same boundary test blank_attr_values already uses at lib.rs:162); in remove_element treat the opener as `<tag` followed by whitespace or `>` / `/`. Better, route these through the quick-xml pass so element/attribute boundaries are parsed rather than pattern-matched.

LOW **Invalid-UTF-8 decode path leaves decrypted secret bytes un-zeroized (violates the module's stated 'partial secrets wiped on parse failure' invariant)**

`/Users/tiagoacioli/dev/bloch-protocol/crates/postern-vault/src/entry.rs:129`

Scenario. Entry::decode parses a Password username (line 129) or a Note body (line 136) via String::from_utf8(r.len_prefixed()?.to_vec()). The inner .to_vec() produces a PLAIN Vec<u8> (not Zeroizing) holding the just-decrypted secret field. If the bytes are not valid UTF-8, from_utf8 returns Err(FromUtf8Error) which OWNS that Vec; .map_err(|_| VaultError::Corrupt) discards the error and its Vec is freed WITHOUT zeroization. This is reachable when get_entry() is called on an untyped value whose bytes happen to start [1, 1(or 2), <len>, <non-UTF8 payload>] (typed and untyped entries are documented to coexist in one vault; see the test v.put("raw", ...) + get_entry), or on any corrupt/tampered-but-decrypted record. The module docstring (line 22) and the crate promise 'every secret field is parsed straight into a Zeroizing container, so partial secrets are wiped from RAM on parse-failure paths' — this path breaks that guarantee, leaving a fragment of a decrypted secret in freed heap.

Fix. Decode into a Zeroizing buffer first and validate UTF-8 without letting the error own the bytes, e.g.: let raw = Zeroizing::new(r.len_prefixed()?.to_vec()); let s = std::str::from_utf8(&raw).map_err(|_| VaultError::Corrupt)?; Zeroizing::new(s.to_string()) — so `raw` (Zeroizing) always wipes on the failure path. Apply to both the username (line 129) and note-body (line 136) decodes.

LOW **KeyStore::retrieve returns a bare Copy [u8;32], breaking the Zeroizing chain and leaving un-wiped key copies on the stack**

`/Users/tiagoacioli/dev/bloch-protocol/crates/postern-vault/src/keychain.rs:58`

Scenario. The trait returns Result<Option<[u8;32]>, _> (line 58); MacKeyStore and SecretServiceKeyStore implement it the same way. [u8;32] is Copy, so the raw vault key is returned by value and every subsequent move copies (never wipes) it. In unwrap_or_derive (line 131) `if let Some(key) = store.retrieve(account)?` binds a plain [u8;32] on the stack, then Zeroizing::new(key) at line 132 COPIES those 32 bytes into the wrapper — the original stack `key` (a Copy value) is left un-zeroized when the function returns, and any code calling retrieve() directly gets a key array that self-wipes on nothing. The rest of the crate deliberately keeps key material in Zeroizing<[u8;32]> (derive_key's return type), and this trait boundary silently drops that protection, so a retrieved keychain key lingers in freed stack frames. Exploitability requires local RAM disclosure (core dump, swap, cold-boot), so severity is low, but it is a real deviation from the crate's own RAM-hygiene posture, independent of the disclosed 'OS keychain is not a TEE' caveat.

Fix. Change the trait to return Zeroizing<[u8;32]> (or a Zeroize-on-drop newtype) from retrieve, and have each backend build it in place; unwrap_or_derive then returns it without an intermediate Copy. If the trait signature must stay, at minimum zeroize the local in unwrap_or_derive before returning (hard with Copy — the type change is the correct fix).

