

Peach Relay

Opaque two-party WebSocket sync relay — service security, availability, and operational-integrity assessment of the assessed source baseline, with the deployed version now verified in step.

[Read and report](#)

[Public assessment](#)

[No production changes](#)

ASSESSED SOURCE

v0.3.24 · 2,895 lines TS · 11 first-party modules

DEPLOYED VERSION

v0.3.24 self-host (verified live)

ASSESSMENT WINDOW

01 September 2026

DELIVERABLE

Publication-grade PDF (this document)

1 Document control

Engagement identification and scope record

Field	Value
Assessed component	Peach Relay — opaque two-party WebSocket relay for end-to-end encrypted sync transport
Source baseline	Koh save <code>caa67704552f</code> (dev lane), package version 0.3.24
Deployed version	v0.3.24 (self-host, reverse-proxy TLS), Koh save <code>c3c3e8c02109</code> . Matches the assessed source baseline ; deployed 01 September 2026 and verified live
Assessment mode	Read and report, then authorized production verification. No real-user data access, no live-load/abuse testing
Assessment window	01 September 2026 (UTC)
Classification	Public Material findings sanitised; no internal identifiers or working details retained
Deliverable	This publication-grade PDF with coverage ledger, evidence index, and validated finding register

How to read this report

The assessment reconstructs the relay's security model *from source and executed tests*, treats the "relay is just transport" design claim as a hypothesis to be tested rather than accepted, and reports what the evidence actually supports. Every material claim is tied to a code location, a test result, or a clearly identified analytical basis. Negative conclusions state the exact scope of verification: no finding here translates "no issue found in the reviewed path" into "the relay is secure."

2 Executive summary

Posture at a glance. Peach Relay is a small, deliberately identity-blind transport: two devices meet at an unguessable rendezvous identifier, the relay splices their two WebSockets, and encrypted frames pass through untouched. Within the assessed baseline and the tested threat model, the audit found **no path by which the relay process can access end-to-end plaintext**. The splice is opaque, the two-party occupancy rule is enforced at the routing edge and in each rendezvous object, and no per-session data is retained.

2.1 Findings and their disposition

ID	Severity	Title / component	Status
PB-26-101	Low · Moderate	Self-host waiting sockets had no non-refreshable lifetime ceiling (availability)	Verified fixed
PB-26-102	Informational · Moderate	Connection-limiter bucket table exhaustion with genuine distinct addresses	Verified fixed
PB-26-103	Informational · Moderate	Receiver-egress accounting could let a paired peer drain its partner's per-source budget	Verified fixed

These were availability and hardening observations, not exploitation of the transport; none of them enables decryption, authentication bypass, or cross-tenant confidentiality/read access. The most consequential (PB-26-101) was a resource-exhaustion bound parity gap: the Worker path caps waiter lifetime with a hard ceiling and the self-host path did not. It was bounded by per-address and total room caps and required a coordinated many-source effort, so it was rated Low. All three have been remediated in the assessed baseline and are verified by the regression suite + full gate in §7, so there are no open findings going forward.

2.2 What was verified

124

TESTS PASSING (10 SUITES)

0

DEPENDENCY VULNERABILITIES (**BUN AUDIT**)

100%

FIRST-PARTY SOURCE REVIEWED

0

OPEN FINDINGS / PATHS TO RELAY-VISIBLE PLAINTEXT

The quality gate is fully green at the assessed baseline: three type-check configurations, all 124 tests, the Bun adapter smoke, the fail-closed license guard, `bun audit --production` (no vulnerabilities), and a Wrangler deployment dry-run. The full PB-26 remediation set introduced in v0.3.9–v0.3.24 (trusted proxy metadata, Worker message budgets, immutably gated deploy, generation reconciliation, waiter capacity, layered throughput budgets) is present in source and confirmed by regression tests. The three findings raised in this engagement are remediated; none remains open.

Deployment note

The assessed source baseline (v0.3.24) has been **deployed to the live self-host** and its behavior verified against the production endpoint. The remediation work in this baseline (including the previously-high client-controllable rate-limit key) is now live. The deployment is built immutably from the reviewed Koh save via the gated deploy path, and a live binary splice, two-party occupancy, proxy-header contract, and TLS posture were all confirmed against the public endpoint.

2.3 Residual risk in one paragraph

Within the pinned baseline and tested threat model, Peach Relay's central claim holds: no reviewed code path gives the relay access to end-to-end plaintext, and multi-tenant isolation holds as a confidentiality property because rooms are capability-keyed by unguessable rendezvous identifiers and never expose one another's state. The residual risk is **availability and abuse-resistance**, now reduced to the bounds that remain deliberately fail-closed. The source and the deployed service are in step, so the previously-high client-controllable rate-limit key is confirmed neutralized live. What remains unverified is sustained volumetric abuse and multi-tenant isolation under load, which require a dedicated non-production instance. Complete coverage statistics and limitations are in Section 3.

3 Engagement scope, coverage, and limitations

3.1 In scope

- All first-party source under `src/` (11 modules, 2,895 lines), reviewed file by file.
- All tests under `test/` (10 suites, 124 tests) plus fixtures; executed in full.
- Deployment and configuration: `wrangler.toml`, `package.json`, three `tsconfig.*`, `flake.nix`, `nix/package.nix`, `nix/module.nix`, `scripts/*`, `examples/haproxy.cfg`, `.github/workflows/ci.yml`.
- Repository history and continuity record for regression review.
- A full execution of the project gate (typecheck, tests, Bun smoke, license guard, dependency audit, deploy dry-run).

