Zzyzx

LIQUIDLINK / FINAL MANUAL SECURITY AUDIT

Liquidlink Manual Security Audit

Score accounting, Stamp and policy settlement, point tokenization, point trading, and extension lifecycle

24	15	9	0
CANONICAL FINDINGS	RESOLVED OR REMOVED	ACKNOWLEDGED / CONDITIONAL	OPEN

Report date	2026-08-17
Assessment baseline	64a477bf9ae945ff585f9dd70231945a9a741ea9 plus reviewed working-tree changes
Primary scope	Liquidlink core and admin packages, point-trading orderbook, Referral, Nemo Staking, and Check-in policies
Classification	Internal Final Security Assessment

Zzyzx Security Review

Contents

1. Executive Assessment

Assessment outcome

The reviewed Liquidlink canonical implementation contains no currently Open implementation finding in this ledger. Residual exposure is concentrated in explicitly acknowledged governance, keeper, economic, and campaign-policy assumptions. These assumptions remain release and monitoring obligations even when no code change is required.

Status matrix

The matrix includes Liquidlink core and policy-ecosystem findings from the full historical review corpus, not only recent or unresolved issues.

14 RESOLVED	**1** ARCHITECTURE ELIMINATED	**9** ACKNOWLEDGED
0 CONDITIONAL / PENDING	**0** OPEN	**0** DEPLOYMENT EXCLUDED

Current disposition

Resolved or architecture-eliminated: 15. Acknowledged or conditional: 9. Open: 0. Deployment-excluded: 0.

2. Architecture, Trust Model, and Invariants

Liquidlink separates score storage from authenticated score production. A Project binds one Scoreboard, authorized source types, an ordered policy pipeline, and exact policy registrations. Sources create a LiquidlinkStamp hot-potato; every required policy settles in configured order before the Scoreboard applies the final operation. Point tokenization materializes eligible score and point trading exchanges point tokens against coin quote assets.

Primary invariants

1. Every Stamp binds the exact Project object, logical project, Scoreboard, source, operation, and nonce.
2. Policy settlement binds exact policy state and registration generation and must follow configured order.
3. A Stamp cannot settle until every required policy has completed exactly once.
4. Tokenized score, holding registry, TreasuryCapStore, and orderbook market must agree on the exact point-token type and project.
5. Orderbook custody, maker fee reserve, taker fee, cancellation, and claim paths conserve both traded assets.
6. Campaign close stops future accrual without confiscating already earned claims.

Trust model

Liquidlink administration can register policies, configure fees, issue or delegate selected capabilities, operate keepers, and close campaigns. Multisig custody, narrow ACL roles, event monitoring, and staged changes are required controls. Referral bonus issuance and Check-in budget behavior are intentional economic assumptions rather than value-conservation bugs.

3. Complex Remediation Evidence

3.1 Exact project, policy state, and generation binding

PolicySettlementCap validation checks the exact Project object, logical project, Scoreboard, policy type, state object, and registration generation. Unregistering and re-registering a policy invalidates the old cap.

Resolved production excerpt: project.move lines 374-389

```
374 public(package) fun assert_policy_settlement_cap<P: drop>(
```

```
    project: &Project,
    policy_state_id: ID,
    cap: &PolicySettlementCap<P>,
) {
    assert_enabled(project);
    let policy = type_name::with_original_ids<P>();
    assert!(cap.project_object_id == object::id(project), EProjectMismatch);
    assert!(cap.project_id == project.project_id, EProjectMismatch);
    assert!(cap.scoreboard_id == project.scoreboard_id, EScoreboardMismatch);
    assert!(cap.policy == policy, EPolicyMissing);
    assert!(cap.policy_state_id == policy_state_id, EPolicyStateMismatch);
    let registration = active_registration(project, policy);
    assert!(registration.policy_state_id == cap.policy_state_id, EPolicyStateMismatch);
    assert!(registration.generation == cap.registration_generation, EGenerationMismatch);
}
```

3.2 Ordered Stamp policy settlement

The policy cursor advances only when the next configured registration matches the exact policy type, state, and generation. A missing, duplicated, stale, or out-of-order policy cannot silently settle the Stamp.

Resolved production excerpt: stamp.move lines 676-700

```
    assert!(project::id(project) == stamp.project_object_id, EWrongProject);
    assert!(project::project_id(project) == stamp.project_id, EWrongProject);
    assert!(project::scoreboard_id(project) == stamp.scoreboard_id, EWrongProject);
}

fun contains_registration(
    values: &vector<PolicyRegistration>,
    value: &PolicyRegistration,
): bool {
    let mut i = 0;
    while (i < values.length()) {
        let candidate = &values[i];
        if (
            project::policy_registration_policy(candidate)
                == project::policy_registration_policy(value)
                && project::policy_registration_state_id(candidate)
                    == project::policy_registration_state_id(value)
                && project::policy_registration_generation(candidate)
                    == project::policy_registration_generation(value)
        ) {
            return true
        };
        i = i + 1;
    };
    false
```

3.3 Fragmentation-resistant maker fee accounting

Maker fee is computed from cumulative filled notional and charged as the delta from fee already paid. Splitting one economic fill no longer multiplies ceil rounding per partial fill.

Resolved production excerpt: orderbook.move lines 1400-1408

```
fun maker_fee_for_fill<T>(order: &mut Order<T>, quote_amount: u64): u64 {
    let filled_before = order.maker_quote_filled;
    let filled_after = point_math::checked_add(filled_before, quote_amount);
    let fee_before = coin_quote_fee_amount_bps(filled_before, order.maker_fee_bps);
    let fee_after = coin_quote_fee_amount_bps(filled_after, order.maker_fee_bps);
    order.maker_quote_filled = filled_after;
    fee_after - fee_before
}

```

4. Finding Register

The register below contains all 24 Liquidlink and policy-ecosystem canonical findings assigned from the historical audit corpus.