LOW **sign_sealed panics (no error channel) on a SenderIdentity whose secret is not exactly 32 bytes**

`crates/postern-courier/src/sender_auth.rs:307`

Scenario. SenderIdentity has pub fields and the whole design assumes identities are persisted and reloaded out-of-band. A caller that reconstructs `SenderIdentity { public, secret }` from a truncated/corrupted key file (secret.len() != 32) and calls sign_sealed hits `Seed::try_from(sender.secret.as_slice()).expect("...")`, which unwinds/aborts the process. sign_sealed returns SignedSealedPayload, not Result, so there is no graceful error path — a malformed local key file turns into a crash (availability), not a handled error. Not a crypto break and not remotely attacker-controlled (it is the sender's own signing seed), hence low.

Fix. Validate the secret length up front and surface it as an error rather than panicking: either make sign_sealed return Result<SignedSealedPayload, CourierError> and map a bad-length seed to a CourierError::SenderKey, or gate SenderIdentity construction behind a checked constructor (private field + `try_from`/`from_secret` returning Result) so a wrong-length seed can never reach the `.expect`.

LOW **Transfer-Encoding: chunked guard is format-sensitive; some valid chunked responses bypass it (safe-fail)**

`/Users/tiagoacioli/dev/bloch-protocol/crates/postern-tor/src/lib.rs:278`

Scenario. An onion/exit peer returns a chunked response whose header does not match the exact literal substring 'transfer-encoding: chunked' after lowercasing — e.g. 'Transfer-Encoding:chunked' (no space), 'Transfer-Encoding: gzip, chunked' (chunked not immediately after colon-space), or a folded header. parse_rpc_response's `head.to_ascii_lowercase().contains("transfer-encoding: chunked")` check misses it, so the stated guarantee ('refuses chunked') does not fire; the naive body split at line 281 then hands chunk framing to serde_json::from_str.

Fix. Parse Transfer-Encoding properly: split head into header lines, match the field name case-insensitively, trim the value, and reject if any transfer-coding token equals 'chunked' (not a fixed substring). The current miss is safe-fail (mis-detected chunked body yields a JSON parse error, not a corrupted-but-valid result), hence low severity.

LOW **verify_response does not bind the response's claimed identity to the challenge it answers, allowing an enrolled insider to burn another user's challenge nonce**

`crates/postern-consiglio-auth/src/server.rs:215`

Scenario. verify_response looks up the enrollment by response.org_id/response.user_id (server.rs:215-217) and checks the nonce against the challenge globally (issued.contains_key(challenge.nonce), 243) without ever asserting response.org_id==challenge.org_id && response.user_id==challenge.user_id, and without checking the IssuedNonce owner recorded at issue time. An enrolled user Eve who observes a live challenge issued to acme/alice can submit a Response with her own org/user and pubkey, echoing alice's nonce and a valid signature over alice's challenge transcript (which Eve can produce with her own key). This passes all checks and consumes (retires) alice's nonce, so alice's subsequent legitimate response is rejected as Replay/UnknownNonce — a targeted login-denial. No impersonation is possible (forging another user's session still requires their secret key), so impact is limited to DoS requiring knowledge of the victim's nonce.

Fix. Early in verify_response, reject unless response.org_id==challenge.org_id and response.user_id==challenge.user_id (and/or verify the IssuedNonce owner recorded for the nonce matches the responding identity), so a nonce can only be consumed by the identity it was issued to.

LOW **Empty-string path entry silently expands to the whole workspace root**

`/Users/tiagoacioli/dev/bloch-protocol/crates/yagabona/src/threshold.rs:189`

Scenario. A manifest with read_paths or write_paths containing an empty string (e.g. write_paths = [""] or a stray trailing empty element) passes confine_one: "" has no ParentDir component and is not absolute, so ws.join("") -> "/ws/" which canonicalizes to the workspace root and passes starts_with(ws). The resulting grant covers the ENTIRE workspace for that access mode (within_any then matches every in-workspace path), rather than the narrow subdir the author presumably intended. Verified: Path::new("").is_absolute()==false, no ParentDir, Path::new("/ws").join("").starts_with("/ws")==true.