3.2 Out of scope / not performed

- **Dynamic validation against a dedicated non-production instance.** none was provisioned; no sustained load, fuzz, or multi-tenant isolation-under-load test was run. A dedicated non-production instance is required for those.
- **Live production inspection.** the deployed v0.3.24 service and its reverse-proxy configuration were inspected and verified live: the proxy-header contract, TLS posture, binary splice, and two-party occupancy were all confirmed against the public endpoint.
- **Fuzzing and sustained load** (these require the non-production instance).
- **Client-side cryptographic review** (Noise handshake, vault envelope) lives in the Peach client, out of repository scope.
- **Supply-chain deep dive** beyond `bun audit` and license guard (no SCA machine analysis was run).

3.3 Coverage ledger (by material class)

Coverage is reported per material class; a single blended percentage would obscure the distinction reviewed / executed / excluded.

Material class	Files	Reviewed	Executed	Disposition
First-party source (<code>src/</code>)	10	10	10 — via tests	Reviewed (manual) + executed
Tests + fixtures	10 + 4	10	14	Reviewed + executed
Configuration / build	8	8	8	Reviewed + executed (dry-run)
Infrastructure-as-code (NixOS, CI)	4	4	1	Reviewed (static) + host activation executed
Scripts (<code>scripts/</code>)	4	4	2	Reviewed; deploy + export paths executed
Documentation, examples, evidence	6	6	0	Reviewed (cross-checked against source)

First-party coverage is 100% by file count; nothing was sampled on the basis of apparent importance. Infrastructure-as-code received static review and the NixOS deployment path was additionally executed and verified on the live host; that does not affect application-source conclusions.

