

Formal Verification Final Report



Fluid Protocol

November 2026

Prepared for Fluid / Mysten Labs

certora

Table of Contents

Project Summary **3**
Project Scope 3
Project Overview 3
Protocol Overview 4
Threat and Security Overview **5**
Findings Summary 20
Severity Matrix 20
Detailed Findings **21**
Informational Issues 22
I-01. Tick placement records debt that no position owes 22
I-02. Rebalance rounds the recorded supply down and leaves it under-backed at liquidity 23
Formal Verification **24**
Verification Methodology 24
Verification Notations 25
General Assumptions and Simplifications 25
Formal Verification Properties Overview 26
Detailed Properties 29
Liquidity Solvency 29
Liquidity Accounting 33
Liquidity Reserve Invariants 36
Liquidity Access Control 38
Lending Accounting 41
Lending Rebalance 44
Lending Solvency 45
Vaults Solvency 47
Vaults Accounting 52
Vaults Liquidation 56
Vaults Library Integrity 64
Timelock Operation Lifecycle 66
Timelock Access Control 69
Timelock Role Governance 72
Timelock Cap Custody 75
Timelock Protocol Wrappers 78
Timelock Protocol Gates 82
Disclaimer **89**
About Certora **89**

Project Summary

Project Scope

Project Name	Repository	First Commit Hash	Latest Commit Hash	Platform
Fluid	https://github.com/Instadapp/fluid-packages-sui	b4dd971	c45d994	Sui / Move

Project Overview

This document describes the specification and verification of the **Fluid Liquidity, Lending and Vaults layers and Timelock** using the Certora Prover. The work was undertaken from **August 31st** to **November 6th.**

The following contract list is included in our scope:

packages/liquidity/**
packages/lending/**
packages/vaults/**
packages/timelock/**

During the verification process, the Certora team discovered informational issues in the Move contracts code, as listed on the following pages.

Protocol Overview

Fluid is a modular DeFi protocol developed by Instadapp around a shared liquidity foundation. Instead of isolating capital across separate products, its architecture allows multiple protocols to draw from the same reserves while maintaining their own accounting and risk controls. On Ethereum, Fluid supports lending markets, collateralized vaults, and decentralized exchange functionality. On Solana, the same model has been implemented through Jupiter Lend, with dedicated programs for liquidity, lending, vaults, oracles, and flash loans. At the center of the Sui implementation is the Liquidity Layer, which provides the system's shared accounting and custody infrastructure. It holds token reserves, records authorized protocols' supply and borrow positions, accrues interest through exchange prices, and enforces utilization, withdrawal, and borrowing limits. End users do not interact with it directly; registered protocols route supply, withdrawal, borrowing, and repayment operations through a common interface. This structure reduces liquidity fragmentation and allows capital to be shared across the wider Fluid system. Lending, vault, flash-loan, and other protocol modules are intended to be built on top of this foundation. The Vaults module adds collateralized borrowing. Each market pairs one collateral token with one debt token, and has its own oracle and risk settings. The vault is an integrator of the Liquidity Layer: it holds one supply position and one borrow position there, and routes all user operations through them. Users deposit, withdraw, borrow and pay back via a call to a single function, and every withdrawal or borrow must pass a health check against the collateral factor. Positions are grouped into ticks by their debt-to-collateral ratio. Liquidation is permissionless and processes whole ticks, so its cost does not depend on the number of positions. A rebalancer keeps the vault's records in line with its positions at the Liquidity Layer. The Timelock module puts a delay in front of governance. The liquidity, lending and vaults layers hand their timelock capabilities to a shared CapStore, so their admin actions run only through scheduled operations. Admin-class actions, including role changes and package upgrades, wait at least 12 hours, and configuration changes at least 6 hours. Separate roles propose, execute and cancel operations, and guardians can cancel harmful changes.

Threat and Security Overview

The Fluid liquidity layer has one solvency goal. The reserve holds coins in its vault. It records supplier claims and borrower obligations in two with-interest buckets and two interest-free buckets. In every reachable state, the supplier claims never exceed the borrower obligations plus the coins in the vault, at the stored exchange prices.

```
ts_if * 1e12 + ts_wi * sep <= tb_if * 1e12 + tb_wi * bep + vault * 1e12
```

where `ts_wi` and `ts_if` are the with-interest and interest-free supply totals, `tb_wi` and `tb_if` the borrow totals, `sep` and `bep` the supply and borrow exchange prices, and `vault` the balance of the reserve.

The vaults sit on the liquidity layer as one more integrator. Each `VaultMarket<Col, Debt>` holds one supply position in the collateral reserve and one borrow position in the debt reserve. It records its users' positions as collateral and debt placed on ticks. The vaults add two solvency goals. First, at the oracle rate, the collateral that the vault records covers the debt that it records:

```
total_supply * sp * oracle >= total_borrow * bp * 1e15
```

Second, the vault's books are backed by its own positions at liquidity. The debt that it records covers its borrow at liquidity, and the collateral that it records does not exceed its supply at liquidity:

```
total_borrow * bp >= liq_borrow * 1e12
```

```
total_supply * sp <= liq_supply * 1e12
```

where `total_supply` and `total_borrow` are the vault's raw collateral and debt totals, `sp` and `bp` the vault's supply and borrow exchange prices, `oracle` the debt value of one unit of collateral at a precision of `1e15`, and `liq_supply` and `liq_borrow` the vault's raw supply and borrow positions at liquidity.

The `fluid_timelock` package adds one governance goal. After setup, a gated admin call of the liquidity, lending or vaults layer runs only through an executed timelock operation. That operation runs at most once, with the params it was scheduled with, and only inside a time window. At launch, the admin timelock has a 12 h minimum delay, the auth timelock 6 h, and both a 7 day grace period.

Security relies on multiple mechanisms, such as:

1. **Possession-based identity.** Every `operate` acts on the `Holder` named by the presented `UserCap<T>`, whose ID must be in the registry's active list. There is no holder parameter and no sender check, so isolation between integrators rests entirely on the user capability.
2. **Exact integer accounting.** Positions and totals are stored as exact `u64` values, and every write path moves the acting position and its bucket total by the same amount.
3. **Rounding in the protocol's favor.** Deposit and payback floor, withdraw and borrow ceil, the borrow price accrues by a ceiled quotient and the supply price by a floored one, and the fee on interest is never zero.
4. **Monotonic exchange prices.** Both prices start at `1e12` and only ever increase.
5. **Role tiers and pause switches.** Governance, auth and guardian are separated in `Auth`, every gate runs the package-version check first, and pausing is available per action at the global, token and position scopes.
6. **Position ownership.** Each `Position` is bound to one `PositionOwnerCap`. A withdrawal or a borrow through `operate` needs the cap whose ID matches the position. A deposit or a payback is open to anyone.
7. **Tick and branch accounting.** `operate` places a position's debt on the tick of its debt-to-collateral ratio, raised by 0.15% (`compute_placement`), and each tick stores the debt of its positions. A liquidation moves tick debt into a branch, whose debt factor records the fraction of the debt that remains.
8. **Collateral-factor health check.** A withdrawal or a borrow that can raise the position's ratio reads the operate rate of the oracle quote, and `check_if_position_safe` aborts when the position ends above the collateral-factor tick.
9. **Bounded liquidation price.** `liquidate` derives the collateral per unit of debt from the oracle liquidate rate raised by the liquidation penalty, and aborts when it pays less than the caller's `col_per_unit_debt`.
10. **One holder at liquidity, reconciled by a rebalancer.** Both `UserCap` objects of a vault name the vault's own holder. The rebalancer settles the difference between the vault's books and its positions at liquidity through `rebalance`.
11. **Liquidation rounding tolerance.** `get_actual_amounts` accepts debt at liquidity up to `max_liquidation_rounding_diff` (100 raw units) above the amount the books release, clamps it, and aborts beyond it.

12. **Cap custody in the CapStore.** The shared `CapStore` holds the layer caps and the package `UpgradeCap`. A governance wrapper borrows a held cap to mint a ticket, and returns no ticket when the `CapStore` holds no cap for the layer.

13. **One timelock per class.** `cap_store::init` creates an admin timelock and an auth timelock and pins both ids in the `CapStore`. Each class helper accepts only the pinned timelock of its class, at the current `CapStore` version.

Roles & Trust Boundaries

Role	Trust Level	Powers
Governance	Highest	Address holding `RoleKey<AdminRole>` in `Auth`. Adds and removes admins, grants and revokes the auth, guardian and migrator tiers, mints and revokes user capabilities, sets the revenue collector, unpauses every scope, and mints or disables the liquidity `TimelockCap`. While that cap is recorded, each of these calls except the two cap switches also needs a ticket of the cap. Runs `migrate` with no ticket.
Package publisher	Highest	Holds the `UpgradeCap` until `store_upgrade_cap` moves it into the `CapStore`. After that, a compatible upgrade needs an executed admin-class `AuthorizeUpgrade` operation, and the migrator tier of each package then calls its `migrate`.
Vault admin	Highest	Address holding `AdminRole` in `VaultAdmin`. Adds and removes vault admins, grants and revokes the vault auth and migrator tiers and the per-market `RebalancerRole`, creates the position display, and mints or disables the vaults `TimelockCap`. While that cap is recorded, each of these calls except the

		two cap switches also needs a ticket of the cap. Runs `migrate_vault_registry` with no ticket.
Lending admin	Highest	Address holding `AdminRole` in the lending `Auth`. Creates lending markets, adds and removes lending admins, grants and revokes `RebalancerRole` and `MigrateRole`, mints and revokes the rewards writer cap, and mints or disables the lending `TimelockCap`. While that cap is recorded, each of these calls except the two cap switches also needs a ticket of the cap.
Setup holder (publisher)	Highest, until setup completes	Holds the `SetupCap` and, from `init`, every timelock role. Grants and revokes timelock roles with no delay until `finalize_roles`, moves the layer caps into the `CapStore` through `enable_all_timelocks`, and burns the `SetupCap` through `complete_setup`.
TimelockCap holder	Highest	Holder of a layer's recorded `TimelockCap` outside the `CapStore`. Mints tickets for that layer with no delay. `enable_timelock` mints the cap and stores it in the `CapStore` in the same call.
Auth tier	High	`RoleKey<AuthorizeRole>` or governance. Lists tokens, sets rate curves, fee and `max_utilization`, configures every position and withdrawal limit, and calls `collect_revenue`. While the liquidity `TimelockCap` is recorded, each of these calls also needs a ticket of the cap.
Vault auth tier	High	`AuthorizeRole` or vault admin. Creates vaults, installs their user capabilities, and sets the collateral factor, liquidation threshold, liquidation max limit, liquidation penalty, borrow fee, withdraw gap, rate

		magnifiers and oracle. While the vaults `TimelockCap` is recorded, each of these calls also needs a ticket of the cap.
Admin proposer	High	`AdminProposerRole` in the timelock `Auth`. Schedules admin-class operations on the admin timelock: top-tier layer actions, timelock role grants and revokes, guardian removals and package upgrades. Stores the `UpgradeCap` in the `CapStore`.
Auth proposer	High	Schedules auth-class operations, the config-tier layer actions, on the auth timelock.
Executor	High	`ExecutorRole`. Executes a pending operation inside its window, with the stored params. Completes the liquidity half of a guardian removal.
Timelock admin	High	`TimelockAdminRole`. Schedules and executes a change of a timelock's minimum delay, within 6 h to 60 days, or of its grace period, within 1 to 60 days. Each change waits the timelock's current minimum delay.
Timelock migrator	High	`MigrateRole` in the timelock `Auth`. Calls `migrate_cap_store` after an upgrade, with no delay.
Guardian	Medium	`RoleKey<GuardianRole>` or governance. Pauses any of the four actions globally (`pause_global`), token-wide (`pause_token`) or per position (`pause_user`), with no ticket. Cannot unpause anything. `complete_remove_guardian` revokes the role after a timelock `RemoveGuardian`.

Admin canceller	Medium	`AdminCancellerRole`. Cancels any pending operation on the admin timelock.
Auth canceller	Medium	`AuthCancellerRole`. Cancels any pending operation on the auth timelock. Guardians are seated with this role too.
Timelock guardian	Medium	`GuardianRole` in the timelock Auth. Cancels pending admin-timelock operations. An executed `RemoveGuardian` removes it.
Rebalancer	Medium	`RebalancerRole` for one vault. Calls `rebalance` and `rebalance_with_amounts`, which supply, withdraw, borrow or pay back at liquidity to close the gap between the vault's books and its positions there.
Integrator (capability holder)	Untrusted	Possession of a `UserCap<T>` whose ID is in `active_caps`. Calls operate on its own `Holder` only, with any mix of supply and borrow legs and any transfer type.
Anyone	Untrusted	Calls the permissionless price update `admin::update_exchange_prices<T>`, gated by the package version only, and every getter.
Liquidator (anyone)	Untrusted	Calls `liquidate` with any debt amount and any `col_per_unit_debt`, and deposits or pays back on any position through `operate`.
Position owner	Untrusted	Holder of a `PositionOwnerCap`, which has `key, store`. Withdraws and borrows on its own position through `operate`.

Primary trust boundaries. (1) **Capability holder vs reserve** the primary modeled adversary. A compromised integrator can call `operate` with any amounts and any transfer type; the rules bound what that call can do to the vault, the totals, the prices and every other holder's position. (2) **Anyone vs reserve** the permissionless price update lets any address crystallize the accrual at will, so the accrual formula itself must never over-credit. (3) **Guardian vs protocol** an intentionally low-privilege key that can halt a token but never unpause it; its effects on the reserve state are bounded by the same rules as every other entry point. (4) **Position owner vs vault** the owner chooses any collateral and debt change on its own position in one `operate`. (5) **Liquidator vs vault** anyone can liquidate any vault with any debt amount and any slippage bound, and can deposit or pay back on any position. (6) **Rebalancer vs vault** a medium-privilege key that moves coins between the vault's books and its positions at liquidity. (7) **Timelock guardian vs governance** a guardian can cancel admin-timelock operations, and an executed `RemoveGuardian` removes it. (8) **Setup holder vs launch** until `finalize_roles`, the `SetupCap` holder changes timelock roles with no delay. (9) **TimelockCap custody** a layer cap outside the `CapStore` skips the delay.