Fix. In confine_one, reject an empty (or all-whitespace) declared path with a Refusal before joining, e.g. `if declared.trim().is_empty() { return Err(Refusal::of(RefusalReason::PathUnresolvable{ declared: declared.into(), io: "empty path".into() })); }`.

LOW **Inline-secret tripwire skips the ritual.granted_at field**

`/Users/tiagoacioli/dev/bloch-protocol/crates/yagabona/src/threshold.rs:248`

Scenario. scan_for_secrets enumerates 8 fields but omits ritual.granted_at (task.name, task.description, task.schedule, ritual.granted_by, ritual.formula, read_paths, write_paths, task.expires are covered; granted_at is not). A credential smuggled into granted_at (a free-form String, not validated as a timestamp anywhere in the crate) evades the heuristic that fires for every other string field. Low impact because the scan is explicitly a best-effort tripwire, but the omission is an inconsistency an author would not expect.

Fix. Add ("ritual.granted_at", m.ritual.granted_at.as_str()) to the fields array so all free-form string fields are scanned uniformly.

LOW **Workspace-escaping read is reported as WriteDenied (mislabeled error)**

`/Users/tiagoacioli/dev/bloch-protocol/crates/yagabona/src/fsboundary.rs:159`

Scenario. resolve_in_workspace is shared by both read and write paths; when a resolved path escapes the workspace root it always returns FsError::WriteDenied (fsboundary.rs:159), even when called from resolve_read (open_read/read). The access is correctly denied, but a denied out-of-workspace READ surfaces to callers and logs as a write denial, which can mislead operators triaging the audit/error output. Cosmetic, not an access-control gap.

Fix. Return a mode-neutral error (e.g. a new FsError::EscapesWorkspace variant) from resolve_in_workspace, or pass the access mode in so the correct ReadDenied/WriteDenied is produced.

LOW **lower_wall permanently and globally disables a Chinese wall for all subjects, with no re-raise and no enforced reason**

`/Users/tiagoacioli/dev/bloch-protocol/crates/ezekiel1/src/eval.rs:214`

Scenario. Owner calls lower_wall("banking-coi", owner, reason) intending a scoped audit engagement (the doc/test frame it as 'dual review approved' for one review). Implementation inserts the wall name into lowered_walls (line 220); from then on decide() at line 287 filters that wall out for EVERY subject on EVERY subsequent access for the entire engine lifetime. There is no raise_wall / clear method, so the Brewer-Nash separation-of-duties barrier stays down permanently and applies to subjects wholly unrelated to the engagement — a broader and longer-lived widening than the 'engagement' framing implies. Additionally the doc (line 212) states it 'requires a non-empty reason', but no reason.is_empty() check exists, so a wall can be lowered with an empty audit reason.

Fix. Add a raise_wall(wall, owner, reason) that removes the name from lowered_walls and records it; consider scoping a lowering to a specific subject/engagement and/or an expiry rather than a process-lifetime global set. Enforce the documented non-empty-reason precondition (return an error on empty reason) or fix the doc to match.

LOW **Subject name uniqueness is never validated; first-match wins can silently pick the wrong subject entity**

`/Users/tiagoacioli/dev/bloch-protocol/crates/ezekiel1/src/eval.rs:248`

Scenario. Resources are checked for duplicate ids (firmament::validate rejects dup/breach), but Policy::validate never checks subject-name uniqueness. A policy with two [[subjects]] named "alice" (e.g. different tenant/layer) passes validation. decide() at line 248 and validate_grants at policy.rs:351 both resolve a subject with iter().find(|s| s.name==subject), which returns only the FIRST entry, silently shadowing the second. The subject's layer, tenant, and thus firmament/lattice outcome depend on declaration order rather than being well-defined, so a grant written against the intended 'alice' can be evaluated against a different 'alice''s clearance/tenant.

Fix. In Policy::validate, reject duplicate subject names (mirror the DuplicateResource check in firmament::validate), so an ambiguous subject identity fails closed at the gate instead of resolving by array order.