Liquidlink Findings

ID / ORIGINAL SEVERITY	FINDING / CURRENT DISPOSITION / REMEDIATION
LL-01 Medium	Exact-in orderbook arithmetic could overflow Resolved - Resolved Use checked full-width arithmetic and reject unrepresentable exact-in values.
LL-02 Medium	Custody mode retained an ambiguous dust head Resolved - Resolved Consolidate custody into PointTokenMarket and define deterministic dust ownership.
LL-03 Medium	Token holding registry accounting could be bypassed Resolved - Resolved Exact-bind token custody, holding registry, project, and orderbook identities.
LL-04 Medium	Best-fill execution lacked an intrinsic limit-price guard Resolved - Resolved by PTB Composition Compose a PTB slippage guard after execution; document it as the canonical safety path.
LL-05 Medium	Fee configuration permitted a 100 percent rate Acknowledged - Accepted Admin Configuration Keep admin-set fee configuration bounded operationally and surface it in governance tooling.
LL-06 Medium	TGE redemption could drain mismatched project assets Architecture Eliminated - Architecture Removed Remove the TGE architecture and its asset-redemption path.
LL-07 Medium	PointCap created an unbounded privileged issuance lane Acknowledged - Accepted Privileged Lane / Hardened Isolate PointCap as a privileged manual lane; canonical protocol rewards use witnesses.
LL-08 Medium	Project version was coupled to the package version Resolved - Resolved Give Project and policy packages independent GlobalConfig version control.
LL-09 Low	Keeper cancellation expands governance authority Acknowledged - Accepted Governance Cleanup Keep keeper cancellation explicit, evented, and scoped through Liquidlink ACL.
LL-10 Low	Orderbook storage and gas lifecycle was insufficiently bounded Acknowledged - Accepted / Mitigated Use KeyedBigVector, price levels, fill limits, level caps, and keeper cleanup.
LL-11 Low	Counterparty registry was not exact-bound to the market Resolved - Resolved Consolidate exact registry and custody binding into PointTokenMarket.
LL-12 Low	Orderbook events mixed gross and net amounts Resolved - Resolved Emit gross input, fees, and net output as distinct event fields.
LL-13 Info	Orderbook fee recipient semantics were inconsistent Resolved - Resolved Route maker and taker fees into one configured treasury source of truth.
LL-14 Medium	A 100 percent redirect could produce a zero add operation Resolved - Resolved Reject zero-result redirects and enforce operation-specific policy semantics.
LL-15 Medium	Multiple policies could overwrite the same linear stream Resolved - Resolved Use ordered policy settlement so only the intended stage may update a linear stream.
LL-16 Medium	Policy unregister did not retire existing linear streams Resolved - Resolved Checkpoint and terminate affected source streams when policy registration closes.
LL-17 Medium	Legacy PointCap issuance could bypass pause and version controls Resolved - Resolved Apply Liquidlink package version and pause checks to the privileged PointCap lane.

Policy and Auxiliary Findings

ID / ORIGINAL SEVERITY	FINDING / CURRENT DISPOSITION / REMEDIATION
AUX-01 Medium	Referral bonus model intentionally increases total issuance Acknowledged - Accepted Economic Model Document the bonus model: the referee keeps 100 percent and the referrer receives additional issuance.
AUX-04 Medium	Nemo unstake completion depends on a keeper Acknowledged - Accepted Liveness + Version Hardened Use a keeper-completed unstake queue with explicit version, pause, and ownership checks.
AUX-05 Medium	Daily check-in has an accepted Sybil and budget boundary Acknowledged - Accepted Economic Risk / Hardened Require position-based eligibility and per-position replay protection; accept the residual economy risk.
AUX-07 Medium	Nemo staking tier and duration transitions were asymmetric Resolved - Resolved Checkpoint source-owned base streams before tier changes without changing referral bonuses.
AUX-08 Low	Check-in reset buckets could be misconfigured Acknowledged - Accepted Governance Validate day length and reset offset and prevent duplicate bucket claims.
AUX-09 Low	A zero referrer could receive or route bonus accounting Resolved - Resolved Reject zero referrers and self-referral before writing binding state.
AUX-11 Low / Operational	Referral pause may intentionally block detach or transfer Acknowledged - Accepted Accept fail-closed pause semantics; campaign close remains a distinct graceful lifecycle.

5. Accepted Boundaries and Release Gates

Accepted boundaries

The accepted population includes PTB-level best-fill limit protection, administrator-configured fees, privileged manual PointCap issuance, keeper cancellation, referral bonus issuance, Nemo unstake completion, Check-in budget and reset assumptions, and pause behavior that may intentionally block policy detach. Acceptance requires documentation, monitoring, and operational discipline.

Release gates

1. Freeze exact package and dependency versions.
2. Restrict AdminCap, UpgradeCap, TreasuryCap, and keeper roles to approved multisig or operational keys.
3. Verify every active Project policy sequence and registration generation.
4. Validate point-token market type, quote coin, fee recipient, minimum notional, and order cleanup policy.
5. Exercise pause, campaign close, claim-after-close, policy unregister, stale-cap rejection, order cancellation, and treasury withdrawal runbooks.
6. Re-run Liquidlink, policy, point-token, and orderbook Move and formal suites from a clean checkout.

Limitations

This assessment does not prove honest administration, bounded Sybil participation, market demand, keeper availability, frontend transaction composition, indexer completeness, or the economic value of tokenized points.