certora

a

Security Assessment

Final Report

fluid

Fluid Oracles

October 2026

Prepared for Fluid

Table of Contents

Project Summary **3**
- Project Scope 3
- Project Overview 3
- Protocol Overview 3

Assessment Methodology **4**

Threat and Security Overview **5**
- Findings Summary 10
- Severity Matrix 10

Detailed Findings **11**
- Low Severity Issues 13
  - L-01: An aging low route overrides a fresh route and liquidates healthy positions 13
  - L-02: Single-feed oracle routes price a stablecoin leg at an implicit $1 15
  - L-03: A vault's oracle is not bound to the market's pair, and repointing takes effect instantly 17
  - L-04: The live shared Pyth Pro contract permits governance replay after source rotation 18
  - L-05: Fixed 600 s operate age blocks borrowers on 24-hour-heartbeat RedStone feeds 20
  - L-06: Unrefreshed Pyth prices up to 600 seconds old let borrowers borrow at CF against a stale-high price 22
  - L-07: Single-route Pyth markets let a liquidator use a stale dip price for up to 7,200 seconds 24
  - L-08: A late or wide-confidence Pyth update can make the shared price object temporarily unreadable ... 26
  - L-09: Health gates reject ratio-lowering partial deleverage above LT or CF, forcing borrowers into extra liquidation penalty 28
  - L-10: Liquidation-only price availability blocks partial borrower rescue actions 30
- Informational Severity Issues 32
  - I-01: Oracle creation does not cross-validate the main and guard routes' pair or inversion 32
  - I-02: Oracles cannot follow RedStone to a new adapter object without per-vault repointing 34
  - I-03: Oracle and vault do not share a price domain: scale_and_invert can emit rates the vault rejects. 36
  - I-04: Last-admin guards can count an unusable role record 38
  - I-05: Oracle fixes take effect only through a relinked, version-migrated Vault upgrade 40
  - I-06: Oracle and Vaults cannot be published to Sui mainnet: manifests omit RedStone's system dependencies 42

Disclaimer **44**

About Certora **44**

Project Summary

Project Scope

Project Name	Repository Link	Commit Hash	Platform
Fluid Oracles	https://github.com/Instadapp/fluid-packages-sui/	Initial: b92124e Final: 870cf29	Sui/Move

Project Overview

This document describes the security review of **Fluid Oracles (PR-36)**. The work was undertaken from **September 28th, 2026** to **October 1st, 2026**.

The following contract list is included in our scope:

packages/oracle/sources/**
The team performed a manual audit of all the files in scope**.** During the manual audit, the Certora team discovered bugs in the code, as listed on the following page.

Protocol Overview

Fluid on Sui is Instadapp's port of the Fluid protocol to Sui Move. It is built from six packages around a single shared liquidity layer. Each listed coin has one shared reserve that holds all deposits and accrues supply and borrow interest from a kinked rate model. Only authorized protocol modules can use the reserve, and each one is subject to withdrawal limits and borrow ceilings. Two products draw on it. Lending offers ERC-4626-style fTokens whose yield combines liquidity interest with a rewards schedule. Vaults provide overcollateralized borrowing on a tick-based position ladder, with batched liquidation through aggregated "branches". Vault pricing comes from the Oracle package.It reads Pyth and RedStone feeds through a main source and an optional guard source, and applies separate freshness and confidence bounds for normal operations and for liquidations.

Assessment Methodology

Our assessment approach combines design level analysis with a deep review of the implementation to ensure that a protocol is secure, economically sound, and behaves as intended under realistic conditions.

At the design level, we evaluate the architecture, the economic assumptions behind the protocol, and the safety properties that should hold independently of a specific chain or environment. This process includes reviewing internal and cross protocol interactions, state transition flows, trust boundaries, and any mechanism that could be exploited to extract value, deny service, or alter core system behavior. At this stage, a focused threat modelling exercise helps identify key attack surfaces and adversarial capabilities relevant to the system. Design level issues often relate to incentive structures, governance implications, or systemic behavior that emerges under adversarial conditions.

Implementation analysis focuses on the concrete behavior of the code within the execution model of the target chain. This involves reviewing the correctness of logic, access control, state handling, arithmetic behavior, and the nuanced behaviors of the chain environment. Familiar classes of vulnerabilities such as reentrancy conditions, faulty permission checks, precision issues, or unsafe assumptions often surface at this layer. These findings require context aware reasoning that takes into account both the code and the architectural intent.

To support this analysis, the codebase is examined through repeated manual passes and supplemented by automated tools when appropriate. High-risk logic areas receive deeper scrutiny, invariants are validated against both design intent and actual implementation, and potential vulnerability leads are thoroughly investigated. Automated techniques such as static analysis, fuzzing, or symbolic execution may be used to complement manual review and provide additional insight.