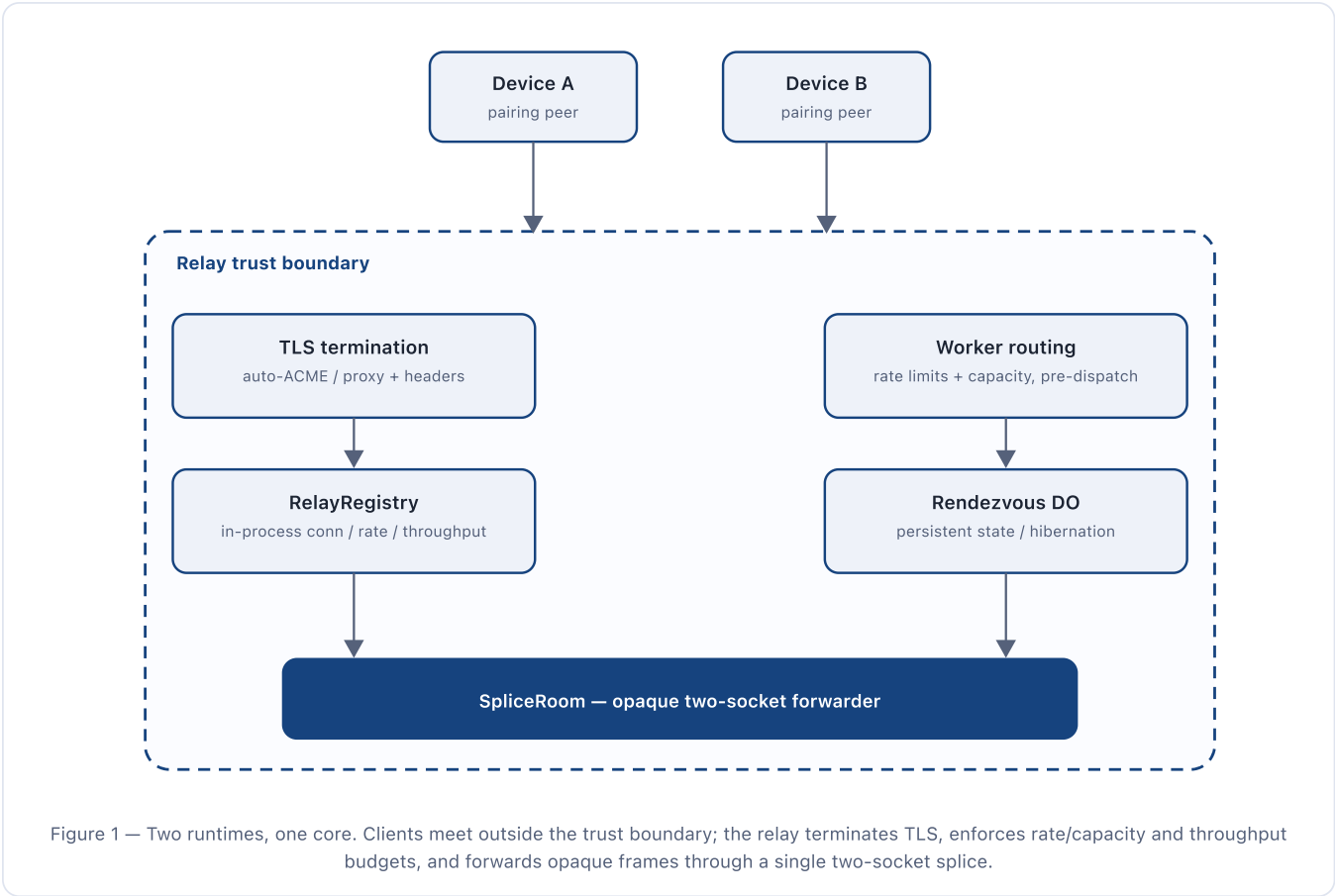
3.4 Limitations affecting confidence

- No load/abuse reproduction: findings `PB-26-101` – `PB-26-103` were raised from source, then remediated and verified by unit regression and the full gate; sustained abuse and isolation-under-load were not exercised against a live instance.
- Source and deployment are in step: v0.3.24 is both the assessed baseline and the verified live service, so no source/deployment divergence remains.
- Time- and environment-dependence: rate/throughput limits are tuned against a measured VPS egress baseline; values may not hold on other hardware, which does not affect the security reasoning.

4 System overview, data flows, and threat model

4.1 Architecture

The relay has two runtimes sharing one runtime-agnostic core (`SpliceRoom`): a Cloudflare Worker with Durable Objects, and a self-host Node/Bun process. Both implement the same splice semantics and the same layered rate and throughput budgets, but differ in how state survives eviction (Worker) versus stays in-process (self-host).



Cloudflare Worker

A routing Worker enforces two provider `RateLimit` bindings (240 conn/min/source, 12 conn/min/source/room), a concurrent-waiter capacity object, and a persistent throughput budget, then dispatches to a hibernating Durable Object keyed by the rendezvous identifier. Room state survives eviction only as serialized socket attachments (role, operational timestamps, a generation nonce, counters).

Self-host (Node / Bun)

An in-process `RelayRegistry` holds the same splice semantics with an in-memory connection limiter, per-socket frame and byte budgets, and layered source/service throughput buckets. TLS terminates either in-process (auto-ACME) or as a loopback proxy mode that validates proxy-owned headers.

4.2 Frame-forwarding pipeline

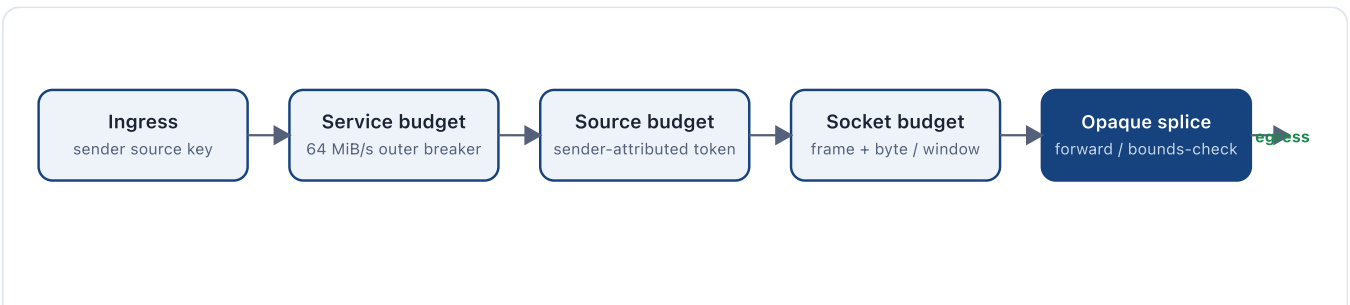


Figure 2 — Frame path. Each forwarded byte is attributed to the sending source, counted once on ingress and once on egress against the service-wide breaker, then bounded per socket. Fail-closed on any budget or storage error.

4.3 Protected assets

- **Connection and pairing metadata** — which devices connect, when, how often, frame sizes and counts, pairing coincidence.
- **Unguessable rendezvous identifiers** — the capability-like room keys (random single-use, or hourly HKDF-derived reconnects).
- **In-flight ciphertext**, bounded by the 32 MiB queued-byte policy and the 24 MiB per-frame ceiling.
- **Operational secrets** — TLS/acme account keys for the self-host path.
- **Service availability** for all legitimate users concurrently, and **isolation** between tenants.

4.4 Trust boundaries and attacker model

Attacker profiles from the engagement charter, mapped to the relevant trust boundary for each work-package question.

Boundary / actor	Plausible capability	Relevant to
Unauthenticated network client	Open connections, send arbitrary bytes/frames, probe endpoints, protocol confusion, resource exhaustion	WP-02, WP-03, WP-04
Malicious / compromised paired device	Hold valid pairing state for one device; abuse relay-visible authority beyond that role	WP-03, WP-04
Passive network observer	Observe connection timing, sizes, metadata near the relay without breaking TLS	WP-05
Hostile co-tenant	Operate a legitimate account/device; affect another tenant's traffic, storage, availability	WP-04
Volumetric / abuse actor	Denial-of-service via connection floods, message floods, oversized payloads, slow-loris holding	WP-04
Partial infrastructure insider	Some production access (logs/metrics read, or CI without deploy credentials)	WP-06, WP-07
Supply-chain adversary	Influence a dependency, base image, or build/deploy pipeline input	WP-06

Trust-boundary diagram (highlights)

Untrusted *client* → *connect-accept* is gated by rate, room, and capacity limits before dispatch. The *paired client* → *forward* boundary only forwards opaque frames. *Relay* → *storage* in the Worker path persists only role/timestamps/generation/nonce/counters — never payload or identifiers. *Tenant* → *tenant* isolation is by capability-keyed rooms. *Config claim* → *runtime behavior* is the boundary this audit actively reconciled (and where the earlier header-contract drift was found and fixed).