Expected Invariants

1. **Liquidity strong solvency (P-01).** The value of all supply positions never exceeds the vault plus the value of all borrow positions, `ts_if * 1e12 + ts_wi * sep <= tb_if * 1e12 + tb_wi * bep + vault * 1e12`, and every relevant state-changing entry point preserves it in every regime.
2. **Exchange prices never decrease (P-03).** Both stored exchange prices start at `1e12` and no entry point that takes `&mut TokenReserve<T>` lowers either one.
3. **Position and totals move together (P-02).** An `operate` leg changes the acting position and the matching bucket total by the same amount, leaves the other bucket untouched, and keeps the position within its bucket total. A configuration update that does not flip the position mode leaves the four totals, both prices and the vault unchanged.
4. **Vault fidelity (P-02).** `operate` moves exactly the requested withdraw or borrow amount out of the vault, and under `Skip` returns exactly that amount to the caller.
5. **No profitable round trip (P-02).** Two `operate` calls by one capability at the same clock, +x then -x or the reverse, leave the vault unchanged and never grow a supply position or shrink a debt.
6. **Holder isolation (P-04).** Every entry point that takes a mutable token reserve leaves all 19 stored fields of the supply and borrow rows of every holder it does not name unchanged.

7. **Exact admin census (P-04).** `add_admin` raises `admin_count` by exactly one, `remove_admin` lowers it by exactly one, at least one admin always remains, and only the role records of the named address change.
8. **Borrows stay within the collateral factor (P-08).** An `operate` that needs a quote leaves the position at or below the collateral-factor tick, and a position at or below that tick is solvent at the oracle rate.
9. **Vault solvency (P-08).** `total_supply * sp * oracle >= total_borrow * bp * 1e15`. Replacing a position by a solvent one, with the totals following the position, keeps a solvent vault solvent. A liquidation keeps the vault solvent when it was solvent with the liquidation penalty on its borrow.
10. **Tick debt follows the position (P-09).** `operate` moves a position's debt from its old tick or liquidated branch to its new tick, within the change of its net debt, and leaves every other tick and branch unchanged.
11. **Books backed at liquidity (P-08).** `operate` keeps `total_borrow * bp >= liq_borrow * 1e12` and `total_supply * sp <= liq_supply * 1e12` with the slack they had.
12. **Liquidation price bound (P-10).** A liquidation pays at least `col_per_unit_debt` collateral per unit of debt, and the liquidation loop pays at most the debt liquidated times the collateral-per-debt rate it is given.
13. **Liquidation conserves debt (P-10).** The liquidation loop removes from ticks and branches exactly the debt it liquidates, and outside a branch merge it never raises the branch debt factor.
14. **Timelock ticket gate (P-17).** While the liquidity, lending or vaults layer records a `TimelockCap`, each gated admin call returns only with a ticket of that cap, and the sender check of its tier still runs.
15. **Operation window (P-12).** `execute` runs only for a pending operation inside its window, under the action and params type it was scheduled with. It marks the operation done, and only its own timelock accepts the ticket.
16. **Settings stay in bounds (P-12).** A minimum delay update sets a value within 6 h to 60 days and a grace period update a value within 1 to 60 days, each only from a ready operation.
17. **Class delay floor (P-13).** Each class scheduler runs only on the pinned timelock of its class, and no operation it creates is ready before 12 h (admin class) or 6 h (auth class).

18. **Wrapper fidelity (P-16).** The executed reserve-targeted liquidity actions act only on the scheduled reserve, and the executed admin, role, guardian, revenue collector, cap minting, collateral factor and oracle actions apply exactly the scheduled values to the scheduled address or vault. Each runs on the timelock of its class.

Attack Vectors

1. An accrual step credits suppliers more than it charges borrowers

`reserve::calculate_exchange_prices` splits borrower interest between with-interest and interest-free suppliers and borrowers through two ratios quantized to `1e4`. A cap holder steers the reserve composition, and anyone crystallizes the drift through the permissionless price update, so a formula that over-credits suppliers drains the vault at the expense of interest-free suppliers and protocol revenue.

Key properties

- solvency `ts_if * 1e12 + ts_wi * sep <= tb_if * 1e12 + tb_wi * bep + vault * 1e12` is preserved by `update_exchange_prices` in all four regimes

2. Position and totals drift apart

If an `operate` leg or a mode conversion moved a position and its bucket total by different amounts, the totals would misprice every later accrual, and the `total_supply == 0` branch of `calc_revenue` could hand coins that back a live claim to the revenue collector.

Key properties

- a deposit, withdrawal, borrow or payback changes the acting position by at most the requested amount and its bucket total by exactly the same integer, and leaves the other bucket untouched
- the acting position stays at or below its bucket total
- `update_user_supply_config` and `update_user_borrow_configs` preserve solvency for any mode, and a converted amount stays within its new bucket total
- a configuration update that does not flip the mode leaves the four totals, both prices and the vault unchanged

3. Dust cycling and rounding on the four legs

A caller repeats small deposits, withdrawals, borrows and paybacks, to extract rounding residue from the vault or to shrink a debt for free.

Key properties

- deposit then withdraw of the same amount, and the reverse, at the same clock leave the vault unchanged and do not grow the supply position
- borrow then payback of the same amount, and the reverse, at the same clock leave the vault unchanged and do not shrink the debt
- a borrow or a withdrawal lowers the vault by exactly the requested amount; under `Skip` the returned balance equals that amount and under `Direct` nothing is returned
- each single leg of operate with any magnitude and either transfer type preserves solvency

4. An exchange price decreases

A decrease of the supply price shrinks every with-interest claim, a decrease of the borrow price forgives every with-interest debt, and every accounting proof starts from prices at least `1e12`.

Key properties

- every public function in user and admin that takes a mutable token reserve leaves both stored prices at or above their previous values

5. Revenue collection takes coins that back a supplier claim

`calc_revenue` computes `vault + total_borrow - total_supply`, or the whole vault when the ceiled total supply is zero, and `collect_revenue` withdraws that amount to the collector.

Key properties

- `collect_revenue` accrues interest, sets the stored prices to the reference model values, withdraws the revenue and preserves solvency, in all four regimes

6. A compromised capability holder reaches another holder's position

`UserCap<T>` has `key`, `store`, so possession is authorization, and `Holder` is plain data anyone constructs. A stolen or malicious capability must reach only the position it names.

Key properties

- `operate` leaves all 19 stored fields of the supply and borrow rows of any holder other than the capability's holder unchanged
- the position-level admin setters leave every row other than the named user unchanged
- the price update, `collect_revenue`, the rate and token setters and the token-level pauses leave every holder row unchanged

7. The admin census drifts or governance is emptied

`remove_admin` enforces the last-admin guard from `admin_count` alone. A counter that drifts from the number of `AdminRole` records lets governance be emptied, which freezes every unpause, revocation and migration.

Key properties

- `add_admin` sets the governance record of owner and raises `admin_count` by exactly one
- `remove_admin` clears it, lowers `admin_count` by exactly one and leaves at least one admin
- both leave the governance, auth and guardian records of every other address unchanged

8. A borrower takes debt beyond the collateral factor

A withdrawal or a borrow that leaves the position above the collateral-factor tick creates debt that its collateral does not cover at the oracle rate. Such a position can turn into bad debt before a liquidation reaches it.

Key properties

- with a collateral factor and a liquidation threshold of 100%, vault prices of one and an oracle rate of two, every `operate` that needs a quote leaves the position at or below tick 462, the tick of ratio two
- a position at or below tick 462 is solvent at those prices and that rate
- if the rest of the vault is solvent, the new position is solvent, and the totals move by no more than the position, then the vault stays solvent

9. Tick placement records debt that the position does not owe

`operate` places the debt at the perfect ratio times 1.0015 (`compute_placement`), and the tick and the vault totals record that placed debt. The position reads its debt back at the ratio of its

tick. When the two differ, the total borrow drifts from the sum of the positions, and a tick or a branch keeps debt that no position owes.

Key properties

- a deposit or a withdrawal moves the position's collateral and the total supply in the direction of the input, and the total supply never grows by more, or shrinks by less, than the position's collateral
- a borrow grows the position's net debt
- the old tick, or the liquidated branch of a reconstructed position, loses at least the position's old net debt, the new tick gains at most its new net debt, the net change is within the change of the position's net debt, and no other tick or branch changes
- on a borrow the total borrow grows by more than the position's net debt, and on a payback it shrinks by less. The placed debt exceeds the debt at the tick's ratio by about 2e-13 of the debt, in the user's favor

10. The vault's books lose their backing at liquidity

The vault records its users' collateral and debt in its own totals, and moves the tokens through its single position at liquidity. An `operate` that recorded more collateral than it supplied, or less debt than it borrowed, would leave the books promising coins that the liquidity layer does not hold for the vault.

Key properties

- for a deposit or a withdrawal combined with a borrow or a payback, the vault totals and the vault's positions at liquidity move in the direction of the input, and the vault's exchange prices do not change
- the recorded debt changes by at least, and the recorded collateral by at most, the amount that moves at liquidity
- `total_borrow * bp + slack * 1e12 >= liq_borrow * 1e12` and `total_supply * sp <= (liq_supply + supply_slack) * 1e12` hold after the call with the slacks they had before

11. A liquidator takes more collateral than the price allows

Anyone calls `liquidate` with any debt amount and any slippage bound. If the collateral paid per unit of debt exceeded the oracle rate raised by the penalty, a liquidator would take collateral that backs the remaining positions and leave the vault insolvent.

Key properties

- a liquidation pays at least `col_per_unit_debt` collateral per unit of debt taken
- over the real `liquidation_loop`, with zero, one or two ticks liquidated whole, the collateral liquidated is at most the debt liquidated times the collateral-per-debt rate
- a vault that is solvent at the oracle rate with the liquidation penalty on its borrow before the call is solvent at the oracle rate after it

12. The liquidation loop creates or destroys debt

`liquidation_loop` walks down from the top tick, liquidates ticks into the current branch and scales the branch debt factor. A step that removed more or less debt than it liquidated would leave positions owing debt that no tick holds, or erase debt that positions owe. A debt factor that grew would raise the debt of every position in the branch.

Key properties

- the debt of the top tick, of the ticks the walk passes and of both branches, plus the debt liquidated, equals their debt before, with zero, one or two ticks liquidated whole
- when the loop does not merge the branch into its base, the branch debt factor after the loop is at most its old value
- with no tick liquidated whole, the branch debt factor equals its old value times the fraction of the branch debt that was not liquidated
- the loop can take one or two ticks below the top tick that are not liquidated and liquidate them, so the properties above cover runs that liquidate

13. Liquidation rounding leaves debt at liquidity that the books do not record

`liquidate_utils::get_actual_amounts` computes the debt to pay back at liquidity from the liquidated debt and the borrow exchange price. When that amount is above the debt the books release by at most `max_liquidation_rounding_diff` (100 raw units), it clamps the amount down to the books' value instead of aborting. The difference stays as debt at liquidity that the vault's books do not record.

Key properties

- a liquidation takes at most the debt asked, grows neither total, keeps the supply-side backing, and adds at most one tolerance to the borrow-side gap
- liquidations add tolerance at most linearly. Each call adds its own, so the unrecorded debt grows with the number of liquidations

14. A rebalance moves the gap the wrong way

The rebalancer settles the difference between the vault's books and its positions at liquidity, such as the borrow fee. A rebalance that overshot or flipped the gap would move coins from the liquidity layer to the rebalancer, or leave the vault's debt at liquidity above what its users owe.

Key properties

- after `rebalance`, `preview` reports no gap on either leg, the recorded debt covers the debt at liquidity
- `rebalance_with_amounts` leaves the vault's books unchanged, and moves the gap of each leg toward zero without changing its sign

15. Governance acts on a layer without waiting the delay

Before the timelock, a governance or auth key changed a layer at once. If a gated call accepted a missing ticket, or a ticket of another cap, a compromised key could still change the layer with no delay, and users could not exit first.

Key properties

- while a `TimelockCap` is recorded, each gated admin call of liquidity, lending and vaults returns only with a ticket of the recorded cap, and the sender check of its tier still runs
- `timelock_ticket` mints only for the recorded cap, and the ticket carries that cap's id
- `mint_timelock_cap` runs only while no cap is recorded and records exactly the returned cap. `disable_timelock` needs the recorded cap by value and clears the record

16. An operation runs early, late, twice or with other params

The delay protects users only if the operation that runs is the one they saw scheduled, at the earliest at its ready time. A replayed or retyped operation would let a proposer run a change that nobody reviewed.

Key properties

- `schedule` stores the operation under a new id built from its action, params, salt and timelock, with a window that opens after at least the minimum delay and lasts one grace period
- `execute` runs only for a pending operation inside its window, marks it done and removes its params

- an `execute` under another action or another params type never runs the scheduled operation
- `consume` accepts a ticket only from the timelock that minted it, and returns its id and params unchanged

17. The delay is shortened or the window is broken

A minimum delay of zero, or a grace period of zero, would turn the timelock into instant governance or block every execution.

Key properties

- a minimum delay update runs only for a ready operation and sets the stored value, within 6 h to 60 days, and leaves the grace period unchanged
- a grace period update runs only for a ready operation and sets the stored value, within 1 to 60 days, and leaves the minimum delay unchanged
- `timelock::new` returns only with a minimum delay of at most 60 days and a grace period above zero and at most 60 days
- `init` gives the admin timelock a 12 h and the auth timelock a 6 h minimum delay, both with a 7 day grace period

18. An executed operation changes another target or applies other values

Reviewers read the scheduled params. If execute applied the change to another reserve, vault or address, or with other values, the reviewed change would not be the change that runs.