Collaboration with the development team plays an important role throughout the audit. This helps confirm expected behaviors, clarify design assumptions, and ensure an accurate understanding of the protocol's intended operation. All findings are documented with clear reasoning, reproducible examples, and actionable recommendations. A follow up review is conducted to validate the applied fixes and verify that no regressions or secondary issues have been introduced.

certora

Threat and Security Overview

System model

The Oracle package turns third-party price feeds into a rate that a vault can trust for one transaction. Its security-relevant elements are:

- `OracleRegistry` and `Auth`: the package version and two roles. `AdminRole` grants and revokes roles and migrates the registry. `AuthorizeRole` creates oracles.
- `Oracle`: a main `Source` and an optional guard `Source`, each a single Pyth or RedStone feed with an inversion flag. A RedStone source also pins its adapter object ID, and two routes must use different providers. An oracle cannot be edited after creation and stores no price.
- Provider readers: `pyth_core` checks feed identity, age, arrival lag, sign, exponent and confidence on the shared Pyth `PriceInfoObject`. `redstone_core` checks adapter identity, age and value range on the pinned RedStone `PriceAdapter`.
- `view`: builds a `PriceQuote` that carries the oracle ID and an operate rate, a liquidation rate, or both.
- Grades and route rules (by design): the operate grade (600 s age, 2% Pyth confidence) is stricter than the liquidation grade (7,200 s age, 4% confidence). Both grades reject a Pyth print that arrived more than 60 s after its publish time. Operate needs every route, while liquidation drops a dead route; both take the lower of the live routes.
- Vault consumption: the vault accepts only quotes from its bound oracle. It checks withdrawals and borrows against the collateral factor at the operate rate, and deposits and repayments against the liquidation threshold at the liquidation rate. The check runs only when the position keeps debt, and it reads both rates before choosing one, so every debt-leaving action also needs an operate-grade quote (L-10). It also converts the rate into ticks for liquidation and absorption.

The oracle holds no funds. Value moves in the vaults, where the rate decides how much a borrower may draw, whether a position can be liquidated and how much collateral a liquidator receives.

Invariants

The review stated these properties and tried to break each one. These held:

- A quote cannot be forged, stored or carried across transactions, and it binds to one oracle.
- A wrong adapter, feed or price object aborts rather than triggering a fallback, and substituted oracle, provider, feed and adapter identities are caught.
- Zero and negative prices, out-of-range exponents and over-large inverted prices are rejected.
- A liquidator cannot choose the price, within the documented design.
- The capped high-price liquidation value stays conservative for solvency.

These were broken or only partly held:

- An aged route never overrides a fresh one (L-01; single-route markets are exposed through L-07).
- Operate-grade prices bound risk-increasing actions (L-06).
- A vault's oracle prices that vault's pair, inside the vault's price domain (L-02, L-03, I-01, I-03).
- A borrower can rescue a position whenever liquidators can act on it (L-09, L-10).
- Age limits match provider cadence (L-05).
- A shared Pyth price object stays readable after any permissionless update (L-08).
- Upstream provider governance is replay-safe (L-04).
- Oracles survive a provider adapter migration and publish against the live provider packages (I-02, I-06).
- The Oracle always keeps a usable administrator, and oracle fixes reach the vaults (I-04, I-05).

Trust boundaries and assumptions

- Sui object ownership, shared-object scheduling, arithmetic aborts and package upgrades are assumed to behave as specified.

- Pyth operates the shared Pyth package and State. Anyone can store a newer guardian-signed print for a fee of 0, so the caller decides whether a fresh price is present. Pyth governance is assumed safe; L-04 records where the live package is not.
- RedStone's relayer writes prices on an off-chain cadence that can change without a Fluid release. The live adapter currently limits untrusted senders to one write every 48 hours (`min_interval_between_updates_ms`). RedStone controls this setting, and it can also change without a Fluid release. The connector fixes feeds at 8 decimals.
- Governance is trusted to choose feeds, inversion flags, pairs, token decimals and role holders. The vault `auths` tier (any `AuthorizeRole` or `AdminRole` holder on `VaultAdmin`) can repoint a vault's oracle instantly, without holding the oracle package's `AuthorizeRole`.
- Users, liquidators and keepers are untrusted. They choose which provider objects to pass and whether to refresh Pyth.
- No production addresses, oracle configurations, feed lists or economic parameters were supplied, so live wiring was not validated.

