Zzyzx

LIQUIDLINK / FINAL FORMAL VERIFICATION REPORT

Liquidlink Formal Verification

Project binding, policy order, generation, Scoreboard accounting, referral, staking, check-in, and orderbook evidence

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 Project and policy models, point-accounting models, orderbook models, Move regressions, and domain finding crosswalk
Classification	Internal Final Security Assessment

Zzyzx Security Review

Contents

01 Executive result and domain status

02 Verification scope and evidence taxonomy

03 Project, policy, and accounting proofs

04 Orderbook and lifecycle evidence

05 Manual-audit traceability (reference only)

06 Trusted boundaries and reproduction

1. Executive Result

191/191	24	3	0
SHARED SUITE PROOFS	LIQUIDLINK FINDINGS MAPPED	PRIMARY DOMAIN MODELS	OPEN IMPLEMENTATION ROOTS

Formal conclusion

Liquidlink Project binding, strict policy order, registration generation, score and referral accounting, check-in replay protection, and orderbook resource and fee invariants are represented in the shared verified suite and companion Move regressions. Residual risk is principally model fidelity and acknowledged governance or economic assumptions.

Manual Audit Status Reference

Why it is included: this reference confirms that the formal scope accounts for the complete known Liquidlink and policy-ecosystem finding population. The counts are copied from the separate Manual Security Audit; they are not findings or dispositions produced by Formal Verification.

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

2. Verification Scope and Evidence Taxonomy

Primary domain models are liquidlink_project_model, score_policy_model, and orderbook_model, supplemented by accounting_model and executable tests in Liquidlink, Referral, Nemo Staking, and Check-in packages.

Evidence codes

D - direct production proof.
M - mapped protocol model.
T - executable Move regression or attack test.
S - structural ABI or deployment evidence.
E - external, governance, economic, or operational boundary.

Interpretation

A passing policy or accounting model does not independently prove the production transaction graph. Closure depends on exact production mapping, negative tests, capability and object binding, and mutation sensitivity.

3. Project, Policy, and Accounting Proofs

3.1 Unique Project object and policy generation

The model rejects sibling Project objects, mismatched policy state, stale registration generation, and reactivated old settlement caps.

Formal excerpt: liquidlink_project_model.move lines 80-117

```
80  fun stamp_is_bound_to_unique_project_object_spec(
81      stamp: Stamp,
82      sibling_project_object_id: u64,
```

```
) {
    requires(stamp.project_object_id != sibling_project_object_id);
    ensures(!can_settle(&stamp, sibling_project_object_id, stamp.scoreboard_id));
}

#[spec(prove)]
fun policy_cap_is_bound_to_project_state_and_generation_spec(
    cap: PolicyCap,
    registration: PolicyRegistration,
    scoreboard_id: u64,
) {
    ensures(implies(
        cap.project_object_id != registration.project_object_id
            || cap.policy_state_id != registration.policy_state_id
            || cap.generation != registration.generation
            || cap.policy_type != registration.policy_type
            || cap.scoreboard_id != scoreboard_id,
        !cap_matches(&cap, &registration, scoreboard_id),
    ));
}

#[spec(prove)]
fun reregistered_policy_invalidates_old_cap_spec(
    cap: PolicyCap,
    registration: PolicyRegistration,
    scoreboard_id: u64,
) {
    requires(cap.generation != registration.generation);
    ensures(!cap_matches(&cap, &registration, scoreboard_id));
}

#[spec(prove)]
fun policy_order_is_strict_spec(stamp: Stamp, registration: PolicyRegistration) {
    requires(registration.order != stamp.next_policy_index);
    ensures(!policy_applicable(&stamp, &registration));
```

3.2 Complete ordered policy pipeline

Settlement is impossible before all required policies complete in exact configured order. The completed Stamp settles only once.