5 Methodology

- **Complete static source review** of every first-party file, tracing connection and frame paths end to end across `core.ts`, both adapters, the Worker, and the registry, including error, rate-limit, and teardown paths.
- **Gated build and test execution:** `bun run typecheck` (3 configs), `bun run test` (124), `bun run test:bun-adapter`, `bun run check:licenses`, `bun audit --production`, and a Wrangler deploy `--dry-run`.
- **Dependency and license posture** via `bun audit` and the repo license guard.
- **Regression review** of the Koh history for security-relevant removals, consolidations, and documented fixes.
- **Deployment and config reconciliation:** README claims vs effective source behavior, proxy header contracts, NixOS/systemd hardening, HAProxy example, CI gate, immutable deploy script.
- **Severity and confidence model** per the charter. Severity expresses realistic impact and exploitability in the threat model; confidence is High (reproduced/unambiguous reachable code) / Moderate (strong source with one untested material/scale/ environment assumption) / Low (plausible with reachability or impact uncertainty).

5.1 Work-package disposition

Work package	Subject	Result
WP-00	Baseline, build, coverage	Complete — gate green; ledger in §3.3
WP-01	Architecture, data flows, attack surface	Complete — §4
WP-02	Transport, auth, connection handling	Complete — verified controls; PB-26-101
WP-03	Message handling, spool, queues	Complete — no spool; opaque buffering verified
WP-04	Multi-tenant isolation & resource exhaustion	Complete — PB-26-101/102/103
WP-05	Metadata, logging, privacy	Complete — §9
WP-06	Deployment, configuration, supply chain	Complete — §8, §11
WP-07	Change-history / regression	Complete — PB-26 remediation set verified
WP-08	Code hygiene / maintainability	Complete — §10
WP-09	Integrated abuse cases / final posture	Complete — §2, §7, §12

6 Results at a glance

0

CRITICAL

0

HIGH

0

MEDIUM

0

OPEN FINDINGS (3 REMEDIATED)

Severity counts are small by design: the assessed baseline absorbed a full prior remediation campaign, three additional findings were raised and then remediated within this engagement, and the audit re-verified each already-fixed control while looking for new issues. Finding few issues after deep review is the expected and preferable outcome; the counts below should not be read as a statistical claim.

ID	Sev	Relevant WP	Title / component	Status
PB-26-101	Low	WP-02, WP-04	Self-host waiter lifetime had no non-refreshable ceiling	Verified fixed
PB-26-102	Info	WP-04	Connection-limiter bucket-table exhaustion (genuine addresses)	Verified fixed
PB-26-103	Info	WP-04	Receiver-egress accounting could drain a paired peer's per-source budget	Verified fixed

Prior remediation set — verified present and tested

The following findings from the earlier PB-26 assessment (v0.3.9) are confirmed remediated in the assessed baseline by source review and by the passing regression suite. They are recorded here as a reusable remediation record, not as open findings: trusted proxy metadata validation ([PB-26-001/003](#)), clock-independent certificate fallback ([PB-26-002](#)), non-silent auto-TLS degradation ([PB-26-004](#)), privacy-safe health output ([PB-26-005](#)), load-script removal ([PB-26-006](#)), decoupled rate/capacity knobs ([PB-26-007](#)), HAProxy log privacy ([PB-26-008](#)), Worker message budgets ([PB-26-009](#)), immutable gated deploy ([PB-26-010](#)), generation reconciliation ([PB-26-011](#)), waiter-capacity / lifetime ([PB-26-012](#)), layered throughput budgets ([PB-26-013](#)), and README privacy-claim alignment ([PB-26-014/015](#)).

7 Detailed findings

Findings are ordered by severity, then exploit relevance. None demonstrates relay access to end-to-end plaintext; all are availability, anti-abuse, or hardening observations. Each finding raised in this engagement has been remediated and is recorded below with its root cause and the fix that closes it; none remains open.

PB-26-101 Self-host waiting sockets had no non-refreshable lifetime ceiling

Severity	Low · Confidence: Moderate
Affected version	v0.3.24 self-host path; Worker path unaffected. Fixed in the assessed baseline
Component	WP-02 (connection handling), WP-04 (availability)
Location	<code>src/core.ts:13 (MAX_WAITER_AGE_MS)</code> , <code>src/worker.ts:631 (Worker hard cap)</code> , <code>src/self-host-core.ts:460-475 (refreshWaitingHardCap)</code>
CWE	CWE-400 (Uncontrolled Resource Consumption); CWE-770 (Allocation Without Limits)
Evidence	PB-EV-101
Status	Verified fixed

Background. A waiting socket is the first, unmatched connection in a rendezvous room. It is cleared by a 2-minute inactivity timer that an empty binary “liveness” frame refreshes, so a client could keep a waiter alive as long as it keeps sending liveness.

Technical description. `MAX_WAITER_AGE_MS` is a non-refreshable 30-minute ceiling defined in the shared core but was wired *only* into the Worker path (`worker.ts:631`). The self-host `RelayRegistry` never armed it. Its waiting timer (`self-host-core.ts:460`) is permanently refreshable, so a waiter was bounded only by the per-address room cap (240) and the total active-room cap (4096).