Attack vectors considered

- Roles and access rights: oracle creation without `AuthorizeRole`, role escalation, loss of the last usable administrator, and oracle repointing (L-03, I-04).
- Quote integrity: forgery, persistence, replay, reuse in one programmable transaction, and quotes from an older package version (I-05).
- Object substitution: a wrong adapter, feed, oracle or pair (L-03, I-01). The adapter pin that blocks substitution also prevents following a legitimate RedStone adapter migration (I-02).
- Freshness and route selection: stale-high borrowing, stale-dip liquidation, aged against fresh routes, disagreeing fresh routes, and late or wide-confidence Pyth prints (L-01, L-06, L-07, L-08).
- Value flows and economic risks: over-borrowing, wrongful liquidation, oversized seizure, wholesale absorption, stablecoin depegs and blocked deleverage (L-01, L-02, L-09, L-10).
- Price arithmetic: scaling, inversion rounding, overflow and the vault's price and tick edges (I-03).

• Dependency surfaces: the exact Pyth, Wormhole and RedStone pins, including upstream fixes, live mainnet objects, governance replay, relayer cadence, adapter migration and mainnet publishability (L-04, L-05, I-02, I-06).

• Cross-protocol interactions: how provider outages reach vault rescue and liquidation (L-10, L-05).

Review, testing and verification

• Manual review covered all 12 Oracle production modules and the vault price-consumption paths in scope. It compared the removed hop, math and Pyth modules against their replacements, and checked the relevant provider code at the pinned revisions.

• The 370 actionable questions of a 430-item public audit checklist (Solodit) were dispositioned for the package graph, and external EVM heuristics were translated to Sui equivalents (object identity, PTB composition, provider objects). A 2,134-pattern vulnerability library produced 108 leads, each refuted or reproduced in a sandbox.

• The project suites passed with Sui 1.80.1: 72/72 Oracle tests and 151/151 Vault tests, at 98.53% and 83.93% production-code coverage. They also passed on Sui 1.79.1, with identical production bytecode.

• Separate from the project suites, every finding has archived reproduction evidence run against the unchanged sources at commit 4d7b051: a Move PoC for 15 findings (L-04's against the pinned Pyth source) and mainnet publish dry-runs for I-06. A local validator also ran 17 transactions with signed Pyth and RedStone updates. They covered the 60/61-second arrival-lag boundary, both fallback directions, poisoning with atomic rollback, a wrong adapter, and
provider-backed operate, rescue and liquidation. Publish, upgrade and migration runs, mainnet publish dry-runs and defense-removal mutants completed the checks.

• Live data sized the likelihoods: 39,150 RedStone price writes over about 239 days, RedStone's relayer manifest, and 30 days of Pyth mainnet price objects.

• No coverage-guided or stateful fuzzing campaign targeted the oracle, and no formal verification was performed; the review used bounded generated models and exact counterexamples. Bounded generated-input probes (128 real-fixture Pyth scaling cases and 2,000 arbitrary source encodings) held their invariants; they were executed and then

removed, not archived. Liquidator-versus-borrower races were modelled by transaction order, not run on a validator.

Test coverage

The Oracle suite has 72 tests, 42 of which expect an abort. They cover:

- the two quote grades and boundaries on age, arrival lag and confidence;
- future timestamps and a rewound clock;
- route minimum, fallback in both directions, and both routes dead;
- a wrong adapter, feed or unwritten feed;
- price validity: zero, negative, exponent limits, inversion, and RedStone timestamps and bounds;
- oracle creation, roles, the last-admin guard and migration.

Five of the 151 Vault tests run on a fixture with a real signed RedStone adapter. One prices a borrow end to end. Two check at quote level that a 601 s-old price aborts the operate grade and is still accepted by the vault's liquidation-rate accessor. One checks that the quote matches the signed payload. One checks that the vault rejects a quote from its previous oracle, using a fabricated quote. No test runs a liquidation or rescue through a real provider quote. All other Vault tests use a fabricated quote that never goes stale.

Findings Summary

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

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	-	-	-
Low	10	9	3
Informational	6	6	3
Total	16	15	6

Severity Matrix

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

certora

Detailed Findings

ID	Title	Severity	Status
L-01	[L-01] An aging low route overrides a fresh route and liquidates healthy positions	Low	Fixed
L-02	[L-02] Single-feed oracle routes price a stablecoin leg at an implicit $1	Low	Acknowledged
L-03	[L-03] A vault's oracle is not bound to the market's pair, and repointing takes effect instantly	Low	Acknowledged
L-04	[L-04] The live shared Pyth Pro contract permits governance replay after source rotation	Low	Acknowledged
L-05	[L-05] Fixed 600 s operate age blocks borrowers on 24-hour-heartbeat RedStone feeds	Low	Acknowledged
L-06	[L-06] Unrefreshed Pyth prices up to 600 seconds old let borrowers borrow at CF against a stale-high price	Low	Acknowledged
L-07	[L-07] Single-route Pyth markets let a liquidator use a stale dip price for up to 7,200 seconds	Low	Acknowledged
L-08	[L-08] A late or wide-confidence Pyth update can make the shared price object temporarily unreadable	Low	Acknowledged