Key properties

- the reserve-targeted liquidity actions execute only on the scheduled reserve and on the timelock of their class
- `MintUserCap`, `UpdateRole` and `InitVaultConfig` execute only under the type they were scheduled with
- the executed admin, role, guardian, revenue collector, cap minting, collateral factor and oracle actions apply exactly the scheduled values, to the scheduled address or vault only

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	-	-	-
Low	-	-	-
Informational	2	2	0
Total	2	2	0

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
		Likelihood		

Detailed Findings

ID	Title	Severity	Status
I-01	Tick placement records debt that no position owes	Informational	Acknowledged
I-02	Rebalance rounds the recorded supply down and leaves it under-backed at liquidity	Informational	Acknowledged

Informational Issues

I-01. Tick placement records debt that no position owes

Description: `operate` can leave debt in a tick and in total_borrow that no position owes. `compute_placement` computes the position's ratio from its net debt and its collateral. It takes the tick at or below that ratio from `tick_math::get_tick_at_ratio` and places the position one tick higher. It sets `ratio_new` to the perfect ratio of the lower tick times 1.0015, rounded down. The tick records `ratio_new * col >> 48` as the position's gross debt. The position stores the difference between that gross debt and its net debt as dust, and `total_borrow` grows by the net debt. `ratio_new` is not read from `tick_math::get_ratio_at_tick(tick)`, the ratio of the tick that the position is placed on, and it can be above that ratio. `position::get_position_info` reads the gross debt back at `get_ratio_at_tick(tick)` and subtracts the stored dust. When `ratio_new` is above that ratio, the read-back gross debt and the read-back net debt are below the placed amounts. When the position pays back or moves, `operate` removes only the read-back gross debt from the tick and only the read-back net debt from `total_borrow`. The rest stays in the tick and in `total_borrow`, and `admin::preview` counts it as recorded debt.

The gap changes with the position's ratio and tick. In sampled inputs at ticks 0 to 2, it was at most about 2e-13 of the debt, in the user's favor.

Recommendation: Compute the placed debt from `tick_math::get_ratio_at_tick(tick)`, the ratio of the tick that the position is placed on, or cap `ratio_new` at that ratio. Apply the same change to the Solana program.

Violated properties: P-08 and P-09.

Customer's response: Acknowledged.

I-02. Rebalance rounds the recorded supply down and leaves it under-backed at liquidity

Description: After a rebalance, the vault's supply at liquidity can be lower than the collateral its books owe depositors. The difference is less than one raw unit. `rebalance_with_amounts` calls `preview` to measure the gap on each leg. `preview` computes the recorded collateral with `mul_div_u128(total_supply, vault_supply_ex_price, EXCHANGE_PRICES_PRECISION)` and then `unscale_amounts`, and both steps round down. The rebalance then moves the supply at liquidity to this floored value. Either it deposits the difference between the floored books and the supply at liquidity, or it withdraws the surplus and sends it to the rebalancer. In both cases, the fraction of the books that the floor drops is not backed at liquidity.

The gap does not accumulate, because each rebalance computes it again from the current books. Depositors cannot withdraw the missing fraction, because each withdrawal rounds against the user. The debt leg uses the same floor, but there it keeps the debt at liquidity at or below the debt that positions owe, which favors the protocol.

Recommendation: In `preview`, round the recorded collateral up: compute `col_recorded` with `mul_div_ceil_u128` and `unscale_amounts_up`, and keep the floor on the debt leg. A rebalance then leaves the supply at liquidity at or above the books. The cost to the rebalancer is at most one raw unit per call.

Violated properties: P-08.

Customer's response: Acknowledged.

Formal Verification

Verification Methodology

We performed verification of the **Fluid** protocol using the Certora verification tool which is based on Satisfiability Modulo Theories (SMT). In short, the Certora verification tool works by compiling formal specifications written in the Certora Verification Language for Move (CVLM) and **Fluid**'s implementation source code written in Move.
More information about Certora's tooling can be found in the Certora Technology Whitepaper.

If a property is verified with this methodology it means the specification in CVLM holds for all possible inputs. However specifications must introduce assumptions to rule out situations which are impossible in realistic scenarios (e.g. to specify the valid range for an input parameter). Additionally, SMT-based verification is notoriously computationally difficult. As a result, we introduce overapproximations (replacing real computations with broader ranges of values) and underapproximations (replacing real computations with fewer values) to make verification feasible.

Rules: A rule is a verification task possibly containing assumptions, calls to the relevant functionality that is symbolically executed and assertions that are verified on any resulting states from the computation.

Inductive Invariants: Inductive invariants are proved by induction on the structure of a smart contract. We use constructors/initialization functionality as a base case, and consider all other (relevant) externally callable functions that can change the storage as step cases.
Specifically, to prove the base case, we show that a property holds in any resulting state after a symbolic call to the respective initialization function. For proving step cases, we generally assume a state where the invariant holds (induction hypothesis), symbolically execute the functionality under investigation, and prove that after this computation any resulting state satisfies the invariant. Each such case results in one rule.

Verification Notations

Formally Verified	The rule is verified for every state of the contract(s), under the assumptions of the scope/requirements in the rule.
Formally Verified After Fix	The rule was violated due to an issue in the code and was successfully verified after fixing the issue
Violated	A counterexample exists that violates one of the assertions of the rule.

General Assumptions and Simplifications

1. To reduce the computational complexity, `reserve::calc_borrow_rate` is replaced by a summary that returns an arbitrary borrow rate within the `u64` range.

Formal Verification Properties Overview

ID	Title	Impact	Status
P-01	Liquidity Solvency	A supplier could withdraw more value than the vault and the outstanding debt cover.	Verified
P-02	Liquidity Accounting	A leg that moves a different amount than it records lets value leak from the vault.	Verified
P-03	Liquidity Reserve Invariants	A decreasing exchange price shrinks every with-interest claim or forgives debt.	Verified
P-04	Liquidity Access Control	A call that edits a position it does not name lets one account act on another's funds.	Verified
P-05	Lending Accounting	A user could split or round-trip an operation so the rounding pays out more than one call would, extracting value from the remaining holders.	Verified
P-06	Lending Rebalance	A funded rebalance could leave the vault's claims above its liquidity position, so the gap that rewards open would stay unbacked.	Verified
P-07	Lending Solvency	A user operation could move more value out of the backing at liquidity than the shares it burns are worth, so the remaining suppliers could not all exit.	Verified
P-08	Vaults Solvency	A position could owe more than its collateral is worth at the oracle rate, or the vault's books could claim more at liquidity than its positions there hold, so a liquidation or a withdrawal could not be funded.	Violated

		Violated by I-01 and I-02.	
P-09	Vaults Accounting	A position, its tick and the vault totals could drift apart, so the vault would price or liquidate debt that no position owes, or miss debt that one does. Violated by I-01.	Violated
P-10	Vaults Liquidation	A liquidation could hand out more collateral than the price allows, lose or create debt while it walks the ticks, or leave the branches in a state that the next liquidation or position fetch misreads.	Verified
P-11	Vaults Library Integrity	The summaries rest on these library facts. A wrong ratio, inverse or rounding would make the rules of the other modules prove the wrong statement, and would misprice positions in production.	Verified
P-12	Timelock Operation Lifecycle	An operation could run before its delay, after its window, twice, or with other params than the ones scheduled, so users would lose the time the timelock promises them to react.	Verified
P-13	Timelock Access Control	An account without the right role could schedule, run or cancel operations, an action could run on the wrong timelock, or a sensitive change could wait less than its class requires.	Verified
P-14	Timelock Role Governance	A role could lose its last holder and stall governance, one address could hold both GuardianRole and AdminCancellerRole, or a guardian could block or survive its own removal.	Verified

P-15	Timelock Cap Custody	A layer could be governed around the timelock, through a cap that the CapStore does not hold, or the package could be upgraded to code that governance did not schedule.	Verified
P-16	Timelock Protocol Wrappers	An executed operation could change another reserve, vault or address than the scheduled one, or apply other values, so the reviewed change would not be the change that runs.	Verified
P-17	Timelock Protocol Gates	An admin could change a liquidity, lending or vaults setting without waiting for the timelock delay, by calling a gated entry point without a ticket of the recorded cap, or by replacing or clearing the recorded cap.	Verified

Detailed Properties

Liquidity Solvency

Module General Assumptions

- The zero-accrual rules assume the prices are already settled at the call's clock. The accrual is proved separately over update_exchange_prices and collect_revenue.
- We assume that last_utilization fits in a u16, 0 < fee <= 100%, and max_utilization fits in a u64.

Module Properties

LIQ-CONF-11

Status: Verified	A user configuration update that keeps the position's mode leaves the totals, the exchange prices and the vault balance unchanged.

Rule Name	Status	Description	Link to rule report
totals_unchanged_supply_config_no_flip	Verified	*When admin::update_user_supply_config keeps the position's mode, the totals, the exchange prices and the vault balance are unchanged.*	Report
totals_unchanged_borrow_config_no_flip	Verified	*When admin::update_user_borrow_configs keeps the position's mode, the totals, the exchange prices and the vault balance are unchanged.*	Report

LIQ-SOLV-01

Status: Verified	The value of all supply positions never exceeds the vault plus the value of all borrow positions: ts_if * 1e12 + ts_wi * sep <= vault * 1e12 + tb_if * 1e12 + tb_wi * bep.

Rule Name	Status	Description	Link to rule report
liquidity_solvency_supply_config_any	Verified	*admin::update_user_supply_config preserves strong solvency over the vault. Upon a conversion, the converted amount stays within its new bucket total.* *Assumption: Exchange prices are settled at the call's clock.* *Assumption: The holder's supply amount is at most its bucket total.*	Report
liquidity_solvency_borrow_config_any	Verified	*admin::update_user_borrow_configs preserves strong solvency over the vault. Upon a conversion, the converted amount stays within its new bucket total.* *Assumption: Exchange prices are settled at the call's clock.*	Report
liquidity_solvency_update_exchange_price_no_if	Verified	*Strong solvency over the vault is preserved, in the regime with no interest-free supply and no interest-free borrow.* *Assumption: The reserve has no interest-free supply and no interest-free borrow.* *Assumption: Before the call, last_utilization * supply_value <= borrow_value * 1e4.*	Report
liquidity_solvency_update_exchange_price_no_ts_if	Verified	*Strong solvency over the vault is preserved, in the regime with no interest-free supply and some interest-free borrow.* *Assumption: The reserve has no interest-free supply and some interest-free borrow.* *Assumption: Before the call, last_utilization * supply_value <= borrow_value * 1e4.*	Report
liquidity_solvency_update_exchange_price_no_tb_if	Verified	*Strong solvency over the vault is preserved, in the regime with some interest-free supply and no interest-free borrow.* *Assumption: The reserve has some interest-free supply and no interest-free borrow.* *Assumption: Before the call, last_utilization * supply_value <= borrow_value * 1e4.*	Report
liquidity_solvency_update_exc	Verified	*Strong solvency over the vault is preserved, in the regime with both interest-free supply and*	Report

hange_price_with_if		*interest-free borrow.* *Assumption: The reserve has both interest-free supply and interest-free borrow.* *Assumption: Before the call, last_utilization * supply_value <= borrow_value * 1e4.*	
liquidity_solvency_collect_revenue_no_if	Verified	*admin::collect_revenue with accrual preserves strong solvency over the vault, in the regime with no interest-free supply and no interest-free borrow.* *Assumption: The reserve has no interest-free supply and no interest-free borrow.* *Assumption: Before the call, last_utilization * supply_value <= borrow_value * 1e4.*	Report
liquidity_solvency_collect_revenue_no_ts_if	Verified	*admin::collect_revenue with accrual preserves strong solvency over the vault, in the regime with no interest-free supply and some interest-free borrow.* *Assumption: The reserve has no interest-free supply and some interest-free borrow.* *Assumption: Before the call, last_utilization * supply_value <= borrow_value * 1e4.*	Report
liquidity_solvency_collect_revenue_no_tb_if	Verified	*admin::collect_revenue with accrual preserves strong solvency over the vault, in the regime with some interest-free supply and no interest-free borrow.* *Assumption: The reserve has some interest-free supply and no interest-free borrow.* *Assumption: Before the call, last_utilization * supply_value <= borrow_value * 1e4.*	Report
liquidity_solvency_collect_revenue_with_if	Verified	*admin::collect_revenue with accrual preserves strong solvency over the vault, in the regime with both interest-free supply and interest-free borrow.* *Assumption: The reserve has both interest-free supply and interest-free borrow.* *Assumption: Before the call, last_utilization * supply_value <= borrow_value * 1e4.*	Report
liquidity_solvency_operate_deposit	Verified	*A deposit through user::operate preserves solvency. The holder's supply position stays within its bucket total.* *Assumption: Exchange prices are settled at the*	Report

		call's clock. *Assumption: The holder's supply amount is at most its bucket total.*	
liquidity_solvency_operate_withdraw	Verified	*A withdrawal through user::operate preserves solvency. The holder's supply position stays within its bucket total.* *Assumption: Exchange prices are settled at the call's clock.* *Assumption: The holder's supply amount is at most its bucket total.*	Report
liquidity_solvency_operate_borrow	Verified	*A borrow through user::operate preserves solvency.* *Assumption: Exchange prices are settled at the call's clock.* *Assumption: The holder's supply amount is at most its bucket total.*	Report
liquidity_solvency_operate_payback	Verified	*A payback through user::operate preserves solvency.* *Assumption: Exchange prices are settled at the call's clock.* *Assumption: The holder's supply amount is at most its bucket total.*	Report

Liquidity Accounting

Module General Assumptions

- We assume that last_utilization fits in a u16, 0 < fee <= 100%, and max_utilization fits in a u64.