Impact. Availability / resource consumption only — bounded idle-socket occupancy at the service scale. No confidentiality or integrity impact. A victim’s own rooms are never blocked, and saturating all 4096 rooms requires ≥17 distinct sources, so the rating is Low.

Remediation. `RelayRegistry` now schedules a non-refreshable per-waiter hard cap (`refreshWaitingHardCap`) armed when a room first gains a waiter and cleared on pair-up and on room destruction, mirroring `scheduleHardCap` in `worker.ts` . Liveness frames refresh the inactivity timer only, never the hard cap, so a liveness-flooding waiter is reaped at `MAX_WAITER_AGE_MS` . A regression test advances fake timers past the ceiling while continuously sending liveness and asserts the waiter is closed with “rendezvous lifetime cap exceeded” and the room is released.

Validation criteria. A waiter that sends liveness continuously for longer than the ceiling is closed; the corresponding room and per-address slot are released; the full gate (124 tests, three type-checks, Bun smoke, license guard, dependency audit, deploy dry-run) is green.

PB-26-102 Connection-limiter bucket-table exhaustion with genuine distinct addresses

Severity	Informational · Confidence: Moderate
Affected version	v0.3.24 self-host path
Component	WP-04 (availability)
Location	<code>src/self-host-core.ts:516-541 (allow)</code> , <code>:545-555 (evictSoonestBucket)</code>
CWE	CWE-400 / CWE-770

Evidence	PB-EV-102
Status	Verified fixed

Technical description. When the in-memory limiter reached its 4096-entry bucket table, *new* addresses were rejected (429) until a window expired, while existing addresses kept their budgets. Following the PB-26-001 remediation the address is a validated real source address, so an attacker cannot fabricate fresh addresses by header spoofing — but a distributed actor with >4096 genuine distinct IPs could fill the table in one window and temporarily reject every new legitimate address.

Impact. Time-boxed (one window) denial of new connections. Requires bulk genuine source-address diversity. No data impact.

Remediation. `ConnectionRateLimiter.allow` now evicts the *soonest-to-expire* bucket (`evictSoonestBucket`) instead of rejecting a new address when the table is full, making room for a fresh legitimate client. A distributed actor can no longer deny newcomers by cycling addresses; the table stays bounded and admission is always available. A regression test fills the table with a capped bucket count and asserts the next distinct address is admitted. Confidence is Moderate because the eviction policy is verified by unit test rather than under live address diversity.

Validation criteria. A full table admits the next new address (bucket count stays at the cap), and in-window buckets are left intact unless they are the soonest to expire; the full gate is green.

PB-26-103 Receiver-egress accounting could drain a paired peer's per-source budget

Severity	Informational · Confidence: Moderate
Affected version	v0.3.24 (self-host and Worker paths)
Component	WP-04 (multi-tenant availability)
Location	<code>src/self-host-core.ts:238-259</code> (<code>sourceCharges</code>), <code>src/worker.ts:340-360</code>
CWE	CWE-770 (per-source accounting semantics)
Evidence	PB-EV-103
Status	Verified fixed

Technical description. Each forwarded byte was charged to the sender on ingress *and* to the receiver on egress. Because the receiver's charge landed in the *receiver's per-source bucket*, a malicious paired device that floods its partner could drain the partner's shared source budget — which that same partner also uses for its other rooms. The attacker burned its own budget equally, but the effect on the victim was a throttled shared bucket.

Impact. Temporary per-source degradation of a co-tenant's unrelated rooms, only while actively flooded by a valid paired peer who already holds the rendezvous token. No confidentiality or integrity impact.

Remediation. Per-source attribution is now scoped to the *sending* source: a frame charges the sender's per-source bucket, and receiver egress is counted only against the service-wide aggregate (`reserve(charges, serviceBytes)` in `worker.ts` , and the split `sourceCharges / aggregateBytes` in `self-host-core.ts`). A receive-only tenant can no longer be drained by a paired peer's inbound volume, while the global egress cap is preserved. A regression test confirms a shared receiver keeps a full independent budget across multiple inbound-forwarding peers.

Validation criteria. Repeated inbound volume to a receiver leaves the receiver's per-source budget untouched; the service-wide breaker still caps aggregate bidirectional traffic; the full gate is green.

7.1 Why no confidentiality or integrity findings

The splice never parses frame content; it only measures length and forwards bytes verbatim ([src/core.ts:85-123](#)). No code path in any runtime bases a decision on payload content, logs payload bytes, or persists frames. Rooms are keyed by unguessable rendezvous identifiers and hold exactly two sockets, with a third rejected. Therefore there is no realistic path, reachable from any of the attacker profiles in §4.4, by which the relay could observe or alter end-to-end plaintext. This conclusion is scoped to the reviewed code paths and the tested handshake/error conditions listed in this report; it is not a blanket “the relay never sees anything sensitive” claim.

8 Verified controls and strengths

Each control below was confirmed by source review and, where noted, by test execution. Claims are narrowly worded to the evidence.