L-09	[L-09] Health gates reject ratio-lowering partial deleverage above LT or CF, forcing borrowers into extra liquidation penalty	Low	Fixed
L-10	[L-10] Liquidation-only price availability blocks partial borrower rescue actions	Low	Fixed
I-01	[I-01] Oracle creation does not cross-validate the main and guard routes' pair or inversion	Informational	Acknowledged
I-02	[I-02] Oracles cannot follow RedStone to a new adapter object without per-vault repointing	Informational	Acknowledged
I-03	[I-03] Oracle and vault do not share a price domain: scale_and_invert can emit rates the vault rejects	Informational	Acknowledged
I-04	[I-04] Last-admin guards can count an unusable role record	Informational	Fixed
I-05	[I-05] Oracle fixes take effect only through a relinked, version-migrated Vault upgrade	Informational	Fixed
I-06	[I-06] Oracle and Vaults cannot be published to Sui mainnet: manifests omit RedStone's system dependencies	Informational	Fixed

Low Severity Issues

L-01: An aging low route overrides a fresh route and liquidates healthy positions

Severity: Low	Impact: High	Likelihood: Low
Files: packages/oracle/sources/constants.move packages/oracle/sources/module/view.move packages/oracle/sources/module/admin.move packages/oracle/sources/providers/redstone_core.move packages/vaults/sources/utils/liquidate.move packages/vaults/sources/logic/liquidate.move	Status: Fixed	

Description:

For liquidation, `compute_liquidate` (`view.move:89-110`) accepts each route if it is under 7,200 seconds old and returns `rate::min(main, guard)`. The two routes' ages are never compared, and there is no bound on how far they may disagree. A route that froze at a dip therefore wins for up to two hours while the other route shows the recovery, even though the same price is rejected as stale for operate (600 seconds). Only liquidation is affected.

Example: BTC dips to $88,000, one route freezes there, BTC recovers to $100,000 and the other route follows. Liquidation prices at `min($88,000, $100,000)` = $88,000 for as long as the frozen route is under 7,200 seconds old.

The low quote does more than make positions liquidatable. The vault derives `max_tick` from it, and absorption runs on every liquidate call, even a zero-amount one (`logic/liquidate.move:81`). In the example, a borrower at 80% LTV is judged at 90.9%, above the 90% max limit, and absorbed wholesale, losing all equity. Positions between the liquidation

threshold and the limit are penalty-liquidated at the low price and the seized collateral (`1e30 / rate`) is oversized. The trigger is permissionless and free.

Reaching this state needs a provider freeze at a real-time dip (an old dip cannot be pushed later; Pyth's 60-second arrival-lag check rejects it), a recovery within two hours, and no refresh in between. Typically a RedStone relayer outage or a Pyth pusher outage. Normal RedStone lag is well under 1%, and no such event appears in the recent Pyth or RedStone data.

Recommendations:

Grade each observation by age alone: fresh when at most 600 seconds old. When both routes are usable for liquidation, prefer a fresh route over an aged one, and take `min()` only between routes of the same grade. The 2-hour window remains a fallback for when no fresh price exists. Operate behaviour is unchanged.

See also L-08. Deploy together with I-05.

Customer Response:

Fixed in 95ac3ec.

Fix Review:

Fix confirmed.

L-02: Single-feed oracle routes price a stablecoin leg at an implicit $1

Severity: Low	Impact: High	Likelihood: Low
Files: packages/oracle/sources/state/structs.move packages/oracle/sources/state/oracle.move packages/oracle/sources/oracle_calcs/rate.move packages/oracle/sources/module/view.move packages/oracle/sources/module/admin.move packages/vaults/sources/invokes/oracle.move	Status: Acknowledged	

Description:

A vault prices collateral in units of its debt token, so every market is a coin-to-coin pair. Nearly all feeds on Sui are USD-quoted. After this PR a route reads one feed only (`Source { feed_id, invert, source_type }`), and the guard is an alternative price for the same rate from the other provider, not a second leg. Passing a second Pyth price object aborts with 32001. The base branch composed up to four hops (`helpers/hop.move`), the PR removed that.

For a market such as USDC collateral / SUI debt, the only expressible route is SUI/USD inverted, which assumes USDC is worth exactly $1. The RedStone guard is USD-quoted too, so it adds no depeg protection. Markets with a direct feed (ETH/USDC via a direct cross feed, LST/SUI via a redemption-rate feed) are unaffected.

If the stablecoin depegs by more than `1 - CF` (about 20% at CF 80%): with stable collateral, positions become insolvent but stay "healthy" and cannot be liquidated, leaving bad debt for lenders, with stable debt, the debt is overvalued and healthy borrowers are liquidated. This requires governance to list such a market and a depeg larger than `1 - CF`, the largest historical USDC depeg was about 13%.