LOW **Business policies can leave subjects/resources tenant-less, collapsing them into one shared implicit firmament**

`/Users/tiagoacioli/dev/bloch-protocol/crates/ezekiel1/src/firmament.rs:109`

Scenario. same_firmament treats None==None as the same firmament (intended for the personal single-household case). In Business edition nothing requires subjects or resources to declare a tenant (Subject.tenant/Resource.tenant are #[serde(default)] Option, and edition::enforce only forbids tenants in Personal). A business author who omits `tenant` on some entities makes every tenant-less subject and tenant-less resource mutually reachable through step (3) at eval.rs:268, silently defeating firmament isolation for those entities with no validation error.

Fix. In Business edition, require every subject and resource to name a declared tenant (reject tenant-less entities), or treat a missing tenant as its own non-matching firmament so an omission fails closed rather than merging everything into a shared None firmament.

LOW **Stateless evaluate_json/wasm surface cannot enforce stateful Brewer-Nash across accesses**

`/Users/tiagoacioli/dev/bloch-protocol/crates/ezekiel1/src/api.rs:62`

Scenario. evaluate_json builds a fresh Engine::new_ephemeral per call (line 85), so history is always empty and the wall check at eval.rs:290-291 sees no prior touches — may_access always returns Allow. This is correct for a single first-access, but the doc (lines 52-53) says it 'Runs the entire Engine pipeline ... Brewer-Nash walls', and a deployer who wires this JSON/wasm surface as their enforcement point (instead of a persistent Engine via Engine::new) gets NO Chinese-wall enforcement across successive accesses: bank-b is allowed after bank-a because nothing carries history between calls. Only the separate wall_barred_json entry point (which requires the caller to supply history) actually enforces the wall.

Fix. Document explicitly that evaluate_json is single-shot and does not enforce stateful Brewer-Nash (history must be maintained by the caller and checked via wall_barred_json), or provide a stateful evaluation surface that threads history across calls, so the wasm/console path is not mistaken for a wall-enforcing gate.

LOW **Unchecked `tip + 1` overflows if a BlockSource reports tip_height == u64::MAX**

`crates/bloch-tokens/src/indexer.rs:96`

Scenario. sync() takes an arbitrary `BlockSource`. A misbehaving/compromised source whose `tip_height()` returns `u64::MAX` reaches `let remote_len: u64 = tip.map_or(0, |t| t + 1);`. `u64::MAX + 1` overflows: under the crate's default (dev) profile — which has overflow-checks on, confirmed empirically — this panics, aborting the indexer (DoS); under a release build it silently wraps to `remote_len = 0`, so the indexer treats the chain as empty and rewinds/discards ALL indexed token state. This is the only unchecked add in an otherwise meticulously checked-arithmetic codebase. Reachability is narrow: the documented mapping is tip = getblockcount-1 (max representable tip is u64::MAX-1, which is safe), so only a BlockSource impl returning u64::MAX directly triggers it — hence low, not medium.

Fix. Use `tip.map_or(0, |t| t.saturating_add(1))` to match the crate's existing defensive-arithmetic style, or explicitly reject/clamp an out-of-range tip before computing remote_len.

LOW **Replay-guard watermark can regress on out-of-order commit (commit assigns instead of max; approve() never re-checks)**

`integration/signer-protocol/src/replay.rs:93`

Scenario. ReplayGuard::commit does `self.last_counter = req.counter` unconditionally rather than `max(last, counter)`, and PhoneApprover::approve (protocol.rs:235) signs + commits WITHOUT re-running guard.check (the only check is in receive(), before the human decision). A caller that holds two Pendings at once — receive req A (counter 5), receive req B (counter 6), both pass check because nothing is committed yet — then approves B (commits last_counter=6) and afterwards approves A: approve() re-signs A and commit() rolls last_counter back from 6 to 5, re-opening counter 6 as acceptable again. It also means a stale Pending gets signed even though a higher counter was already approved. Requires the requester (a compromised/buggy desktop, the disclosed A5) plus a UI that approves queued requests out of counter order; the on-wire channel (channel.rs monotonic counter + AEAD) still blocks a pure network A2 replay, so real impact is limited to the already-compromised-endpoint model — but the guard is meant to be belt-and-suspenders and here it silently weakens.