Module Properties

LIQ-ACCT-02			
Status: Verified		An operate leg changes the position and its bucket total by the same amount, at most the requested one. The other bucket is untouched, and the position stays within its bucket total.	
Rule Name	Status	Description	Link to rule report
operate_integrity_deposit_or_withdraw	Verified	*A deposit or a withdrawal through user::operate changes the supply position and its bucket total by the same amount, at most the requested one. The other bucket is untouched, and the position stays within its bucket total.*	Report
operate_integrity_borrow_or_payback	Verified	*A borrow or a payback through user::operate changes the borrow position and its bucket total by the same amount, at most the requested one. The other bucket is untouched, and the position stays within its bucket total.*	Report

LIQ-ROUND-03			
Status: Verified		Two operate calls by one capability at one clock, +x then -x or the reverse, leave the vault unchanged. They never grow a supply position or shrink a debt.	

Rule Name	Status	Description	Link to rule report
round_trip_deposit_then_withdraw	Verified	*A deposit of x followed by a withdrawal of x at the same clock leaves the vault unchanged and does not increase the supply position.*	Report
round_trip_withdraw_then_deposit	Verified	*A withdrawal of x followed by a deposit of x at the same clock leaves the vault unchanged and does not increase the supply position.*	Report
round_trip_borrow_then_payback	Verified	*A borrow of x followed by a payback of x at the same clock leaves the vault unchanged and does not decrease the borrow position.*	Report
round_trip_payback_then_borrow	Verified	*A payback of x followed by a borrow of x at the same clock leaves the vault unchanged and does not decrease the borrow position.*	Report

LIQ-UTIL-01

Status: Verified	After operate and after update_exchange_prices, the stored utilization equals floor(borrow_value * 1e4 / supply_value) over the stored totals and prices (with x / 0 == 0).

Rule Name	Status	Description	Link to rule report
liquidity_utilization_update_exchange_price	Verified	*After admin::update_exchange_prices, the stored utilization equals floor(borrow_value * 1e4 / supply_value) over the stored totals and prices.*	Report
liquidity_utilization_operate_deposit	Verified	*After a deposit through user::operate, the stored utilization equals floor(borrow_value * 1e4 / supply_value) over the stored totals and prices.*	Report

liquidity_utilization_operate_withdraw	Verified	*After a withdrawal through user::operate, the stored utilization equals floor(borrow_value * 1e4 / supply_value) over the stored totals and prices.*	Report
liquidity_utilization_operate_borrow	Verified	*After a borrow through user::operate, the stored utilization equals floor(borrow_value * 1e4 / supply_value) over the stored totals and prices.*	Report
liquidity_utilization_operate_payback	Verified	*After a payback through user::operate, the stored utilization equals floor(borrow_value * 1e4 / supply_value) over the stored totals and prices.*	Report

LIQ-VAULT-01

Status: Verified	operate moves exactly the requested withdraw or borrow amount out of the vault. Under Skip the caller receives exactly that amount.

Rule Name	Status	Description	Link to rule report
operate_borrow_claim_integrity	Verified	*A borrow through user::operate lowers the vault by exactly the requested amount. Under Skip the caller receives that amount, under Direct nothing is returned.*	Report
operate_withdraw_claim_integrity	Verified	*A withdrawal through user::operate lowers the vault by exactly the requested amount. Under Skip the caller receives that amount, under Direct nothing is returned.*	Report

Liquidity Reserve Invariants

Module Properties

LIQ-PRICE-01			
Status: Verified		Both exchange prices never decrease.	
Rule Name	Status	Description	Link to rule report
exchange_prices_monotonic_operate	Verified	*user::operate never lowers either stored exchange price.*	Report
exchange_prices_monotonic_update_exchange_prices	Verified	*admin::update_exchange_prices never lowers either stored exchange price.*	Report
exchange_prices_monotonic_collect_revenue	Verified	*admin::collect_revenue never lowers either stored exchange price.*	Report
exchange_prices_monotonic_update_rate_data_v1	Verified	*admin::update_rate_data_v1 never lowers either stored exchange price.*	Report
exchange_prices_monotonic_update_rate_data_v2	Verified	*admin::update_rate_data_v2 never lowers either stored exchange price.*	Report
exchange_prices_monotonic_u	Verified	*admin::update_token_config never lowers either stored exchange price.*	Report

pdate_token_config			
exchange_prices_monotonic_update_user_supply_config	Verified	*admin::update_user_supply_config never lowers either stored exchange price.*	Report
exchange_prices_monotonic_update_user_borrow_configs	Verified	*admin::update_user_borrow_configs never lowers either stored exchange price.*	Report
exchange_prices_monotonic_update_user_withdrawal_limit	Verified	*admin::update_user_withdrawal_limit never lowers either stored exchange price.*	Report
exchange_prices_monotonic_pause_token	Verified	*admin::pause_token never lowers either stored exchange price.*	Report
exchange_prices_monotonic_unpause_token	Verified	*admin::unpause_token never lowers either stored exchange price.*	Report
exchange_prices_monotonic_pause_user	Verified	*admin::pause_user never lowers either stored exchange price.*	Report
exchange_prices_monotonic_unpause_user	Verified	*admin::unpause_user never lowers either stored exchange price.*	Report

Liquidity Access Control

Module Properties

LIQ-AUTH-01			
Status: Verified		Every entry point that takes &mut TokenReserve leaves the full supply and borrow rows of every other holder unchanged.	
Rule Name	Status	Description	Link to rule report
holder_frame_operate	Verified	*user::operate leaves the supply and borrow rows of every other holder unchanged.*	Report
holder_frame_update_user_supply_config	Verified	*admin::update_user_supply_config leaves the supply and borrow rows of every other holder unchanged.*	Report
holder_frame_update_user_borrow_configs	Verified	*admin::update_user_borrow_configs leaves the supply and borrow rows of every other holder unchanged.*	Report
holder_frame_update_user_withdrawal_limit	Verified	*admin::update_user_withdrawal_limit leaves the supply and borrow rows of every other holder unchanged.*	Report
holder_frame_pause_user	Verified	*admin::pause_user leaves the supply and borrow rows of every other holder unchanged.*	Report
holder_frame_unpause_user	Verified	*admin::unpause_user leaves the supply and borrow rows of every other holder unchanged.*	Report
holder_frame_update_exchange_prices	Verified	*admin::update_exchange_prices leaves the supply and borrow rows of every other holder unchanged.*	Report

holder_frame_collect_revenue	Verified	*admin::collect_revenue leaves the supply and borrow rows of every holder unchanged.*	Report
holder_frame_update_rate_data_v1	Verified	*admin::update_rate_data_v1 leaves the supply and borrow rows of every holder unchanged.*	Report
holder_frame_update_rate_data_v2	Verified	*admin::update_rate_data_v2 leaves the supply and borrow rows of every holder unchanged.*	Report
holder_frame_update_token_config	Verified	*admin::update_token_config leaves the supply and borrow rows of every holder unchanged.*	Report
holder_frame_pause_token	Verified	*admin::pause_token leaves the supply and borrow rows of every holder unchanged.*	Report
holder_frame_unpause_token	Verified	*admin::unpause_token leaves the supply and borrow rows of every holder unchanged.*	Report

LIQ-ROLE-05

Status: Verified	add_admin and remove_admin move admin_count by exactly one, at least one admin always remains, and only the role records of the named address change.

Rule Name	Status	Description	Link to rule report
auth_frame_add_admin	Verified	*admin::add_admin sets the governance record of owner and raises admin_count by one. The records of every other address are unchanged.*	Report
auth_frame_remove_admin	Verified	*admin::remove_admin clears the governance record of owner and lowers admin_count by one. At*	Report

		least one admin remains, and the records of every other address are unchanged.	

Lending Accounting

Module Properties

LEND-ROUND-01	
Status: Verified	Splitting an operation never beats one call: shares(deposit(x1)) + shares(deposit(x2)) <= shares(deposit(x1 + x2)), payout(redeem(b1)) + payout(redeem(b2)) <= payout(redeem(b1 + b2)) and burned(withdraw(a1)) + burned(withdraw(a2)) >= burned(withdraw(a1 + a2)).

Rule Name	Status	Description	Link to rule report
lending_round_split_deposit	Verified	*Two user::deposit calls at one clock mint at most the shares of a single deposit of the summed amount. Proved through the stepping stones D1 to D3.* *Assumption: Both vault copies sit at the accrual boundary: the liquidity prices and the lending accrual are settled at the call's clock.* *Assumption: Both copies start at the same stored prices.*	Report
lending_round_split_redeem	Verified	*Two user::redeem calls at one clock pay out at most the payout of a single redeem of the summed shares. Proved through the stepping stones R1 to R3.* *Assumption: Both vault copies sit at the accrual boundary: the liquidity prices and the lending accrual are settled at the call's clock.* *Assumption: Both copies start at the same stored prices.*	Report
lending_round_split_withdraw	Verified	*Two user::withdraw calls at one clock burn at least the shares of a single withdrawal of the summed amount. Proved through the stepping stones W1 to W4.* *Assumption: Both vault copies sit at the accrual boundary: the liquidity prices and the lending accrual are settled at the call's clock.* *Assumption: Both copies start at the same stored prices and the same raw position.*	Report

		Assumption: The installed UserCap names the vault's holder on both copies.	

LEND-ROUND-02

Status: Verified	A round trip never returns more than it took in: redeem(deposit(x)) <= x, redeem(mint(n).shares) <= mint(n).required, preview_redeem(preview_deposit(x)) <= x and preview_deposit(preview_redeem(n)) <= n.

Rule Name	Status	Description	Link to rule report
lending_round_trip_deposit	Verified	*Redeeming exactly the shares a user::deposit minted never returns more than the deposit took in.*	Report
lending_round_trip_mint	Verified	*Redeeming exactly the shares a user::mint minted never returns more than the mint required.*	Report
lending_round_trip_preview_deposit	Verified	*Chaining accounting::preview_deposit and then the redeem preview never quotes more assets out than went in.*	Report
lending_round_trip_preview_redeem	Verified	*Chaining accounting::preview_redeem and then the deposit preview never quotes more shares out than went in. Proved through the stepping stones PR0 to PR3.*	Report

LEND-ROUND-03

Status: Verified	A withdrawal of at least one asset that does not abort burns at least one share.

Rule Name	Status	Description	Link to rule report
lending_round_withdraw_burns_share	Verified	*A user::withdraw call that returns leaves strictly fewer shares than before. Proved through the stepping stones B1 to B3.* *Assumption: The installed UserCap names the vault's holder.*	Report

Lending Rebalance

Module Properties

LEND-SOLV-10			
Status: Verified		A fully funded rebalance backs the vault's claims: shares * tep <= raw * sep + sep + P and shares * tep / P <= raw * sep / P + sep / P + 1 over the stored post-state.	
Rule Name	Status	Description	Link to rule report
lending_solvency_rebalance	Verified	*After admin::rebalance returns change, the stored claims shares * tep are at most the liquidity position raw * sep, up to sep + P.*	Report

Lending Solvency

Module Properties

LEND-SOLV	
Status: Verified	Every user entry point keeps the vault's claims backed by its liquidity position: liq_pre + lend_post <= liq_post + lend_pre + P + shares_pre * (tep_post - tep_pre) with lend = shares * tep, liq = raw * sep and P = 1e12, and the same inequality with zero slack over the floored backing liq = floor(raw * sep / P) * P; the token exchange price never falls and the supply exchange price is unchanged.

Rule Name	Status	Description	Link to rule report
lending_solvency_deposit	Verified	*A deposit through user::deposit preserves the solvency bound. The shares it mints are covered by the raw units liquidity books.* *Assumption: The liquidity exchange prices are settled at the call's clock.*	Report
lending_solvency_mint	Verified	*A mint through user::mint preserves the solvency bound. The shares it mints are covered by the raw units liquidity books.* *Assumption: The liquidity exchange prices are settled at the call's clock.*	Report
lending_solvency_withdraw	Verified	*A withdrawal through user::withdraw preserves the solvency bound. The backing removed at liquidity never exceeds the value of the shares it burns, up to one asset unit in scaled units, and exactly in floored units.* *Assumption: The liquidity exchange prices are settled at the call's clock.* *Assumption: The lending accrual is settled at the call's clock.* *Assumption: The installed UserCap names the vault's holder.*	Report

lending_solvency_redeem	Verified	*A redeem through user::redeem preserves the solvency bound. The payout never exceeds the value of the shares it burns.* *Assumption: The liquidity exchange prices are settled at the call's clock.*	Report
lending_solvency_update_rate	Verified	*user::update_rate accrues the token exchange price. The price never falls, and its growth is what the bound's growth term grants.*	Report

Vaults Solvency

Module General Assumptions

- The rules that call the vault assume both tokens have nine decimals, so raw amounts and token amounts share one scale.
- The rules of the mocked model summarise the tick math, the position pricing, the safety checks, the position fetch, the absorb sweep, the liquidation loop and the oracle-derived ticks by the bounds that the Accounting, Liquidation and Library Integrity modules prove on the real functions, and the vault's calls into liquidity at exchange prices of one. operate_position_safe_tick runs the real tick math, pricing and safety check.
- The liquidity solvency rules summarise the vault's calls into liquidity: a leg moves the vault's position at liquidity by the token amount at exchange prices of one, and settle_existing_position sets the settled position and the top tick to arbitrary values.
- The liquidate and liquidity solvency rules assume both reserves have stored exchange prices of one, settled at the call's clock, so a raw delta at liquidity equals the token amount.
- The liquidate rules fix the quote's liquidation rate, and operate_position_safe_tick both quote rates, at two debt units per collateral unit.
- The vault-totals rules run user::operate under the setup of the Accounting module's operate rules: the position starts at tick 1 with dust at most debt and a net debt above the dust threshold of 2,000 raw units, ends at a tick in [0, 2] with dust at most debt, and a branch above the vault's total branch id holds no debt.

Module Properties

VAULT-SOLV-01	
Status: Violated	user::operate keeps the vault solvent at the oracle rate. The proof has four steps: the vault totals follow the position, operate leaves the position at or below the collateral-factor tick, a position at or below that tick is solvent, and a vault of solvent positions is solvent. The first step is violated on the borrow side by finding I-01.