Recommendations:

- Decide per market: list only pairs with a direct feed, or reintroduce two-leg routes (`num / den`, each leg with its own feed and inversion, every leg checked for freshness at the same grade, and a route alive only if all its legs are alive).
- If a par assumption is accepted for a market, document it and keep CF and LT below 1 - max expected depeg. This bounds only the stable-collateral direction.
- Adopt a listing rule: every stablecoin-leg market must use the stablecoin's own feed or a direct cross, and its guard must be composed the same way or absent.

Customer Response:

Acknowledged.

L-03: A vault's oracle is not bound to the market's pair, and repointing takes effect instantly

Severity: Low	Impact: High	Likelihood: Low
Files: packages/vaults/sources/module/admin.move packages/vaults/sources/invokes/oracle.move packages/vaults/sources/state/auth.move packages/oracle/sources/state/oracle.move packages/oracle/sources/state/structs.move packages/oracle/sources/module/admin.move	Status: Acknowledged	

Description:

An `Oracle` stores `{ id, nonce, main, guard }` and nothing about which collateral/debt pair it prices. `init_vault_config<Col, Debt>` accepts any `&Oracle`, and `update_oracle` (`vaults/module/admin.move:384`) only rejects rebinding to the same oracle, then applies the new one immediately, with no pair, orientation or deviation check. At runtime the vault checks only `quote.oracle_id() == vault.oracle()`.

As a result, a wrong-pair or wrong-direction binding takes effect in the same transaction. A too-high rate lets anyone borrow beyond the collateral factor and drain the debt reserve (with an 80% CF, 1 COL borrows 0.79 DEBT under the correct oracle but 1,000 DEBT after repointing to a BTC-feed oracle). A too-low rate, such as a flipped `invert`, makes healthy positions liquidatable. Both are irreversible once borrowers or liquidators act. This requires a trusted-role mistake, such as a rushed repoint during a provider migration (I-02).

Recommendations:

- Record the market pair on the `Oracle` (base and quote `TypeName`), set at creation, and check it against the vault's `Col/Debt` in `init_vault_config` and `update_oracle`.
- Before any new or repointed oracle is submitted, dry-run a quote from it against an independent reference and check each route's `feed_id` and `invert`.
- Do not add a mandatory deviation bound or timelock to update_oracle, repointing is how a vault recovers when its oracle stops producing prices.

Customer Response:

Acknowledged.

L-04: The live shared Pyth Pro contract permits governance replay after source rotation

Severity: Low	Impact: Medium	Likelihood: Low
Files: packages/oracle/Move.toml packages/oracle/sources/providers/pyth_core.move	Status: Acknowledged	

Description:

Fluid reads Pyth prices from the live Pyth Pro-compatible package `0x55300367…e2c0` and its State `0x03719fae…39d`, both operated by Pyth. In that code, `pyth::governance::verify_vaa` returns a receipt that stores only the payload, digest and sequence, not the source it was verified against. At execution the only check is `sequence > last_executed_governance_sequence`, and `SetGovernanceDataSource` resets that counter. In one transaction, a caller can verify an old governance message and a rotation, execute the rotation, then execute the old message past the reset counter. Pyth has merged a fix upstream (source rotation aborts); the live Pro package does not include it.

The main risk is a liveness freeze: a replayed `SetDataSources` could stop price updates on the Pro State, so operate fails after 600 seconds on every market with a Pyth route and liquidation fails after 7,200 seconds on Pyth-only markets, until Pyth sends a new governance message.

The attack is not possible today. The Pro State has never executed governance (`last_executed_governance_sequence = 0`), and only Pyth can produce a rotation message. It needs Pyth to run governance on the Pro State and publish a rotation while the package is still unfixed. Once Pyth upgrades the live package, Fluid is protected without any change on its side.

Recommendations:

This is an upstream issue in Pyth's package and Fluid cannot fix it in code. Confirm with Pyth that the live Pro package will include the fix before any governance runs on the Pro State, and note the package version in the release checklist.

Customer Response:

Acknowledged.

L-05: Fixed 600 s operate age blocks borrowers on 24-hour-heartbeat RedStone feeds

Severity: Low	Impact: Medium	Likelihood: Low
Files: packages/oracle/sources/constants.move packages/oracle/sources/providers/redstone_core.move packages/oracle/sources/module/view.move packages/oracle/sources/state/oracle.move packages/oracle/sources/state/structs.move packages/vaults/sources/utils/operate.move packages/vaults/sources/invokes/oracle.move packages/vaults/sources/logic/operate.move	Status: Acknowledged	

Description:

The oracle's age limits are package-wide constants: `max_age_operate = 600` s and `max_age_liquidate = 7,200 s` (`constants.move:16-18`), and the Pyth confidence factors are fixed the same way. There is no per-oracle or per-feed setting, so changing a limit requires a package upgrade that applies to every market. `compute_operate` aborts if either route is stale; `compute_liquidate` drops a stale route and aborts only if none remains.