Fix. Make the watermark monotonic under any call order: `self.last_counter = self.last_counter.max(req.counter);` in commit, and have PhoneApprover::approve re-assert `self.guard.check(&pending.req)?` immediately before signing so a Pending that became stale (or whose counter no longer advances) is refused rather than signed. Optionally retire the nonce/counter atomically with a single check-and-commit method to remove the check→approve TOCTOU on concurrently held Pendings.

LOW **rpcPort interpolated into innerHTML without escaping (stored XSS via poisoned settings)**

`/Users/tiagoacioli/dev/bloch-protocol/desktop/src/modules/node.js:57`

Scenario. node.js builds the Node pane with root.innerHTML and, unlike the two sibling fields which are escaped (line 55 `value="${UI.esc(s.binary)}"`, line 56 `UI.esc(s.dataDir)`), interpolates rpcPort raw: `value="${s.rpcPort}"`. s.rpcPort comes from loadSettings() (ui.js:26-30), which does Object.assign(def, JSON.parse(localStorage['postern.settings'])) with NO type validation. Normal UI writes coerce it with parseInt (node.js:123), but an attacker who can write the app's WebView localStorage (local filesystem access to the app's storage, or any prior in-webview code exec) can plant {"rpcPort":"\"><img src=x onerror=...>"}. Next time the user opens the Node pane the payload executes in the app origin — which can read the unlocked vault, keys, and mnemonic. Persistent, and it fires on a routine navigation.

Fix. Escape it like its neighbours: value="${UI.esc(s.rpcPort)}"; better, coerce rpcPort to an integer in loadSettings (Number()/parseInt clamped to 1-65535) so a non-numeric value can never survive into any sink.

LOW **rpcPort concatenated into RPC URL authority without validation (potential off-box JSON-RPC / zero-network break)**

`/Users/tiagoacioli/dev/bloch-protocol/desktop/src/ui.js:34`

Scenario. P.rpcUrl = () => `http://127.0.0.1:${P.state.settings.rpcPort}`. rpcPort is merged unvalidated from localStorage (ui.js:26-30). A poisoned value such as "@attacker.tld" yields http://127.0.0.1:@attacker.tld, whose parsed host is attacker.tld (the literal 127.0.0.1 becomes userinfo). This URL is passed to the real backend `rpc` command by explorer.js:11, wallet.js:167/185 and the main.js:151 poll loop, so getbalance/getnetworkinfo calls (which carry wallet addresses) could be directed to an external origin — contradicting the zero-network posture. The only numeric guard is parseInt in node.js:123, applied only when the field is edited in the UI, not in loadSettings or rpcUrl.

Fix. Validate rpcPort centrally in loadSettings (integer 1-65535, else fall back to default) and/or build the URL as `http://127.0.0.1:${Number(P.state.settings.rpcPort)||8645}`; the backend rpc command (src-tauri, out of scope) should also reject non-loopback hosts as defense in depth.

LOW **LUKS /persist first-boot discards the only known key when TPM2 enroll fails, bricking every subsequent boot; comment falsely claims a recoverable bootstrap keyslot**

`/Users/tiagoacioli/dev/bloch-protocol/os/cloud.nix:291`

Scenario. In postern-persist's script, first boot formats /persist with a random `$key` (line 286-287), opens it, then `PASSWORD="$key" systemd-cryptenroll --tpm2 ... || echo 'keeping the bootstrap keyslot'` (line 291-292) and immediately `unset key` (line 293). If the TPM2 enroll fails (the `||` branch — plausible on providers without a usable TPM2, the exact 'validate per provider' case), no TPM keyslot is added and the random passphrase — the only credential for the surviving keyslot 0 — is discarded from memory. The volume mounts this boot (mapper still open) but on the NEXT boot the elif branch attempts a TPM2 unlock that cannot succeed, and no known passphrase exists, so /persist (WireGuard key, node state) is permanently unrecoverable. The inline comment 'keeping the bootstrap keyslot' is misleading: the keyslot exists but its key is gone.