Rule Name	Status	Description	Link to rule report
integrity_vault_totals_operate_fetch	Violated	*After user::operate on a reconstructed position, the total borrow grows by at most the position's net debt change and the total supply grows by at least the position's collateral change.* *Assumption: The position's tick is liquidated, or was liquidated after the position was placed.* *Violated by I-01.*	Report
integrity_vault_totals_operate_latest	Violated	*After user::operate on a position whose tick is intact, the total borrow grows by at most the position's net debt change and the total supply grows by at least the position's collateral change.* *Assumption: The position's tick is not liquidated.* *Violated by I-01.*	Report
safe_tick_implies_position_solvency	Verified	*At exchange prices of one, an oracle rate of two and a collateral factor of 100 percent, a position at or below tick 462 is solvent.* *Assumption: The position's dust debt is at most its debt.* *Assumption: The position's tick is at most tick 462.*	Report
vault_solvency_lemma	Verified	*If the rest of the vault and the position are solvent, and the totals follow the position, the vault is solvent.*	Report
operate_position_safe_tick	Verified	*With the real safety check and pricing, user::operate leaves the position at or below the collateral-factor tick whenever the operation needs a price.* *Assumption: The collateral factor and the liquidation threshold are 100 percent, a configuration that risk_config::new rejects.* *Assumption: The stored vault exchange prices are one.*	Report

VAULT-SOLV-02

Status: Verified	user::liquidate keeps the vault solvent at the oracle rate, and hands out at least the collateral per unit of debt the caller asked for.

Rule Name	Status	Description	Link to rule report
integrity_slippage_liquidate	Verified	*After user::liquidate, the collateral handed out per unit of debt taken is at least the bound the caller asked for.* *Assumption: The call asks for a positive debt amount and a positive slippage bound.*	Report
vaults_solvency_liquidate	Verified	*After user::liquidate, the vault's supply valued at its price and the oracle rate still covers its borrow valued at its price.* *Assumption: The call asks for a positive debt amount and a positive slippage bound.*	Report

VAULT-SOLV-03

Status: Verified	user::operate keeps the vault aggregate covered by its positions at liquidity: the total supply at the vault's supply price is at most the supply position at liquidity, and the total borrow at the vault's borrow price is at least the borrow position at liquidity, each up to the same slack before and after. Each total and its position at liquidity move in the direction of the input, and both vault prices are kept.

Rule Name	Status	Description	Link to rule report
liquidity_solvency_operate_deposit_and_borrow	Verified	*After user::operate with a deposit and a borrow of positive amounts, the vault's supply and its supply at liquidity move in the direction of the input, so do its borrow and its borrow at liquidity, and the vault stays covered by its positions at liquidity with the same slacks.*	Report

liquidity_solvency_operate_deposit_and_payback	Verified	*After user::operate with a deposit and a payback of positive amounts, the vault's supply and its supply at liquidity move in the direction of the input, so do its borrow and its borrow at liquidity, and the vault stays covered by its positions at liquidity with the same slacks.*	Report
liquidity_solvency_operate_withdraw_and_borrow	Verified	*After user::operate with a withdraw and a borrow of positive amounts, the vault's supply and its supply at liquidity move in the direction of the input, so do its borrow and its borrow at liquidity, and the vault stays covered by its positions at liquidity with the same slacks.*	Report
liquidity_solvency_operate_withdraw_and_payback	Verified	*After user::operate with a withdraw and a payback of positive amounts, the vault's supply and its supply at liquidity move in the direction of the input, so do its borrow and its borrow at liquidity, and the vault stays covered by its positions at liquidity with the same slacks.*	Report

VAULT-SOLV-04

Status: Verified	user::liquidate keeps the vault aggregate covered by its positions at liquidity: it takes at most the debt asked, grows neither total, keeps the supply-side coverage exactly, and adds at most one rounding tolerance of 100 raw units of unrecorded debt at liquidity per call, so the gap grows at most linearly with the number of liquidations.

Rule Name	Status	Description	Link to rule report
liquidity_solvency_liquidate	Verified	*After user::liquidate, the debt taken is at most the amount asked, both totals do not grow, the supply-side slack is kept, and the borrow-side slack grows by at most the liquidation rounding tolerance.* *Assumption: The call asks for a positive debt amount and a positive slippage bound.*	Report
liquidity_gap_bounded	Verified	*Starting from books that exactly cover the debt at liquidity, after two calls to user::liquidate the debt at liquidity exceeds the books by at most two rounding tolerances, 200 raw units. The clamp in*	Report

		liquidate_utils::get_actual_amounts adds at most one tolerance per call, so the gap grows at most linearly with the number of liquidations. *Assumption: The books exactly cover the debt at liquidity before the first call.*	

VAULT-SOLV-05

Status: Violated	admin::rebalance closes the gap between the vault aggregate and its positions at liquidity on both legs and leaves the aggregate covered; admin::rebalance_with_amounts leaves the aggregate unchanged and moves each gap toward zero.

Rule Name	Status	Description	Link to rule report
liquidity_solvency_rebalance	Violated	*After admin::rebalance, admin::preview reports no gap on either leg, the vault's borrow is covered by its borrow at liquidity, and its supply is covered by its supply at liquidity.* *Violated by I-02.*	Report
liquidity_solvency_rebalance_with_amounts	Verified	*After admin::rebalance_with_amounts, the vault's books are unchanged, and on each leg the gap that admin::preview reports is zero or keeps its sign and does not grow.*	Report

Vaults Accounting

Module General Assumptions

- The operate rules run under the mocked model: both tokens have nine decimals, and the tick math, the position pricing, the safety checks, the position fetch, the tick-bitmap walks and the vault's calls into liquidity are summarised by the bounds that the other rules of this module and the Library Integrity module prove on the real functions.
- The operate rules assume that a branch above the vault's total branch id holds no debt. Branches open one above the total branch id, and set_top_tick resets a branch before it lowers the total below it, so every reachable vault satisfies it.
- The operate rules start the position at tick 1 with dust at most debt and a net debt above the dust threshold of 2,000 raw units, and end it at a tick in [0, 2] with dust at most debt.
- The fetch rules run the real fetch_latest_position, tick math and big-number arithmetic over a vault with two branches. The rules that reconstruct a position assume branch 2 is merged into the liquidated branch 1, the debt factors are normalised, branch 1's is at most one and the connection factor is at least it.

Module Properties

VAULT-ACCT-01

Status: Violated	user::operate keeps the position consistent with the inputs: the position's collateral and net debt move in the direction of the inputs, and each vault total moves by no more than the position does.

Rule Name	Status	Description	Link to rule report
integrity_position_operate_fetch	Violated	*After user::operate on a position whose tick was liquidated, the position's collateral and net debt move in the direction of the inputs, and the vault totals move by no more than the position does.* *Assumption: The position's tick is liquidated, or was liquidated after the position was placed.* *Violated by I-01.*	Report

integrity_position_operate_latest	Violated	*After user::operate on a position whose tick is intact, the position's collateral and net debt move in the direction of the inputs, and the vault totals move by no more than the position does.* *Assumption: The position's tick is not liquidated.* *Violated by I-01.*	Report

VAULT-ACCT-02

Status: Verified	user::operate keeps the ticks and branches consistent with the position: the position's debt moves from its source, the old tick or the liquidated branch, to its new tick, the source does not gain, the new tick does not lose, the net change of tick debt is within the change of the position's net debt, and no other tick or branch changes.

Rule Name	Status	Description	Link to rule report
integrity_tick_operate_fetch	Verified	*After user::operate on a reconstructed position, the liquidated branch's debt does not grow, the new tick's debt does not shrink, and the net change is within the change of the position's net debt. No other tick's or branch's debt changes.* *Assumption: The position's tick is liquidated, or was liquidated after the position was placed.* *Assumption: The position ends with no dust.* *Assumption: The final tick has no debt when it has no ids or is liquidated.* *Assumption: The branch's debt covers the position's debt plus the minimum branch debt.*	Report
integrity_tick_operate_latest	Verified	*After user::operate on a position whose tick is intact, the position's debt moves from its old tick to its new tick: the old tick loses at least the position's old net debt, the new tick gains at most its new net debt, and no other tick changes. When the position stays on its tick, that tick's debt rises by at most what the position borrows, falls by at least what it repays, and never rises on a repayment.* *Assumption: The position's tick is not liquidated.* *Assumption: The position ends with no dust.* *Assumption: The position's tick holds at least the*	Report

		position's net debt. *Assumption: The final tick has no debt when it has no ids or is liquidated.* *Assumption: The branch's debt covers the position's debt plus the minimum branch debt.*	

VAULT-ACCT-03

Status: Verified	operate_utils::fetch_latest_position brings a position whose tick was liquidated up to date: a fully liquidated position is empty, a position has collateral exactly when it has debt, its debt is strictly below the stored debt, and its collateral sits at its tick's ratio adjusted by the branch's partials.

Rule Name	Status	Description	Link to rule report
integrity_fully_liquidated_fetch	Verified	*When the position's tick entry is fully liquidated, operate_utils::fetch_latest_position leaves no debt, no collateral and the cold tick.*	Report
integrity_debt_zero_implies_col_zero_fetch	Verified	*After operate_utils::fetch_latest_position, a position without debt has no collateral and the cold tick, and a position with debt has a live tick and collateral, unless its debt is below one collateral unit at that tick.*	Report
integrity_debt_is_reduced_fetch	Verified	*After operate_utils::fetch_latest_position, a position that still has debt has strictly less than before.*	Report
integrity_relates_debt_and_col_fetch	Verified	*After operate_utils::fetch_latest_position, a position with debt has collateral equal to its debt at the ratio of its tick, adjusted by the branch's partials.*	Report

VAULT-ACCT-04

Status: Verified	The summaries of position::get_new_position_info and liquidate_utils::get_ticks_from_oracle_price hold on the real functions: the pricing rounds against the user, each leg within one unit at the exchange price, and the collateral-per-debt rate is at most the inverse of the oracle rate adjusted by the two exchange prices and raised by the liquidation penalty.

Rule Name	Status	Description	Link to rule report
verify_get_new_position_info_lemma	Verified	*position::get_new_position_info rounds the raw collateral down on a deposit and up on a withdrawal, and the raw debt up on a borrow and down on a payback, each within one unit of the amount at the exchange price. A max withdrawal empties the collateral and withdraws at most its value; a max payback empties the debt and pays at least its value.* *Assumption: Both exchange prices are at least one and fit in 64 bits.*	Report
verify_summary_get_ticks_from_oracle_price	Verified	*liquidate_utils::get_ticks_from_oracle_price returns a collateral-per-debt rate of at most the inverse of the oracle rate adjusted by the two exchange prices and raised by the liquidation penalty.* *Assumption: The quote's liquidation rate is two debt units per collateral unit.* *Assumption: Both exchange prices are at least one.*	Report

Vaults Liquidation

Module General Assumptions

- The liquidation walk is summarised: tick::fetch_next_tick_liquidate returns at most two ghost ticks, each strictly below the current tick, then a tick at or below the liquidation tick, and the bitmap is not maintained. The bitmap rules prove that the real walk descends.
- The debt liquidated at a tick, liquidate_utils::get_debt_liquidated, is an arbitrary amount at most the tick's debt.
- The loop and debt-factor rules summarise the tick math as the mocked model does; the absorb and bitmap rules run the real tick math.
- The loop rules summarise the big-number arithmetic: mul_div_big_number returns zero or a normalised factor at most its input, which the Library Integrity module proves, and div_big_number is arbitrary. The debt-factor rule runs the real arithmetic.
- The absorb walk, tick::fetch_next_tick_absorb, finds one fixed tick with fixed collateral and debt, and spends an arbitrary part of its budget.
- The loop and debt-factor rules run over a vault with two branches: branch 2 is current and connects to the liquidated branch 1, branch 1's connected minima is below the liquidation tick, and the two ticks the walk returns are not liquidated before the call. When the branch is liquidated, branch 2's minima is the top tick and the top tick holds no debt. Otherwise branch 2 is fresh.
- The absorb rules run over a vault with two branches: branch 2 is current with total id 2 and connects to the liquidated branch 1, branch 2 is fresh unless the branch is liquidated, and the max tick is strictly inside the tick range.

Module Properties

VAULT-LIQ-01	
Status: Verified	On runs through zero, one or two perfect ticks, liquidate_logic::liquidation_loop liquidates collateral worth at most the debt liquidated at the collateral-per-debt rate, conserves debt across the top tick, the ticks it walks and both branches, and never raises the current branch's debt factor; with no perfect tick it scales that factor by exactly the fraction of the branch's debt that remains.