RedStone's Sui relayer writes some feeds only on a 0.5% deviation or 24-hour heartbeat (ETH and LBTC_FUNDAMENTAL by default); BTC, SUI and USDC have 480-second overrides. Borrowers cannot refresh a feed themselves: untrusted senders are limited to one write per 48 hours.

On a market with a long-heartbeat RedStone route, borrowing, withdrawing, top-ups and partial repayments would fail most of the day, while liquidation stays available through Pyth or, on a

RedStone-only route, for up to 7,200 seconds. A full repayment still works and no funds are lost. Only BTC is used via RedStone today, which fits with about 90 seconds of headroom. The risk is a future listing, or RedStone changing its relayer configuration off-chain.

Recommendations:

No Fluid code change removes this lockout safely. The fix is a RedStone write cadence at or below the operate age limit, and only RedStone controls that cadence, so Fluid has to enforce it through listing and operations.

- Only list RedStone feeds whose relayer heartbeat is below the 600-second operate age (with some margin), and monitor feed ages. For any other feed, ask RedStone for a per-feed heartbeat override first, as already done for BTC, SUI and USDC (480 s).
- Do not raise the age limit to fit a 24-hour heartbeat; the age check is what separates a quiet market from a dead relayer.

Customer Response:

Acknowledged.

L-06: Unrefreshed Pyth prices up to 600 seconds old let borrowers borrow at CF against a stale-high price

Severity: Low	Impact: Medium	Likelihood: Low
Files: packages/oracle/sources/constants.move packages/oracle/sources/providers/pyth_core.move packages/oracle/sources/module/view.move packages/vaults/sources/utils/operate.move	Status: Acknowledged	

Description:

Operate actions accept any Pyth price up to 600 seconds old (`constants.move:16`). The Pyth provider reads whatever price is already stored in the shared PriceInfoObject and checks only its age, arrival lag, validity and confidence (`pyth_core.move:60-96`), nothing requires the borrower to push a fresh update. On a single-route market, compute_operate (`view.move:75-83`) returns that price with no deviation or TWAP bound, and `check_if_position_safe` (`vaults/utils/operate.move:273-316`) prices the new borrow against it.

If a newer, lower signed price exists off-chain but has not been stored, a borrower can skip the refresh and borrow up to CF against the older, higher price. If the price has already fallen by more than 1 - CF within the 600s window, the position is undercollateralized from the moment it opens, and vault lenders absorb the shortfall. On an 80% CF vault this takes a fall of more than about 20% in ten minutes.

The window is narrow. An older print cannot overwrite a newer stored one, prints that arrive more than 60 seconds after publication are rejected, and any user can push the latest print; dual-route markets also need a RedStone relayer outage. Over 30 days, the largest fall within a 600s window on a user-pushed feed was 10.8%. Exposure grows on single-route Pyth markets with volatile collateral and a thin 1 - CF margin, or during a sustained outage of the sponsored Pyth pusher, Pythnet or Wormhole.

Recommendations:

Add a Pyth-only operate age of 60s (for example `max_age_operate_pyth`) and use it in the operate branch of the Pyth freshness check; keep RedStone and the shared 600s and 7,200s ages unchanged. A borrow or withdrawal on a long-tail feed then needs a Pyth push in the same transaction. The fix is partial: a price up to 60s old can still be used. Ship this only with the L-10 fix; otherwise repayments and top-ups on Pyth markets would be blocked after 61s.

Apply together with L-10 and L-07.

Customer Response:

Acknowledged.

L-07: Single-route Pyth markets let a liquidator use a stale dip price for up to 7,200 seconds

Severity: Low	Impact: Low	Likelihood: Medium
Files: packages/oracle/sources/constants.move packages/oracle/sources/providers/pyth_core.move packages/oracle/sources/module/view.move	Status: Acknowledged	

Description:

Liquidation reads accept a Pyth price up to 7,200 s old (constants.move:18). On a single-route market, compute_liquidate (view.move:89-95) uses that one route at this lax grade. Pyth is a pull oracle: if a dip price was pushed and the market then recovered off chain, the stored low price remains usable for liquidation until someone pushes the recovery. A liquidator can choose not to push it, so a borrower who is healthy at the current price is liquidated at the stale dip. In testing, a -14% dip let the liquidator pay 4,691.5 DEBT for collateral worth 5,455, a loss of about 764 DEBT to the borrower.