Formal excerpt: liquidlink_project_model.move lines 121-161

```
fun every_required_policy_must_complete_before_settlement_spec(
    stamp: Stamp,
    expected_project_object_id: u64,
    expected_scoreboard_id: u64,
) {
    requires(stamp.next_policy_index < stamp.required_policy_count);
    ensures(!can_settle(&stamp, expected_project_object_id, expected_scoreboard_id));
}

#[spec(prove)]
fun completed_policy_pipeline_settles_once_spec(
    stamp: Stamp,
    expected_project_object_id: u64,
    expected_scoreboard_id: u64,
) {
    requires(can_settle(&stamp, expected_project_object_id, expected_scoreboard_id));
    let next = settle(stamp, expected_project_object_id, expected_scoreboard_id);
    ensures(next.settled);
    ensures(!can_settle(&next, expected_project_object_id, expected_scoreboard_id));
}

#[spec(prove)]
fun subtract_operation_carries_no_referral_redirect_spec(amount: u64) {
    let referral_redirect = if (OP_SUBTRACT == OP_ADD) amount else 0;
```

```
    ensures(referral_redirect == 0);
}

#[spec(prove)]
fun referral_bonus_does_not_reduce_user_score_spec(amount: u64, referral_bps: u64) {
    requires(referral_bps <= 10_000);
    let bonus = ((amount as u128) * (referral_bps as u128) / 10_000) as u64;
    let user_score = amount;
    ensures(user_score == amount);
    ensures(bonus <= amount);
}

#[spec(prove)]
fun policy_cutoff_stops_only_dependent_stream_spec(
    dependent: bool,
    cutoff_ms: u64,
    now_ms: u64,
```

3.3 Referral and staking accounting

Referral bonus preserves the user's gross score and carries fractional remainder. Staking tier changes checkpoint the old multiplier before the new tier becomes effective.

Formal excerpt: score_policy_model.move lines 264-321

```
fun referral_bonus_preserves_user_and_fraction_spec(
    gross_points: u64,
    referral_bps: u64,
    previous_remainder: u64,
) {
    requires(referral_bps <= BPS_DENOMINATOR);
    requires(previous_remainder < BPS_DENOMINATOR);
    let (bonus, next_remainder) = referral_bonus_step(
        gross_points,
        referral_bps,
        previous_remainder,
    );
    ensures(next_remainder < BPS_DENOMINATOR);
    ensures(
        bonus.to_int().mul(BPS_DENOMINATOR.to_int()).add(next_remainder.to_int())
            == gross_points.to_int().mul(referral_bps.to_int())
                .add(previous_remainder.to_int()),
    );
    // Bonus model: the referee keeps gross_points; the referrer receives bonus.
    ensures(gross_points.to_int().add(bonus.to_int()).gte(gross_points.to_int()));
}

fun weighted_points(elapsed_ms: u64, rate: u64, multiplier_bps: u64): u128 {
    let result = (elapsed_ms as u256) * (rate as u256) * (multiplier_bps as u256)
        / (BPS_DENOMINATOR as u256);
    assert!(result <= (std::u128::max_value!() as u256));
    result as u128
}

#[spec(prove)]
fun staking_tier_change_is_checkpointed_spec(
    first_elapsed_ms: u64,
    second_elapsed_ms: u64,
    rate: u64,
    old_multiplier_bps: u64,
    new_multiplier_bps: u64,
) {
    requires(old_multiplier_bps >= BPS_DENOMINATOR);
    requires(new_multiplier_bps >= BPS_DENOMINATOR);
    requires(
        first_elapsed_ms.to_int().mul(rate.to_int())
            .mul(old_multiplier_bps.to_int())
```

```
            .div(BPS_DENOMINATOR.to_int())
            .add(
                second_elapsed_ms.to_int().mul(rate.to_int())
                    .mul(new_multiplier_bps.to_int())
                    .div(BPS_DENOMINATOR.to_int()),
            )
            .lte(std::u128::max_value!().to_int()),
    );
    let before_change = weighted_points(
        first_elapsed_ms,
        rate,
        old_multiplier_bps,
    );
    let after_change = weighted_points(
        second_elapsed_ms,
        rate,
```

4. Orderbook and Lifecycle Evidence

Orderbook models cover custody conservation, exact-output fee interpretation, resource limits, and terminal fee handling. Move tests cover maker and taker fees, cumulative maker-fee debt, minimum notional, level and market caps, bounded fills, keeper cancellation, owner cancellation, and claim paths.

Check-in replay boundary

The Check-in model blocks replay by both user-day and position-day. A transferred position cannot be reused to create a second same-day claim.