Rule Name	Status	Description	Link to rule report
integrity_debt_factor_execute_liquidation	Verified	*After liquidate_logic::liquidation_loop on a liquidated branch, on a run through no perfect tick, branch 2's debt factor equals its old factor times the fraction of its debt that was not liquidated.* *Assumption: The branch is liquidated, branch 1's minima is below the liquidation tick, and branch 2's debt factor is not zero.* *Assumption: The run passes no perfect tick.*	Report
verify_summary_execute_liquidation_no_perfect_step	Verified	*After liquidate_logic::liquidation_loop on a run through no perfect tick, the collateral liquidated is at most the debt liquidated at the collateral-per-debt rate.*	Report
verify_summary_execute_liquidation_one_perfect_step	Verified	*After liquidate_logic::liquidation_loop on a run through one perfect tick, the collateral liquidated is at most the debt liquidated at the collateral-per-debt rate.*	Report
verify_summary_execute_liquidation_two_perfect_steps	Verified	*After liquidate_logic::liquidation_loop on a run through two perfect ticks, the collateral liquidated is at most the debt liquidated at the collateral-per-debt rate.*	Report
integrity_debt_execute_liquidation_no_perfect_step	Verified	*After liquidate_logic::liquidation_loop on a run through no perfect tick, the debt of the top tick, of the ticks the walk returned and of both branches, plus the debt liquidated, equals their debt before.*	Report
integrity_debt_execute_liquidation_one_perfect_step	Verified	*After liquidate_logic::liquidation_loop on a run through one perfect tick, the debt of the top tick, of the ticks the walk returned and of both branches, plus the debt liquidated, equals their debt before.*	Report
integrity_debt_execute_liquidat	Verified	*After liquidate_logic::liquidation_loop on a run through two perfect ticks, the debt of the top tick,*	Report

ion_two_perfect_steps		*of the ticks the walk returned and of both branches, plus the debt liquidated, equals their debt before.*	
liquidation_loop_reaches_perfect_tick	Verified	*liquidate_logic::liquidation_loop can take a tick below the top tick as a perfect reference tick and liquidate it.*	Report
liquidation_loop_reaches_two_perfect_ticks	Verified	*liquidate_logic::liquidation_loop can take both ticks the walk returns as perfect reference ticks and liquidate them.*	Report
integrity_debt_factor_never_grows_one_perfect_step	Verified	*After liquidate_logic::liquidation_loop on a liquidated branch, on a run through one perfect tick that does not merge branch 2 into its base, branch 2's debt factor is at most its old factor. A merge stores the connection factor in that field instead.* *Assumption: The branch is liquidated and branch 2's debt factor is normalised.*	Report
integrity_debt_factor_never_grows_two_perfect_steps	Verified	*After liquidate_logic::liquidation_loop on a liquidated branch, on a run through two perfect ticks that does not merge branch 2 into its base, branch 2's debt factor is at most its old factor. A merge stores the connection factor in that field instead.* *Assumption: The branch is liquidated and branch 2's debt factor is normalised.*	Report

VAULT-LIQ-02	
Status: Verified	liquidate_logic::absorb keeps the vault totals, the vault prices and the branch minima, and sets the liquidated flag, the top tick, the branch ids, the branches and the absorbed amounts as each of the six cases of the branch layout describes.

Rule Name	Status	Description	Link to rule report
integrity_basic_absorb	Verified	*liquidate_logic::absorb leaves the vault totals, the vault prices and both branches' minima unchanged, and leaves the top tick at most the max tick unless the budget runs out.* *Assumption: Branch 1's minima is at most the max tick.*	Report
integrity_vault_state_absorb_case1	Verified	*When the branch is not liquidated and branch 1 is fresh, liquidate_logic::absorb leaves the vault not liquidated, moves the top tick to the tick the walk found, and keeps the current and the total branch id.* *Assumption: The branch is not liquidated and branch 1 is fresh.*	Report
integrity_branch_absorb_case1	Verified	*When the branch is not liquidated and branch 1 is fresh, liquidate_logic::absorb leaves branch 2 open with a cold minima.* *Assumption: The branch is not liquidated and branch 1 is fresh.*	Report
integrity_col_debt_absorb_case1	Verified	*When the branch is not liquidated and branch 1 is fresh, liquidate_logic::absorb returns the collateral and debt of the tick the walk found.* *Assumption: The branch is not liquidated and branch 1 is fresh.*	Report
integrity_vault_state_absorb_case2a	Verified	*When the branch is liquidated and branch 2's minima is at most the max tick and the next tick, liquidate_logic::absorb opens a new branch 3 as the current branch, raises the total branch id by one, clears the liquidated flag and moves the top tick to the tick the walk found.* *Assumption: The branch is liquidated and branch 2's minima is at most the max tick and the next tick.*	Report
integrity_branch_absorb_case2a	Verified	*When the branch is liquidated and branch 2's minima is at most the max tick and the next tick, liquidate_logic::absorb connects the new branch 3 to branch 2 at branch 2's minima, or opens it fresh when that minima is cold.*	Report

		Assumption: The branch is liquidated and branch 2's minima is at most the max tick and the next tick.	
integrity_col_debt_absorb_case2a	Verified	*When the branch is liquidated and branch 2's minima is at most the max tick and the next tick, liquidate_logic::absorb returns the collateral and debt of the tick the walk found.* *Assumption: The branch is liquidated and branch 2's minima is at most the max tick and the next tick.*	Report
integrity_vault_state_absorb_case2b	Verified	*When the branch is liquidated and branch 2's minima is at most the max tick and above the next tick, liquidate_logic::absorb leaves the vault liquidated with branch 2 current, the top tick at branch 2's minima and the total branch id unchanged.* *Assumption: The branch is liquidated and branch 2's minima is at most the max tick and above the next tick.*	Report
integrity_branch_absorb_case2b	Verified	*When the branch is liquidated and branch 2's minima is at most the max tick and above the next tick, liquidate_logic::absorb leaves branch 2 open.* *Assumption: The branch is liquidated and branch 2's minima is at most the max tick and above the next tick.*	Report
integrity_col_debt_absorb_case2b	Verified	*When the branch is liquidated and branch 2's minima is at most the max tick and above the next tick, liquidate_logic::absorb returns the collateral and debt of the tick the walk found.* *Assumption: The branch is liquidated and branch 2's minima is at most the max tick and above the next tick.*	Report
integrity_vault_state_absorb_case2c	Verified	*When the branch is liquidated, branch 2's minima is above the max tick and branch 1 is fresh, liquidate_logic::absorb closes branch 2 and opens a new branch 3 as the current branch, raises the total branch id by one, clears the liquidated flag and moves the top tick to the tick the walk found. When its budget runs out first, the top tick stays above the max tick.* *Assumption: The branch is liquidated, branch 2's minima is above the max tick and branch 1 is fresh.*	Report

integrity_branch_absorb_case2c	Verified	*When the branch is liquidated, branch 2's minima is above the max tick and branch 1 is fresh, liquidate_logic::absorb closes branch 2 and opens a fresh branch 3. When its budget runs out first, branch 2 stays open.* *Assumption: The branch is liquidated, branch 2's minima is above the max tick and branch 1 is fresh.*	Report
integrity_col_debt_absorb_case2c	Verified	*When the branch is liquidated, branch 2's minima is above the max tick and branch 1 is fresh, liquidate_logic::absorb returns branch 2's collateral and debt plus those of the tick the walk found. When its budget runs out first, it returns the tick's only.* *Assumption: The branch is liquidated, branch 2's minima is above the max tick and branch 1 is fresh.*	Report
integrity_vault_state_absorb_case2d	Verified	*When the branch is liquidated, branch 2's minima is above the max tick and branch 1's minima is at most the max tick and the next tick, liquidate_logic::absorb closes branch 2 and opens a new branch 3 as the current branch, raises the total branch id by one, clears the liquidated flag and moves the top tick to the tick the walk found. When its budget runs out first, the top tick stays above the max tick.* *Assumption: The branch is liquidated, branch 2's minima is above the max tick and branch 1's minima is at most the max tick and the next tick.*	Report
integrity_branch_absorb_case2d	Verified	*When the branch is liquidated, branch 2's minima is above the max tick and branch 1's minima is at most the max tick and the next tick, liquidate_logic::absorb closes branch 2, leaves branch 1 open and connects the new branch 3 to branch 1 at branch 1's minima. When its budget runs out first, branch 2 stays open.* *Assumption: The branch is liquidated, branch 2's minima is above the max tick and branch 1's minima is at most the max tick and the next tick.*	Report
integrity_col_debt_absorb_case2d	Verified	*When the branch is liquidated, branch 2's minima is above the max tick and branch 1's minima is at most the max tick and the next tick, liquidate_logic::absorb returns branch 2's collateral and debt plus those of the tick the walk found. When its budget runs out first, it returns the tick's only.*	Report

		Assumption: The branch is liquidated, branch 2's minima is above the max tick and branch 1's minima is at most the max tick and the next tick.	
integrity_vault_state_absorb_case2e	Verified	*When the branch is liquidated, branch 2's minima is above the max tick and branch 1's minima is at most the max tick and above the next tick, liquidate_logic::absorb leaves the vault liquidated with branch 1 current, the top tick at branch 1's minima and the total branch id unchanged. When its budget runs out first, the top tick stays above the max tick.* *Assumption: The branch is liquidated, branch 2's minima is above the max tick and branch 1's minima is at most the max tick and above the next tick.*	Report
integrity_branch_absorb_case2e	Verified	*When the branch is liquidated, branch 2's minima is above the max tick and branch 1's minima is at most the max tick and above the next tick, liquidate_logic::absorb closes branch 2 and leaves branch 1 open. When its budget runs out first, branch 2 stays open.* *Assumption: The branch is liquidated, branch 2's minima is above the max tick and branch 1's minima is at most the max tick and above the next tick.*	Report
integrity_col_debt_absorb_case2e	Verified	*When the branch is liquidated, branch 2's minima is above the max tick and branch 1's minima is at most the max tick and above the next tick, liquidate_logic::absorb returns branch 2's collateral and debt plus those of the tick the walk found. When its budget runs out first, it returns the tick's only.* *Assumption: The branch is liquidated, branch 2's minima is above the max tick and branch 1's minima is at most the max tick and above the next tick.*	Report

VAULT-LIQ-03	
Status: Verified	tick::fetch_next_tick_liquidate returns a tick strictly below the current tick.

Rule Name	Status	Description	Link to rule report
fetch_next_tick_liquidate_stays_below	Verified	*When the scan of tick::fetch_next_tick_liquidate stops in the current tick's word, the tick it returns is strictly below the current tick, provided that, without clear, no bit at or above the current tick is set in its word.* *Assumption: The current tick is canonical: the library builds no negative zero.* *Assumption: Without clear, no bit at or above the current tick is set in its word.* *Assumption: The current tick's word, after the clear, has a set bit.*	Report
tick_at_lower_word_is_lower	Verified	*Every tick that tick::tick_at addresses in a word below the word of a tick is strictly below that tick, so a scan that moves to a lower word returns a lower tick.*	Report

Vaults Library Integrity

Module General Assumptions

- The multiply-divide rules take operands that fit in 64 bits.

Module Properties

VAULT-LIB-01	
Status: Verified	fluid_library::tick_math satisfies the tick-math summary of the mocked model: get_ratio_at_tick returns the exact constants at ticks -1, 0, 1, 2, 3, 462 and 463 and the range limits at the extreme ticks and lies in the summary's band elsewhere; get_tick_at_ratio returns the band's tick for every ratio in range, with a perfect ratio at most the input and equal to the closed form in the bands of ticks -1 to 2.

Rule Name	Status	Description	Link to rule report
tick_math_exact_ratios	Verified	*tick_math::get_ratio_at_tick returns the summary's constants at ticks -1, 0, 1, 2, 3, 462 and 463, and the band limits at the minimum and maximum ticks.*	Report
tick_math_ratio_bands	Verified	*For every tick in bounds, tick_math::get_ratio_at_tick lies in the band the summary assumes for that tick.*	Report
tick_math_inverse_bands	Verified	*For every ratio in range, the tick that tick_math::get_tick_at_ratio returns lies in the band the summary assumes for that ratio.*	Report
tick_math_inverse_ratio	Verified	*For every ratio in range, the perfect ratio that tick_math::get_tick_at_ratio returns is at most the input.*	Report
tick_math_inverse_perfect_ratio	Verified	*For every ratio in the bands of ticks -1 to 2, the perfect ratio that tick_math::get_tick_at_ratio returns is the summary's closed form for that band: the input divided by the factor left after the tick's threshold divisions.*	Report

VAULT-LIB-02

Status: Verified	math::mul_div_u128 and mul_div_ceil_u128 return the exact floor and ceiling of the product of the first two operands over the third.

Rule Name	Status	Description	Link to rule report
safe_multiply_divide_integrity	Verified	*math::mul_div_u128 returns the exact floor of the product of the first two operands over the third.*	Report
safe_multiply_divide_up_integrity	Verified	*math::mul_div_ceil_u128 returns the exact ceiling of the product of the first two operands over the third.*	Report

VAULT-LIB-03

Status: Verified	big_number::mul_div_big_number of a normalised factor and a multiplier at most 2^64 is zero or a normalised factor at most the input, as the big-number summary of the liquidation-loop rules assumes.

Rule Name	Status	Description	Link to rule report
mul_div_big_number_does_not_grow	Verified	*big_number::mul_div_big_number of a normalised factor and a multiplier of at most 2^64 is zero, or a normalised factor at most the input.* *Assumption: The factor is normalised and the multiplier is at most 2^64.*	Report

Timelock Operation Lifecycle

Module Properties

TL-RULE-C1

Status: Verified	schedule stores the operation under a new id built from its action, params, salt and timelock, with the scheduled params and a window that opens after at least the minimum delay and lasts one grace period.

Rule Name	Status	Description	Link to rule report
schedule_locks_window	Verified	*After timelock::schedule_internal, the operation has a new id built from its action, params, salt and timelock, a delay of at least the minimum delay, a window of one grace period, and the scheduled params. The settings do not change.*	Report

TL-RULE-C2

Status: Verified	Schedule, execute, cancel and the updates of the minimum delay and the grace period never move a pending operation or reopen a done one, cancel removes the named pending operation and its params, and these functions keep params stored exactly for the pending operations.

Rule Name	Status	Description	Link to rule report
pending_frame	Verified	*Schedule, execute, cancel and the updates of the minimum delay and the grace period never move a pending operation: a done operation stays done, and an operation that stays pending keeps its ready time, expiry and action.*	Report
params_iff_pending	Verified	*Schedule, execute, cancel and the updates of the minimum delay and the grace period keep params stored exactly for the pending operations.*	Report

cancel_removes_pending	Verified	*timelock::cancel_internal runs only for a pending operation with the given params type, and removes that operation and its params. The settings do not change.*	Report

TL-RULE-C3

Status: Verified	execute runs only for a pending operation inside its window and only under the action and params type it was scheduled with, marks the operation done, and hands out a ticket with the stored params that only its own timelock accepts.