The borrower's only defence is to push the current price first. This is free and permissionless, and afterwards the liquidation aborts 6027, so the attack is a race between a bot and the borrower; it succeeds only after a significant unpushed recovery and when the borrower does not push first (the race was modelled by transaction order in unit tests). User-pushed long-tail feeds (NS, NAVX, IKA, BLUE, ...) need no outage: 30-day data shows NS rising 14.43% after 565 s without an update. Sponsored feeds need a pusher or Pythnet/Wormhole outage longer than 600 s. A -25% dip below the liquidation max limit allows a full absorb that took about 24,900 of 25,000 equity, though a recovery of that size was not seen in the live data.

Recommendations:

Add a Pyth-specific liquidation age of 600 seconds and use it in the liquidation branch of the Pyth age check. Apply it to every Pyth route read at liquidation grade, including main and guard

routes on dual-route markets. Keep RedStone at 7,200 seconds; do not lower the global liquidation age, because that would block RedStone liquidations on feeds with 24-hour heartbeats. The fix is partial: a dip up to 600 seconds old can still be used.

Apply together with L-06 (same line). Deploy together with I-05.

Customer Response:

Acknowledged.

L-08: A late or wide-confidence Pyth update can make the shared price object temporarily unreadable

Severity: Low	Impact: Low	Likelihood: Low
Files: packages/oracle/sources/providers/pyth_core.move packages/oracle/sources/module/view.move packages/oracle/sources/constants.move	Status: Acknowledged	

Description:

Fluid rejects a Pyth reading that reached the chain more than 60s after its publish time (`pyth_core.move:68-71; constants.move:23`). It also rejects a reading whose confidence interval is wider than 2% of the price for operate or 4% for liquidation (`pyth_core.move:87-94; constants.move:25-27`). These checks run only when Fluid reads the price. Pyth's `update_single_price_feed` runs neither check; it stores any signed print that is newer than the one already stored.

So anyone can store a genuine print that is newer but that Fluid will not accept. The shared PriceInfoObject then stays unreadable for Fluid until a fresh in-bound print is stored. Until then, a caller who does not push their own update in the same PTB cannot operate on any market that needs that Pyth route. Liquidation also aborts on Pyth-only markets (`view.move:93-94` and `105-108`). One bad store affects every market that shares the PriceInfoObject. No funds move.

The late case needs no attacker. On the pinned deployment it happened 150 times in 30 days and blocked liquidation reads on long-tail feeds for up to almost 2 hours. On sponsored feeds it needs a pusher outage. The impact is Low because anyone can restore reads for free (`base_update_fee` is 0) by pushing a fresh in-bound print, when one exists. It becomes real keeper downtime only if Fluid's own liquidation or rescue tooling reads without pushing first.

Recommendations:

No oracle code change is recommended, Fluid cannot add these checks to Pyth's permissionless update. Do not use a last-known-good cache, which would serve outdated prices during crashes. Do not accept only prints pushed in the same transaction either; that would reopen the historical-print attack. For every unsponsored Pyth feed behind a market, run a Fluid pusher (or get sponsorship) with a cadence of about 30s or less, and make this a listing rule. Keepers, liquidators, rescue bots and front ends should push the latest Pyth print in the same PTB as every Pyth-dependent operate or liquidation. This is a partial fix: while no tight print exists, Pyth-only operate and liquidation still fail closed by design.

See also L-01.

Customer Response:

Acknowledged.

L-09: Health gates reject ratio-lowering partial deleverage above LT or CF, forcing borrowers into extra liquidation penalty

Severity: Low	Impact: Low	Likelihood: Low
Files: packages/vaults/sources/utils/operate.move packages/vaults/sources/logic/operate.move	Status: Fixed.	

Description:

The health gate in packages/vaults/sources/utils/operate.move (lines 270-336) requires the position to end below a threshold even when the operation strictly lowers its ratio.

- CF branch (primary issue). Any operation with a withdrawal or a debt increase is checked against the collateral factor (lines 293-298). A withdraw-funded payback that lowers both ratio and net debt, but still ends above CF, aborts `6011` (line 329), even though the code's own comment (lines 325-327) says such a deleverage should be allowed because it makes the position safer. A borrower who already holds the debt tokens can make a pure payback of the same 19.6 DEBT, which is accepted, so the CF-branch loss applies only when the repayment is funded by withdrawing collateral.

- LT branch. A pure deposit or payback must land at or below the liquidation threshold, otherwise it aborts `6063`. This is documented as intended (line 328), but combined with the CF branch it leaves a borrower no way to partially deleverage.