Formal excerpt: score_policy_model.move lines 336-371

```
fun can_check_in(
    state: &CheckInState,
    current_day: u64,
    position_eligible: bool,
): bool {
    position_eligible
        && state.user_last_day < current_day
        && state.position_last_day < current_day
}

fun record_check_in(mut state: CheckInState, current_day: u64): CheckInState {
    assert!(can_check_in(&state, current_day, true));
    state.user_last_day = current_day;
    state.position_last_day = current_day;
    state
}

#[spec(prove)]
fun check_in_blocks_user_and_transferred_position_replay_spec(
    previous_user_day: u64,
    previous_position_day: u64,
    current_day: u64,
) {
    requires(previous_user_day < current_day);
    requires(previous_position_day < current_day);
    let state = CheckInState {
        user_last_day: previous_user_day,
        position_last_day: previous_position_day,
    };
    ensures(can_check_in(&state, current_day, true));
    let state = record_check_in(state, current_day);
    ensures(!can_check_in(&state, current_day, true));
    // Changing the owner off-model cannot change the position keyed day.
    ensures(state.position_last_day == current_day);
}

```

Residual formal boundary

No Liquidlink-domain implementation root is currently Open in this ledger. Formal evidence does not prove honest administration, Sybil resistance, economic value, keeper availability, frontend PTB composition, or indexer completeness.

5. Manual Audit Traceability Matrix

Why this matrix appears in a Formal Verification report

This is not a second manual assessment. It maps every Liquidlink and policy-ecosystem finding to the proof, model, executable test, structural evidence, or trusted boundary that addresses it. The matrix is retained to demonstrate coverage completeness and to expose findings that have no formal evidence. The MANUAL STATUS* column is copied from the separate Manual Security Audit for reference only. Formal Verification cannot lower severity, close a finding, or replace the project's documented risk decision.

All 24 Liquidlink and policy-ecosystem canonical findings are mapped below for completeness and evidence traceability. *Manual status is quoted from the separate Liquidlink Manual Security Audit and is not a formal-verification conclusion.

Liquidlink Findings

ID	MANUAL STATUS*	EVIDENCE	FORMAL COVERAGE / RESIDUAL
LL-01	Resolved	M/T	Model fidelity to production transitions remains a trusted mapping boundary.
LL-02	Resolved	M/T	Model fidelity to production transitions remains a trusted mapping boundary.
LL-03	Resolved	M/T	Model fidelity to production transitions remains a trusted mapping boundary.
LL-04	Resolved	T/E	External or operational assumptions remain trusted boundaries.
LL-05	Acknowledged	M/T/E	External or operational assumptions remain trusted boundaries.
LL-06	Architecture Eliminated	S/T	Executable or structural evidence only; no universal theorem is claimed.
LL-07	Acknowledged	M/T/E	External or operational assumptions remain trusted boundaries.
LL-08	Resolved	M/T	Model fidelity to production transitions remains a trusted mapping boundary.
LL-09	Acknowledged	M/T/E	External or operational assumptions remain trusted boundaries.
LL-10	Acknowledged	M/T	Model fidelity to production transitions remains a trusted mapping boundary.
LL-11	Resolved	M/T	Model fidelity to production transitions remains a trusted mapping boundary.
LL-12	Resolved	M/T	Model fidelity to production transitions remains a trusted mapping boundary.
LL-13	Resolved	M/T	Model fidelity to production transitions remains a trusted mapping boundary.
LL-14	Resolved	M/T	Model fidelity to production transitions remains a trusted mapping boundary.
LL-15	Resolved	M/T	Model fidelity to production transitions remains a trusted mapping boundary.
LL-16	Resolved	M/T	Model fidelity to production transitions remains a trusted mapping boundary.
LL-17	Resolved	M/T	Model fidelity to production transitions remains a trusted mapping boundary.

Policy and Auxiliary Findings

ID	MANUAL STATUS*	EVIDENCE	FORMAL COVERAGE / RESIDUAL
AUX-01	Acknowledged	M/T/E	External or operational assumptions remain trusted boundaries.
AUX-04	Acknowledged	M/T/E	External or operational assumptions remain trusted boundaries.
AUX-05	Acknowledged	M/T/E	External or operational assumptions remain trusted boundaries.
AUX-07	Resolved	M/T	Model fidelity to production transitions remains a trusted mapping boundary.
AUX-08	Acknowledged	M/T/E	External or operational assumptions remain trusted boundaries.
AUX-09	Resolved	M/T	Model fidelity to production transitions remains a trusted mapping boundary.
AUX-11	Acknowledged	M/T/E	External or operational assumptions remain trusted boundaries.

6. Trusted Boundaries and Reproduction

Trusted boundaries include AdminCap and UpgradeCap custody, policy and fee governance, keeper liveness, referral and Check-in economics, frontend sequencing, quote-coin liquidity, point-token market demand, and completeness of the policy specification.

Reproduction

Run the shared verified suite and mutation checks from the frozen release commit, then run Liquidlink, Referral, Nemo Staking, Check-in, point-token, and orderbook Move packages from a clean checkout. Preserve verifier, compiler, and dependency versions with the release evidence.