Rule Name	Status	Description	Link to rule report
execute_only_in_window	Verified	*timelock::execute_internal runs only for a pending operation inside its window.*	Report
execute_refuses_other_action	Verified	*An execute under another action never runs an operation that was scheduled under the first action.*	Report
execute_refuses_other_params	Verified	*An execute with another params type never runs an operation that was scheduled with the first params type.*	Report
execute_same_action_reachable	Verified	*An operation can be executed later under the action and params type it was scheduled with.*	Report
execute_consumes_slot	Verified	*After timelock::execute_internal, the operation is done, its params are removed, and the ticket carries this timelock, the operation id and the stored params. The settings do not change.*	Report
consume_binds_timelock	Verified	*timelock::consume accepts a ticket only from the timelock that minted it, and returns its id and params unchanged.*	Report

TL-RULE-C7

Status: Verified	timelock::new sets the minimum delay and the grace period within their bounds, each setting update runs only for a ready operation and sets its stored value within its bounds without changing the other setting, and schedule, execute and cancel leave both settings unchanged.

Rule Name	Status	Description	Link to rule report
update_min_delay_applies_stored_value	Verified	*timelock::execute_update_min_delay runs only for a ready operation and sets the minimum delay to the stored value, between 6 h and 60 days. The grace period does not change.*	Report
update_grace_period_applies_stored_value	Verified	*timelock::execute_update_grace_period runs only for a ready operation and sets the grace period to the stored value, between 1 day and 60 days. The minimum delay does not change.*	Report
new_bounds_settings	Verified	*timelock::new returns only with a minimum delay of at most 60 days and a grace period above zero and at most 60 days.*	Report

Timelock Access Control

Module Properties

TL-ACC-13

Status: Verified	The admin and auth class helpers schedule and execute only on the timelock of their class and only at the current CapStore version, and each class scheduler runs only on the timelock of its class.

Rule Name	Status	Description	Link to rule report
admin_scheduler_class	Verified	*Each admin-class scheduler runs only on the admin timelock.*	Report
auth_scheduler_class	Verified	*Each auth-class scheduler runs only on the auth timelock.*	Report
schedule_admin_pinned	Verified	*timelock_ops::schedule_admin runs only on the admin timelock that the CapStore records, and only at the current version.*	Report
schedule_auth_pinned	Verified	*timelock_ops::schedule_auth runs only on the auth timelock that the CapStore records, and only at the current version.*	Report
execute_admin_pinned	Verified	*timelock_ops::execute_admin runs only on the admin timelock that the CapStore records, and only at the current version.*	Report
execute_auth_pinned	Verified	*timelock_ops::execute_auth runs only on the auth timelock that the CapStore records, and only at the current version.*	Report

TL-ACC-C1

Status: Verified	schedule, execute, cancel and the setting entry points of the timelock run only for a sender who holds the role that the timelock assigns to that action.

Rule Name	Status	Description	Link to rule report
role_gate_schedule	Verified	*timelock::schedule runs only for a sender who holds the timelock's proposer role.*	Report
role_gate_execute	Verified	*timelock::execute runs only for a sender who holds the timelock's executor role.*	Report
role_gate_cancel	Verified	*timelock::cancel runs only for a sender who holds the timelock's canceller role.*	Report
role_gate_schedule_update_min_delay	Verified	*timelock::schedule_update_min_delay runs only for a sender who holds the timelock's admin role.*	Report
role_gate_execute_update_min_delay	Verified	*timelock::execute_update_min_delay runs only for a sender who holds the timelock's admin role.*	Report
role_gate_schedule_update_grace_period	Verified	*timelock::schedule_update_grace_period runs only for a sender who holds the timelock's admin role.*	Report
role_gate_execute_update_grace_period	Verified	*timelock::execute_update_grace_period runs only for a sender who holds the timelock's admin role.*	Report

TL-RULE-S4

Status: Verified	The class helpers delay an operation by the class floor, 12 h for admin and 6 h for auth, or by the timelock's minimum delay if that is longer, and no operation that a class scheduler creates is ready before its class floor.

Rule Name	Status	Description	Link to rule report
admin_scheduler_delay_floor	Verified	*No operation that an admin-class scheduler creates is ready before 12 h.*	Report
auth_scheduler_delay_floor	Verified	*No operation that an auth-class scheduler creates is ready before 6 h.*	Report
schedule_admin_delay_floor	Verified	*After timelock_ops::schedule_admin, the operation becomes ready after 12 h or the timelock's minimum delay, whichever is longer, and stays ready for one grace period.*	Report
schedule_auth_delay_floor	Verified	*After timelock_ops::schedule_auth, the operation becomes ready after 6 h or the timelock's minimum delay, whichever is longer, and stays ready for one grace period.*	Report

Timelock Role Governance

Module Properties

TL-INV-C3

Status: Verified	No grant and no finalize_roles gives an address both GuardianRole and AdminCancellerRole.

Rule Name	Status	Description	Link to rule report
grant_never_creates_role_conflict_guardian	Verified	*An executed GrantRole of GuardianRole never gives an address both GuardianRole and AdminCancellerRole.*	Report
grant_never_creates_role_conflict_admin_canceller	Verified	*An executed GrantRole of AdminCancellerRole never gives an address both GuardianRole and AdminCancellerRole.*	Report
bootstrap_grant_guardian	Verified	*role_gov::grant_role of GuardianRole runs only during setup and only for an address without AdminCancellerRole. The holder count rises by one.*	Report
bootstrap_grant_admin_canceller	Verified	*role_gov::grant_role of AdminCancellerRole runs only during setup and only for an address without GuardianRole. The holder count rises by one.*	Report
finalize_roles_leaves_no_role_conflict	Verified	*role_gov::finalize_roles never gives an address both GuardianRole and AdminCancellerRole.* *Assumption: Before finalize_roles, setup leaves no address with both GuardianRole and AdminCancellerRole, as documented in docs/internal/pre-launch-gates.md.*	Report

TL-INV-C4

Status: Verified	A scheduled grant or revoke of ExecutorRole changes only the scheduled address, no revoke of ExecutorRole removes its last holder, and setup completes only when every role has a holder and every layer cap is stored.

Rule Name	Status	Description	Link to rule report
grant_round_trip_executor	Verified	*An executed GrantRole of ExecutorRole gives the role to the scheduled address only, if it did not hold it yet, and the holder count rises by one.*	Report
revoke_round_trip_executor	Verified	*An executed RevokeRole of ExecutorRole removes the role from the scheduled address only, and only if another holder remains. The holder count drops by one.*	Report
bootstrap_revoke_keeps_a_holder	Verified	*role_gov::revoke_role of ExecutorRole runs only during setup and only if another holder remains. The holder count drops by one.*	Report
finalize_roles_needs_every_holder	Verified	*role_gov::finalize_roles runs only once, and only if every role has a holder.*	Report
complete_setup_needs_all_caps	Verified	*role_gov::complete_setup runs only after finalize_roles, and only if the CapStore holds the liquidity, oracle, vaults, lending and rewards caps.*	Report

TL-TEMP-C1

Status: Verified	A guardian's cancel of a GrantRole runs only on the admin timelock and never cancels a pending RemoveGuardian, an executed RemoveGuardian strips exactly the scheduled guardian, only an executor completes the removal, and a revoke of AuthCancellerRole never touches a guardian.

Rule Name	Status	Description	Link to rule report
cancel_admin_gate_grant_role	Verified	*guardian_gov::cancel_admin of a GrantRole runs only for a guardian, on the admin timelock and at the current version, and removes the operation.*	Report
cancel_admin_never_reaches_remove_guardian	Verified	*A guardian cancel of a GrantRole never cancels a pending RemoveGuardian.*	Report
remove_guardian_round_trip	Verified	*An executed RemoveGuardian runs only on the admin timelock, removes GuardianRole from the scheduled guardian, and AuthCancellerRole too if it was a guardian, and marks its liquidity removal as owed. The guardian and auth canceller records of every other address and the holder counts of the other roles do not change.*	Report
complete_remove_guardian_exact	Verified	*guardian_gov::complete_remove_guardian runs only for an executor and only for an owed removal of an address that is no longer a guardian. It clears that removal and the liquidity guardian record of that address only.*	Report
revoke_role_spares_guardians	Verified	*An executed RevokeRole of AuthCancellerRole never removes that role from a guardian.*	Report

Timelock Cap Custody

Module Properties

TL-INV-05

Status: Verified	init creates the admin and auth timelocks with their launch settings, and for the liquidity, lending and vaults layers enable_timelock leaves the CapStore holding the cap the layer records, while execute_disable_timelock releases the cap and clears the record together.

Rule Name	Status	Description	Link to rule report
enable_timelock_liquidity	Verified	*After liquidity_gov::enable_timelock, the CapStore holds the liquidity cap that the liquidity layer records.*	Report
enable_timelock_lending	Verified	*After lending_gov::enable_timelock, the CapStore holds the lending cap that the lending layer records.*	Report
enable_timelock_vaults	Verified	*After vaults_gov::enable_timelock, the CapStore holds the vaults cap that the vaults layer records.*	Report
execute_disable_timelock_liquidity	Verified	*After liquidity_gov::execute_disable_timelock, the CapStore holds no liquidity cap and the liquidity layer records none.*	Report
execute_disable_timelock_lending	Verified	*After lending_gov::execute_disable_timelock, the CapStore holds no lending cap and the lending layer records none.*	Report
execute_disable_timelock_vaults	Verified	*After vaults_gov::execute_disable_timelock, the CapStore holds no vaults cap and the vaults layer records none.*	Report

init_pins_canonical_timelocks	Verified	*cap_store::init shares one CapStore and two distinct timelocks, records them as the admin and auth timelocks, and gives them their roles, a 12 h and a 6 h minimum delay, and a 7 day grace period.*	Report

TL-RULE-S1

Status: Verified	The liquidity, lending and vaults tickets come only from a cap the CapStore holds, one execute per layer runs while that layer's timelock is on only with the held cap the layer records, and an upgrade needs the held upgrade cap.

Rule Name	Status	Description	Link to rule report
ticket_of_fails_closed_liquidity	Verified	*liquidity_gov::ticket_of returns no ticket while the CapStore holds no liquidity cap, and otherwise a ticket for the cap the liquidity layer records.*	Report
ticket_of_fails_closed_lending	Verified	*lending_gov::ticket_of returns no ticket while the CapStore holds no lending cap, and otherwise a ticket for the cap the lending layer records.*	Report
ticket_of_fails_closed_vaults	Verified	*vaults_gov::ticket_of returns no ticket while the CapStore holds no vaults cap, and otherwise a ticket for the cap the vaults layer records.*	Report
ticket_of_fails_closed_vaults_liq	Verified	*vaults_gov::liq_ticket returns no ticket while the CapStore holds no liquidity cap, and otherwise a ticket for the cap the liquidity layer records.*	Report
execute_requires_held_cap_liquidity	Verified	*While the liquidity timelock is on, liquidity_gov::execute_update_revenue_collector runs only if the CapStore holds the cap the liquidity layer records.*	Report
execute_requires_held_cap_lending	Verified	*While the lending timelock is on, lending_gov::execute_update_admin runs only if the CapStore holds the cap the lending layer records.*	Report

execute_requires_held_cap_vaults	Verified	*While the vaults timelock is on, vaults_gov::execute_update_rebalancer runs only if the CapStore holds the cap the vaults layer records.*	Report
execute_requires_held_cap_vaults_liq	Verified	*While the liquidity timelock is on, vaults_gov::execute_mint_and_install_caps runs only if the CapStore holds the cap the liquidity layer records.*	Report
upgrade_round_trip	Verified	*upgrade_gov::schedule_upgrade runs only for an admin proposer while the CapStore holds the upgrade cap, and the operation waits at least 12 h. Once executed, it returns an upgrade ticket for the held cap's package with exactly the scheduled policy and digest.*	Report
commit_upgrade_needs_version	Verified	*upgrade_gov::commit_upgrade runs only at the current version and only while the CapStore holds the upgrade cap, which it keeps.*	Report

Timelock Protocol Wrappers

Module Properties

TL-RULE-S2

Status: Verified	The reserve-targeted liquidity actions execute only on the scheduled reserve, and MintUserCap, UpdateRole and InitVaultConfig execute only under the type they were scheduled with.

Rule Name	Status	Description	Link to rule report
update_role_refuses_other_type	Verified	*An UpdateRole scheduled for RebalancerRole never executes for another role.*	Report
mint_user_cap_refuses_other_type	Verified	*A MintUserCap scheduled for one token type never executes under another token type.*	Report
unpause_token_binds_reserve	Verified	*An executed UnpauseToken runs only on the admin timelock and only on the scheduled reserve.*	Report
unpause_user_binds_reserve	Verified	*An executed UnpauseUser runs only on the admin timelock and only on the scheduled reserve.*	Report
update_rate_data_v1_binds_reserve	Verified	*An executed UpdateRateDataV1 runs only on the auth timelock and only on the scheduled reserve.*	Report
update_rate_data_v2_binds_reserve	Verified	*An executed UpdateRateDataV2 runs only on the auth timelock and only on the scheduled reserve.*	Report

update_token_config_binds_reserve	Verified	*An executed UpdateTokenConfig runs only on the auth timelock and only on the scheduled reserve.*	Report
update_user_supply_config_binds_reserve	Verified	*An executed UpdateUserSupplyConfig runs only on the auth timelock and only on the scheduled reserve.*	Report
update_user_borrow_configs_binds_reserve	Verified	*An executed UpdateUserBorrowConfigs runs only on the auth timelock and only on the scheduled reserve.*	Report
update_user_withdrawal_limit_binds_reserve	Verified	*An executed UpdateUserWithdrawalLimit runs only on the auth timelock and only on the scheduled reserve.*	Report
collect_revenue_binds_reserve	Verified	*An executed CollectRevenue runs only on the auth timelock and only on the scheduled reserve.*	Report
init_vault_config_refuses_other_types	Verified	*An InitVaultConfig scheduled for one collateral and debt type pair never executes under the swapped pair.*	Report

TL-RULE-S3

Status: Verified	The executed admin, role, guardian, revenue collector, cap minting, collateral factor and oracle actions apply exactly the scheduled values, on the timelock of their class.