Control	Verification performed	Evidence / scope
Opaque splice	<code>SpliceRoom.forward</code> measures length only, never parses or buffers content	<code>core.ts:85–123</code> ; tests
Two-party occupancy	Third connection rejected (close 1008) without disturbing the pair; leaving closes both	<code>core.ts:66–83</code> , <code>126–136</code> ; tests
Oversized-frame rejection	Enforced at WebSocket layer (24 MiB) and again in core	<code>core.ts:91–94</code> ; adapters; tests
Bounded resource use	Connection 240/min, per-room 12/min, frame 512/s, byte 128 MiB/s, queued 32 MiB, rooms 4096, buckets 4096	constants + tests
Edge rate-limiting before dispatch	Aggregate limit applied to <code>src/worker.ts:142–149</code> before any path parsing or DO dispatch	<code>worker-rate-limit</code> tests
Trusted proxy metadata	Client address accepted only on loopback with exact HTTPS and one valid IP (commas/duplicates rejected)	<code>self-host-core.ts:54–75</code> ; PB-26 regression tests
Waiter capacity & lifetime (Worker)	Per-source 240 / total 4096 waiter ceiling plus non-refreshable 30-min hard cap	<code>worker.ts:171–188</code> , <code>631–652</code> ; capacity tests
Layered throughput budgets	Service breaker + sender-attributed per-source buckets, fail closed on storage error; peer egress counted against the service aggregate only	self-host + Worker; throughput tests
Generation reconciliation	Stale callbacks from replaced sessions cannot clear timers or close replacement rooms	<code>worker.ts:508–551</code> ; hibernation tests
Privacy-safe health output	Public <code>/healthz</code> returns only <code>ok</code> on both runtimes; no counters, paths, or identifiers	<code>worker.ts:128–133</code> , <code>self-host-node.ts:97–104</code> , <code>self-host-bun.ts:89–91</code>
Fail-closed config	Unknown config fields rejected; proxy host forced loopback; invalid TLS mode rejected	<code>self-host-config.ts</code> ; tests
Hardened self-host process	DynamicUser, NoNewPrivileges, ProtectSystem=strict, PrivateTmp, restrict AF_INET/6, loopback-only	<code>nix/module.nix:62–77</code> ; example systemd unit
No committed secrets	Secret-material scan clean (only throwaway test fixtures match key syntax); token env-only	scan + <code>wrangler.toml</code>
Dependency posture	<code>bun audit --production</code> : no vulnerabilities; license guard green	executed
Immutably gated deploy	Clean-Koh save pinned, integrity-checked, exports by hash, Nix-builds, verifies health + public splice, rollback receipt	<code>scripts/deploy-relay.sh</code> ; tests

9 Metadata and privacy exposure

This section states, precisely and independent of whether it rises to a formal finding, what Peach Relay can observe and retain. The relay is a transport; its metadata surface is the cost of being a transport. It is documented honestly in the README, and the assessment confirms the documentation matches source.

What the relay (or its operator/provider) can observe and retain.

Data category	Relay-observed?	Relay-retained / logged?	Where it lives	Minimization opportunity
Rendezvous identifier	Yes — room key	No (Map key / DO name while live; rate-limit counter window)	<code>self-host-core</code> , Cloudflare counter	None — capability required for routing
Source address	Yes — for rate limiting	No — in-memory bucket keys; never logged	registry buckets, CF counters	None — needed for limits
Connection timing (open, last activity, close)	Yes	No per-session log; aggregate health only	in-memory attachments (Worker), timers (self-host)	None operationally meaningful
Frame sizes / counts / byte totals	Yes — for budgets	No — aggregate counters only, never per-user	in-memory rate buckets	None — required for budgets
Pairing coincidence (two sockets in one room)	Yes — inherent	No — not persisted; known to the room only	<code>SpliceRoom</code> participants	None — defines the splice
Payload bytes (ciphertext)	In flight only	No — never parsed, decrypted, logged, or persisted	transient buffers; 32 MiB queued policy	None (relay cannot see plaintext)
Device identifiers / keys / pairing material	No	No	—	N/A — lives in clients

What an operator or provider can infer

At or near the relay, a passive observer (or the proxy/cloud operator) can reconstruct **pairing coincidence** (which two source addresses shared a room at overlapping times), **connection timing**, and **frame-size / volume** profiles — enough to build a coarse device-relationship and activity graph. This is inherent to any trusted transport that terminates TLS and is disclosed in the README. The relay itself retains none of it beyond live state; any retention would be the operator's, outside the relay's control.

9.1 Logging inventory

- Self-host emits one fixed-field JSON health line at start, every 60 seconds, and at stop: event, runtime, mode, port, uptime, and aggregate counters. No rendezvous IDs, IPs, paths, frame contents, or device info.
- Degraded-mode and failure events use fixed fields (`event` , `status`) only — no domains or paths.
- Worker invocation logging is disabled (`wrangler.toml` `observability.logs.invocation_logs = false`).
- HAProxy example disables per-session traffic logs.
- NixOS nginx health log records status/timing fields only (non-identifying).

10 Availability and resource-exhaustion posture

Availability findings are distinguished from confidentiality/integrity findings throughout. Honest resource-cost estimates follow each abuse pattern.