Fix. Treat TPM2 enroll failure as fatal (fail the unit) rather than continuing, OR do not `unset key` / add a stable operator-recovery keyslot before discarding the bootstrap key, so a TPM-less provider degrades to a recoverable state instead of a bricked volume.

LOW **Predictable /tmp log + evidence paths written via tee/`>` are symlink-clobberable (CWE-377/CWE-59)**

`/Users/tiagoacioli/dev/bloch-protocol/os/boot-harness/boot-qemu-aarch64.sh:47`

Scenario. LOG defaults to the fixed, world-predictable `/tmp/qemu-aarch64-boot.log` (line 47) and EVIDENCE is derived as `${LOG%.log}.evidence` (line 93). The evidence sidecar is written with `> "$EVIDENCE"` (line 94-109) and the boot log with `boot ... | tee "$LOG"` (line 166) — both follow symlinks. On a shared build/CI host, a local unprivileged attacker who pre-creates `/tmp/qemu-aarch64-boot.log` or `/tmp/qemu-aarch64-boot.evidence` as a symlink to a file the harness user can write causes tee/redirect to truncate and overwrite that target. If the harness runs as root in CI, this is arbitrary-file overwrite.

Fix. Create the log/evidence under a per-run `mktemp -d` (as sibling scripts already do), or open with `O_NOFOLLOW` `/`>|` into a freshly-created private dir; do not use a fixed name directly in /tmp.

LOW **Unquoted command substitution word-splits/globs bundle filenames when building SHA256SUMS**

`/Users/tiagoacioli/dev/bloch-protocol/packaging/build-signed-bundle.sh:95`

Scenario. Line 95: `( cd "${BUNDLE_DIR}" && sha256_cmd $(for f in "${ARTIFACTS[@]}"; do basename "$f"; done) )`. The `$(...)` is unquoted, so any bundle artifact whose name contains whitespace is split into multiple arguments (hashing wrong/nonexistent paths) and any name containing a glob metacharacter (`*`, `?`, `[`) is expanded against the cwd. The rest of the script handles names safely via the ARTIFACTS array, so only this line diverges — a bundle dir with such a filename yields an incorrect SHA256SUMS or a hard error, and the manifest/SHA256SUMS cross-check downstream would misbehave. (Lower impact because the bundle dir is normally a controlled tauri output.)

Fix. Feed filenames to the hasher without an unquoted expansion, e.g. iterate `for f in "${ARTIFACTS[@]}"; do (cd "$BUNDLE_DIR" && sha256_cmd "$(basename "$f")"); done`, or use `find -print0 | xargs -0`.

LOW **esc() omits single-quote escaping, inconsistent with escapeHtml elsewhere (latent XSS)**

`apps/postern-enterprise/console.html:471`

Scenario. console.html runs under script-src unsafe-inline and renders imported master/sub-bundle fields (app.name, app.tagline, s.label, s.expires, s.token) into innerHTML via esc(). esc() at line 471 replaces only ampersand, less-than, greater-than and double-quote, not the single quote, unlike escapeHtml in the other apps. All current sinks are text or double-quoted-attribute context so it is not exploitable today, but a future single-quoted attribute built from esc-ed bundle data would allow break-out and, under unsafe-inline, an onerror/onclick from a malicious imported bundle.

Fix. Add a single-quote mapping to esc()'s replacement map and regex class so esc matches the stricter escapeHtml used across the other apps.