Rule Name	Status	Description	Link to rule report
update_admin_round_trip	Verified	*An executed UpdateAdmin runs only on the admin timelock and sets the admin status of the scheduled address, and no other, to the scheduled value.*	Report

update_role_round_trip	Verified	*An executed UpdateRole of RebalancerRole runs only on the admin timelock and sets that role of the scheduled address, and no other, to the scheduled value.*	Report
mint_rewards_writer_cap_round_trip	Verified	*An executed MintRewardsWriterCap runs only on the admin timelock and only on the scheduled rewards state. It mints one writer cap for that state, records it, and sends it to the scheduled recipient.*	Report
add_admin_round_trip (liquidity_wrappers)	Verified	*An executed AddAdmin runs only on the admin timelock and makes the scheduled address, and no other, a liquidity admin. The admin count rises by one.*	Report
remove_admin_round_trip	Verified	*An executed RemoveAdmin runs only on the admin timelock and only if another admin remains. It removes the scheduled address, and no other, from the liquidity admins, and the admin count drops by one.*	Report
update_guardians_round_trip	Verified	*Only a guardian addition can be scheduled. Once executed, it runs only on the admin timelock and adds the scheduled guardian, and no other.*	Report
update_revenue_collector_round_trip	Verified	*An executed UpdateRevenueCollector runs only on the admin timelock and sets the scheduled revenue collector.*	Report
mint_user_cap_round_trip	Verified	*An executed MintUserCap runs only on the admin timelock and only if the user has no cap yet. It mints one cap for the scheduled user and token and sends it to the scheduled recipient.*	Report
set_user_authorization_round_trip	Verified	*An executed SetUserAuthorization runs only on the admin timelock and turns the user's cap on or off as scheduled.*	Report
add_admin_round_trip (vaults_wrappers)	Verified	*An executed AddAdmin runs only on the admin timelock and makes the scheduled address, and no other, a vaults admin. The admin count rises by one.*	Report

mint_and_install_caps_round_trip	Verified	*An executed MintAndInstallCaps runs only on the admin timelock and only on the scheduled vault. It installs a liquidity cap for exactly the scheduled sides, collateral, debt or both.*	Report
update_collateral_factor_round_trip	Verified	*An executed UpdateCollateralFactor runs only on the auth timelock and only on the scheduled vault, and sets the scheduled collateral factor. The vault stores the value divided by ten.*	Report
update_oracle_round_trip	Verified	*An executed UpdateOracle runs only on the auth timelock and only on the scheduled vault and oracle, and sets the vault's oracle to the scheduled one.*	Report

Timelock Protocol Gates

Module Properties

TL-ACC-01	
Status: Verified	In liquidity, lending and vaults, while a TimelockCap is recorded, every gated entry point needs a ticket of the recorded cap, and the sender guard of its tier still runs.

Rule Name	Status	Description	Link to rule report
lending_gate_init_lending	Verified	*admin::init_lending returns only if the sender was an admin and, while a cap is recorded, the ticket carries that cap.*	Report
lending_gate_init_lending_with_metadata	Verified	*admin::init_lending_with_metadata returns only if the sender was an admin and, while a cap is recorded, the ticket carries that cap.*	Report
lending_gate_update_admin	Verified	*admin::update_admin returns only if the sender was an admin and, while a cap is recorded, the ticket carries that cap. The recorded cap is unchanged.*	Report
lending_gate_update_role	Verified	*admin::update_role returns only if the sender was an admin and, while a cap is recorded, the ticket carries that cap. The recorded cap is unchanged.*	Report
lending_gate_mint_rewards_writer_cap	Verified	*admin::mint_rewards_writer_cap returns only if the sender was an admin and, while a cap is recorded, the ticket carries that cap.*	Report
lending_gate_revoke_rewards_writer	Verified	*admin::revoke_rewards_writer returns only if the sender was an admin and, while a cap is recorded, the ticket carries that cap.*	Report

liquidity_gate_add_admin	Verified	*admin::add_admin returns only if the sender was governance and, while a cap is recorded, the ticket carries that cap. The recorded cap is unchanged.*	Report
liquidity_gate_remove_admin	Verified	*admin::remove_admin returns only if the sender was governance and, while a cap is recorded, the ticket carries that cap. The recorded cap is unchanged.*	Report
liquidity_gate_update_auths	Verified	*admin::update_auths returns only if the sender was governance and, while a cap is recorded, the ticket carries that cap. The recorded cap is unchanged.*	Report
liquidity_gate_update_guardians	Verified	*admin::update_guardians returns only if the sender was governance and, while a cap is recorded, the ticket carries that cap. The recorded cap is unchanged.*	Report
liquidity_gate_update_migrators	Verified	*admin::update_migrators returns only if the sender was governance and, while a cap is recorded, the ticket carries that cap. The recorded cap is unchanged.*	Report
liquidity_gate_update_revenue_collector	Verified	*admin::update_revenue_collector returns only if the sender was governance and, while a cap is recorded, the ticket carries that cap.*	Report
liquidity_gate_mint_user_cap	Verified	*admin::mint_user_cap returns only if the sender was governance and, while a cap is recorded, the ticket carries that cap.*	Report
liquidity_gate_set_user_authorization	Verified	*admin::set_user_authorization returns only if the sender was governance and, while a cap is recorded, the ticket carries that cap.*	Report
liquidity_gate_unpause_global	Verified	*admin::unpause_global returns only if the sender was governance and, while a cap is recorded, the ticket carries that cap.*	Report
liquidity_gate_unpause_token	Verified	*admin::unpause_token returns only if the sender was governance and, while a cap is recorded, the ticket carries that cap.*	Report

liquidity_gate_unpause_user	Verified	*admin::unpause_user returns only if the sender was governance and, while a cap is recorded, the ticket carries that cap.*	Report
liquidity_gate_init_token_reserve	Verified	*admin::init_token_reserve returns only if the sender was an auth or governance and, while a cap is recorded, the ticket carries that cap.*	Report
liquidity_gate_init_token_reserve_with_metadata	Verified	*admin::init_token_reserve_with_metadata returns only if the sender was an auth or governance and, while a cap is recorded, the ticket carries that cap.*	Report
liquidity_gate_update_rate_data_v1	Verified	*admin::update_rate_data_v1 returns only if the sender was an auth or governance and, while a cap is recorded, the ticket carries that cap.*	Report
liquidity_gate_update_rate_data_v2	Verified	*admin::update_rate_data_v2 returns only if the sender was an auth or governance and, while a cap is recorded, the ticket carries that cap.*	Report
liquidity_gate_update_token_config	Verified	*admin::update_token_config returns only if the sender was an auth or governance and, while a cap is recorded, the ticket carries that cap.*	Report
liquidity_gate_update_user_supply_config	Verified	*admin::update_user_supply_config returns only if the sender was an auth or governance and, while a cap is recorded, the ticket carries that cap.*	Report
liquidity_gate_update_user_borrow_configs	Verified	*admin::update_user_borrow_configs returns only if the sender was an auth or governance and, while a cap is recorded, the ticket carries that cap.*	Report
liquidity_gate_update_user_withdrawal_limit	Verified	*admin::update_user_withdrawal_limit returns only if the sender was an auth or governance and, while a cap is recorded, the ticket carries that cap.*	Report

liquidity_gate_collect_revenue	Verified	*admin::collect_revenue returns only if the sender was an auth or governance and, while a cap is recorded, the ticket carries that cap.*	Report
vaults_gate_update_rebalancer	Verified	*admin::update_rebalancer returns only if the sender was an admin and, while a cap is recorded, the ticket carries that cap. The recorded cap is unchanged.*	Report
vaults_gate_update_auths	Verified	*admin::update_auths returns only if the sender was an admin and, while a cap is recorded, the ticket carries that cap. The recorded cap is unchanged.*	Report
vaults_gate_update_migrators	Verified	*admin::update_migrators returns only if the sender was an admin and, while a cap is recorded, the ticket carries that cap. The recorded cap is unchanged.*	Report
vaults_gate_add_admin	Verified	*admin::add_admin returns only if the sender was an admin and, while a cap is recorded, the ticket carries that cap. The recorded cap is unchanged.*	Report
vaults_gate_remove_admin	Verified	*admin::remove_admin returns only if the sender was an admin and, while a cap is recorded, the ticket carries that cap. The recorded cap is unchanged.*	Report
vaults_gate_create_display	Verified	*position::create_display returns only if the sender was an admin and, while a cap is recorded, the ticket carries that cap. The recorded cap is unchanged.*	Report
vaults_gate_init_vault_config	Verified	*admin::init_vault_config returns only if the sender was an auth or admin and, while a cap is recorded, the ticket carries that cap.*	Report
vaults_gate_init_vault_state	Verified	*admin::init_vault_state returns only if the sender was an auth or admin and, while a cap is recorded, the ticket carries that cap.*	Report
vaults_gate_mint_and_install_caps	Verified	*admin::mint_and_install_caps returns only if the sender was an auth or admin and, while a vaults cap is recorded, the vaults ticket carries that cap.*	Report

vaults_gate_update_supply_rate_magnifier	Verified	*admin::update_supply_rate_magnifier returns only if the sender was an auth or admin and, while a cap is recorded, the ticket carries that cap.*	Report
vaults_gate_update_borrow_rate_magnifier	Verified	*admin::update_borrow_rate_magnifier returns only if the sender was an auth or admin and, while a cap is recorded, the ticket carries that cap.*	Report
vaults_gate_update_collateral_factor	Verified	*admin::update_collateral_factor returns only if the sender was an auth or admin and, while a cap is recorded, the ticket carries that cap.*	Report
vaults_gate_update_liquidation_threshold	Verified	*admin::update_liquidation_threshold returns only if the sender was an auth or admin and, while a cap is recorded, the ticket carries that cap.*	Report
vaults_gate_update_liquidation_max_limit	Verified	*admin::update_liquidation_max_limit returns only if the sender was an auth or admin and, while a cap is recorded, the ticket carries that cap.*	Report
vaults_gate_update_withdraw_gap	Verified	*admin::update_withdraw_gap returns only if the sender was an auth or admin and, while a cap is recorded, the ticket carries that cap.*	Report
vaults_gate_update_liquidation_penalty	Verified	*admin::update_liquidation_penalty returns only if the sender was an auth or admin and, while a cap is recorded, the ticket carries that cap.*	Report
vaults_gate_update_borrow_fee	Verified	*admin::update_borrow_fee returns only if the sender was an auth or admin and, while a cap is recorded, the ticket carries that cap.*	Report
vaults_gate_update_core_settings	Verified	*admin::update_core_settings returns only if the sender was an auth or admin and, while a cap is recorded, the ticket carries that cap.*	Report

vaults_gate_update_oracle	Verified	*admin::update_oracle returns only if the sender was an auth or admin and, while a cap is recorded, the ticket carries that cap.*	Report

TL-RULE-01

Status: Verified	In liquidity, lending and vaults, mint_timelock_cap runs only while no cap is recorded, records exactly the returned cap, needs a top-tier sender at the current version, and changes nothing else.

Rule Name	Status	Description	Link to rule report
lending_switch_mint_timelock_cap	Verified	*admin::mint_timelock_cap returns only from an admin sender at the current version while no cap is recorded. Afterwards the returned cap is the recorded one, and the role records of any address are unchanged.*	Report
liquidity_switch_mint_timelock_cap	Verified	*admin::mint_timelock_cap returns only from a governance sender at the current version while no cap is recorded. Afterwards the returned cap is the recorded one, and admin_count and the role records of any address are unchanged.*	Report
vaults_switch_mint_timelock_cap	Verified	*admin::mint_timelock_cap returns only from an admin sender at the current version while no cap is recorded. Afterwards the returned cap is the recorded one, and admin_count and the role records of any address are unchanged.*	Report

TL-RULE-02

Status: Verified	In liquidity, lending and vaults, disable_timelock needs the recorded cap by value and a top-tier sender at the current version, clears the record, and changes nothing else.

Rule Name	Status	Description	Link to rule report
lending_switch_disable_timelock	Verified	*admin::disable_timelock returns only from an admin sender at the current version with the recorded cap. Afterwards no cap is recorded, and the role records of any address are unchanged.*	Report
liquidity_switch_disable_timelock	Verified	*admin::disable_timelock returns only from a governance sender at the current version with the recorded cap. Afterwards no cap is recorded, and admin_count and the role records of any address are unchanged.*	Report
vaults_switch_disable_timelock	Verified	*admin::disable_timelock returns only from an admin sender at the current version with the recorded cap. Afterwards no cap is recorded, and admin_count and the role records of any address are unchanged.*	Report

TL-RULE-03

Status: Verified	In liquidity, lending and vaults, timelock_ticket mints only for the recorded cap and binds the ticket to that cap.

Rule Name	Status	Description	Link to rule report
lending_timelock_ticket_binds_recorded_cap	Verified	*auth::timelock_ticket returns only if the cap is the recorded one, and the ticket carries that cap's id.*	Report
liquidity_timelock_ticket_binds_recorded_cap	Verified	*auth::timelock_ticket returns only if the cap is the recorded one, and the ticket carries that cap's id.*	Report
vaults_timelock_ticket_binds_recorded_cap	Verified	*auth::timelock_ticket returns only if the cap is the recorded one, and the ticket carries that cap's id.*	Report

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline and is widely used by developers and security researchers during audits and bug bounties.

Certora also provides architectural design reviews, where researchers analyze system design and trust boundaries early to identify structural risks and reduce complexity before implementation.

Certora conducts off-chain audits covering backend services, APIs, frontend, and mobile applications, focusing on vulnerabilities in authentication, data handling, key management, and wallet-related workflows.

Certora also offers Penetration Testing and Red Teaming to simulate real-world attacks and uncover exploitable weaknesses across infrastructure, applications, and business logic.

In addition, Certora provides operational security (OpSec) assessments, reviewing key management, access controls, and operational processes to strengthen overall security posture.