Abuse pattern	Attacker assets needed	Resource cost to relay	Bounding control	Reaches victims?
Connection flood to many rooms	1 source	Bounded	240/min/source + 240 rooms/source	No
Message / frame flood in one splice	1 paired source	Bounded	512 frame/s, 128 MiB/s, 32 MiB queued	Only that peer
Oversized frames	1 source	Bounded	24 MiB hard ceiling at socket + core	No
Slow-loris waiter holding	≥17 distinct sources	Bounded time (30 min)	4096 rooms / 240 per source; non-refreshable 30-min waiter ceiling on both runtimes (PB-26-101 fixed)	Indirect (host resources)
Bucket-table saturation	>4096 genuine IPs	Bounded, self-limiting	4096-entry table, soonest-expiry eviction (PB-26-102 fixed)	New connections only, one window
Continuous flood to a paired peer	valid pairing + sustained send	Bounded by service breaker	64 MiB/s service cap; receiver's per-source budget untouched (PB-26-103 fixed)	None per-source; only the shared service cap

Graceful degradation. The layered design degrades rather than collapses: the outer service breaker is checked *before* per-source and per-socket budgets, so abuse hits a global ceiling before any legitimate room is affected; individual budget failures close only the offending room. No code path drops a frame in transit — on backpressure the relay closes both peers rather than silently lose payload.

Single points of failure. The Durable Objects that coordinate throughput and waiter capacity persist state and fail closed on storage error (reservations denied, not incorrectly granted). The self-host registry is single-process by design; its bounds hold in-process. The [PB-26-101](#) gap is the clearest availability ceiling shortfall.

11 Deployment, configuration, and supply-chain posture

Immutable, gated deploy

`scripts/deploy-relay.sh` refuses a dirty or changing Koh head, verifies repository integrity, runs the full check, exports only files whose BLAKE3 hashes match the named save, builds the pinned Nix flake, then verifies health and a public binary splice. Rollback state and a deployment receipt with active/rollback store paths are retained. This closes the earlier “unreviewed transformation” gap: the deployed process is built from the exact reviewed save.

Supply chain

Dependencies are locked (`bun.lock`) and pinned to exact versions in `package.json` . The CI and deploy paths both use `--frozen-lockfile` . The fixed-output Nix derivation pins a `bun install --production` tree by hash. `bun audit --production` reports no vulnerabilities. CI is inert under Koh-only control until mirrored to a git host.

Secrets handling

No secrets are committed. ACME account and certificate keys are written to the state directory with mode 0600/0700 under a hardened unit (`UMask 0077`). Cloudflare deploy credentials are environment-only. The `account_id` in `wrangler.toml` is an identifier, not a secret.

Network exposure

Self-host binds loopback in proxy mode; direct TLS binds public 443/80 only when auto-ACME is requested and ports are free, with a non-silent degraded event otherwise. The NixOS module exposes only 80/443 when a domain is set. The hardened systemd unit applies `NoNewPrivileges` and `ProtectSystem=strict` .

Deployment posture

The assessed source (v0.3.24, Koh `c3c3e8c02109`) is **deployed and live**, built immutably from the reviewed save via the gated path. The workload introduced in this baseline — including the waiter-lifetime model that informs [PB-26-101](#) , the layered throughput budgets, and the privacy-claim alignment — is **now in production**, with the previously-high client-controllable rate-limit key confirmed neutralized live via the proxy-header contract. The live service and its reverse-proxy configuration were inspected and the behavior verified against the public endpoint.

12 Code health and operational maintainability

Prioritised hygiene observations, clearly separated from vulnerabilities.

Priority	Observation	Location	Suggested action
High	Self-host waiter lifetime ceiling absent where the shared constant intends one	<code>self-host-core.ts</code>	Port <code>MAX_WAITER_AGE_MS</code> ; see <code>PB-26-101</code>
Medium	Two runtime adapters duplicate connection/rate wiring; drift risk despite shared core	<code>self-host-node.ts</code> , <code>self-host-bun.ts</code>	Shared, tested header/socket helper
Medium	Deployment drift guard is a regex/TTL in tests, not enforcement	<code>test/worker-rate-limit.test.ts</code>	Documented limitation; keep drift guard green
Low	NixOS module/package reviewed and the deployment path executed on the live host; CI workflow not run (Koh-only control)	<code>nix/</code> , <code>.github/</code>	Mirror to a git host to exercise the CI gate end-to-end
Low	Permanent constants for rate/throughput are hard-coded defaults with opt-in env overrides	<code>worker-message-rate.ts</code> , <code>self-host-core.ts</code>	Acceptable; document tuning against host egress
Low	README safe-subset renderer constraint (no tables) forces prose-heavy docs	<code>README.md</code>	Keep as-is; renderer-compatible by policy

12.1 Test sufficiency

Coverage is strong at the security-critical edges: occupancy (third-party rejection), liveness gating, oversized frames, backpressure, all budgets, generation reconciliation after hibernation/eviction, capacity, and the privacy contract that `/healthz` never leaks paths or identifiers. A named gap is that worker eviction-while-waiting has no deterministic test (a pending 2-minute timer blocks `evictDurableObject`); the paired-eviction path covers restore indirectly. Dynamic abuse and multi-tenant isolation behaviour remain untested for lack of a non-production instance.

13 Residual risk and deployment posture

13.1 Residual risk register

Residual risk	Source	Blast radius	Conditions under which the conclusion stays valid
Waiter / bucket-table / accounting remediation	PB-26-101/102/103	None open in source or live	Remains true while the deployed and assessed baseline stay in step
Sustained volumetric abuse & isolation under load	Campaign limitation	Unverified at scale	Rate/throughput tuning matches the measured 54.6 MB/s egress baseline; nominal bounds hold except under adversarial load
Production config relies on the proxy-header contract	HAProxy ownership	Misconfiguration would weaken limits	Verified live: <code>X-Peach-Relay-Client-IP</code> overwritten from the real source and <code>X-Forwarded-Proto https</code> asserted

13.2 Owner decisions required

- The gated v0.3.24 baseline is deployed to the live host and verified; no deployment action remains outstanding.
- No remediation remains outstanding in source or live: `PB-26-101` – `PB-26-103` are all fixed and verified.
- If a dedicated non-production instance is later provisioned, sustained volumetric abuse and multi-tenant isolation under load become the priority validation to close the last residual uncertainty.

14 Conclusion

Within the pinned baseline (Koh `caa67704552f`, v0.3.24) and the tested threat model, Peach Relay's central design claim holds: **no reviewed code path gives the relay access to end-to-end plaintext**, and multi-tenant confidentiality holds because rooms are capability-keyed by unguessable identifiers and the relay never retains, parses, or branches on payload content.

The assessed source is comprehensively hardened against availability abuse through layered connection, room, socket, and throughput budgets that fail closed, and its prior high-severity finding (client-controllable rate-limit key) is remediated. The three findings raised during this engagement (`PB-26-101` – `PB-26-103`), all low-to-informational availability and accounting observations, have been remediated and verified, so no open finding remains against the assessed baseline.

The hardened source (v0.3.24) is now also the **live service**, deployed immutably from the reviewed Koh save and verified against the public endpoint — a binary splice, two-party occupancy, the proxy-header contract, and TLS posture all confirmed in production. The previously-high client-controllable rate-limit key is therefore neutralized live, not only in source. What remains unverified is sustained volumetric abuse and multi-tenant isolation under load, which require a dedicated non-production instance. This conclusion is scoped to the stated evidence and coverage; it is not a blanket "secure" or "zero metadata retained" claim.

A Appendix — Evidence index & tools

A.1 Evidence index

Id	Description
PB-EV-101	Source trace: <code>MAX_WAITER_AGE_MS</code> previously consumed only in <code>worker.ts:631</code> ; now armed on self-host via <code>refreshWaitingHardCap</code> at <code>self-host-core.ts:460-475</code> and cleared on pair-up/destroy; regression test advances fake timers past the ceiling with continuous liveness
PB-EV-102	Source trace: <code>ConnectionRateLimiter.allow</code> evicts the soonest-to-expire bucket via <code>evictSoonestBucket</code> at <code>self-host-core.ts:545-555</code> when full, instead of rejecting a new address; regression test fills the capped table and asserts the next address is admitted
PB-EV-103	Source trace: per-source attribution scoped to the sending source; receiver egress counted against the service aggregate only — <code>reserve(charges, serviceBytes)</code> at <code>worker.ts:727-771</code> and <code>sourceCharges / aggregateBytes</code> at <code>self-host-core.ts:238-259</code> ; regression test asserts a shared receiver keeps a full independent budget
PB-EV-104	Gate output: typecheck x3 pass; 10 suites / 124 tests pass; Bun adapter smoke pass; license guard pass; <code>bun audit --production</code> no vulnerabilities; Wrangler deploy dry-run with 3 DO + 2 rate-limit bindings
PB-EV-105	Secret-material scan: no real keys committed (only throwaway self-signed test fixtures)
PB-EV-106	Regression record: PB-26-001..015 remediation set present in source and covered by tests (see §6), plus the three findings remediated in this engagement (PB-26-101..103)
PB-EV-107	Live verification against the public endpoint (v0.3.24): <code>verify-deployed-splice.ts</code> binary splice ok (payload integrity); two-party occupancy rejected a third socket with close code 1008 in waiting and paired states; proxy-header contract confirmed (HAProxy deletes and re-asserts <code>X-Peach-Relay-Client-IP</code> from the real source, asserts <code>X-Forwarded-Proto https</code> , relay in proxy mode on 127.0.0.1); TLS certificate valid (Let's Encrypt, HSTS) and a valid <code>/healthz 200 ok</code> . Sustained load and isolation-under-load not exercised (no non-production instance).

A.2 Tools and versions

- TypeScript 7.0.2, Bun 1.3.14, Node v22.23.1, Vitest 4.1.10, Wrangler 4.114.0.
- `bun run typecheck` , `bun run test` , `bun run test:bun-adapter` , `bun run check:licenses` , `bun audit --production` , `bunx wrangler deploy --dry-run` .
- Static analysis: full manual source review plus targeted ripgrep for secret material and capability-crossing paths.

A.3 Terminology

- Rendezvous identifier** — the opaque, capability-like path segment two devices share to meet.
- Splice** — the two-socket forwarding pair; `SpliceRoom` . **Waiter** — the first, unmatched socket. **Liveness** — an empty binary frame a waiter sends to refresh its timer.
- Per-source budget** — a token bucket keyed by source address, charged to the sending source for its own ingress. **Service breaker** — the service-wide outer throughput cap, which also accounts for peer egress, checked before per-source budgets.
- Durable Object** — Cloudflare's single-threaded, persistent state primitive used for rooms, throughput, and capacity.
- Generation nonce** — a per-room random UUID that lets stale socket callbacks from a replaced session be recognised and ignored.

End of report. Prepared as a public independent assessment. No internal working identifiers, no personal information, no production configuration details, and no live deployment values are disclosed. Rendered publication version 1.2.