Because of these gates, a borrower between CF/LT and the liquidation max limit who has no flash-loan or swap access, or holds a thin pair, cannot partially rescue the position, so it is liquidated on the larger debt and pays extra penalty instead of only swap costs. With a 5% penalty on a 1,000 COL / 790 DEBT position and a price drop to 0.93, the measured extra penalty is +0.393 DEBT for a rejected 2-DEBT repay (LT branch) and +1.405 DEBT for a rejected 20 COL / 19.6 DEBT deleverage (CF branch). No principal is lost, and borrowers with flash-loan access can avoid most of the loss by fully crossing the threshold.

EVM VaultT1 has no LT gate and skips the CF check when the tick fell and net debt did not grow (main.sol lines 416-418); this comparison is from reading main.sol, not execution. The loss grows with position size and penalty, and is largest on long-tail pairs with no flash-loan or swap liquidity.

Recommendations:

In `logic/operate.move`, keep the existing debt > 0 condition and skip the post-operation health check (and its quote read) in either case: (1) the tick strictly fell, the operation includes a repayment, and net debt did not rise beyond a rounding tolerance of old × (1 + 1e-9) + 1; (2) the operation is a pure deposit or repayment, with no withdrawal and no borrow. All other operations keep the CF check, the withdrawal-gap check and the cap requirement. The LT branch can then be removed.

Apply together with L-10.

Customer Response:

Fixed in 95ac3ec.

Fix Review:

Fix confirmed.

L-10: Liquidation-only price availability blocks partial borrower rescue actions

Severity: Low	Impact: Low	Likelihood: Low
Files: packages/vaults/sources/utils/operate.move packages/vaults/sources/invokes/oracle.move packages/vaults/sources/logic/operate.move packages/oracle/sources/module/view.move packages/oracle/sources/constants.move	Status: Fixed.	

Description:

The oracle has two freshness limits: 600s for operate and 7,200s for liquidation (`constants.move:16-18`). The vault uses the operate rate for withdrawals and borrows and the liquidation rate for deposits and repayments, but it chooses too late. `check_if_position_safe (vaults/utils/operate.move:283-286)` always calls get_both_exchange_rate, which reads both rates from the quote, and only then picks the one it needs. Every operation that leaves debt on the position goes through this check.

A full quote (`view::get_both_exchange_rate`) can only be built while every route is under 600s old. Once any route is 601–7,200s old, the full quote aborts (`30003/32003`), a liquidation-only quote is rejected by the vault (`31001/32004`), and the deposit or repayment fails. Liquidation of the same position, which needs only the liquidation-only quote, keeps working.

A borrower who wants to add collateral or partially repay can't, in exactly the window where they can be liquidated. A full repayment still works, because a zero-debt position skips the health check. On Pyth routes the borrower can usually push a fresh price in the same transaction. On RedStone routes they can't: the live adapter allows untrusted senders one write per 48 hours, so the window lasts until RedStone's relayer writes again. For feeds on a 24-hour heartbeat (see L-05), this would be the normal state rather than an outage.

Recommendations:

In `check_if_position_safe`, read only the rate the branch needs: `rate_operate()` for withdraw/borrow, `rate_liquidate()` for deposit/payback. This is never stricter than today, since every price valid for operate is also valid for liquidation. The frontend must then build a liquidate-only quote for deposits and repayments.

Apply together with L-09. See also L-05.

Customer Response:

Fixed in 95ac3ec.

Fix Review:

Fix confirmed.

Informational Severity Issues

I-01: Oracle creation does not cross-validate the main and guard routes' pair or inversion

Files:

packages/oracle/sources/module/admin.move
packages/oracle/sources/state/structs.move
packages/oracle/sources/state/oracle.move
packages/oracle/sources/module/view.move

Description:

An Oracle is meant to hold two independent routes for the same rate, and every quote takes rate::min(main, guard) (view.move, lines 75-110). admin::init_oracle_config only checks that the nonce is free, that each feed ID is valid, and that the two routes use different providers. It does not check that both routes quote the same pair or use the same invert direction. The Oracle has no setter, so a wrong route cannot be fixed in place. As a result, a mis-inverted guard, or a guard on a cheaper asset, becomes the lower route and prices both operate and liquidation, which undervalues collateral and makes healthy positions liquidatable; liquidations already executed cannot be undone. Harm requires two trusted-role mistakes: an oracle authorizer creating a mismatched or inverted guard route, and a vault authority adopting that oracle through init_vault_config or update_oracle. Most cases show up in get_both or a dry-run quote before adoption, so they can be caught before any harm. Harm was shown at quote level only. A guard on a correlated but different asset that reads higher than main at creation passes every pre-adoption check, then silently prices the market once the assets diverge.

Recommendations:

Before the first publish, change `init_oracle_config` in place to take the RedStone adapter, the Pyth price object and the clock. Read every route at operate grade, including main when there is no guard, and abort unless main and guard agree within a fixed 500 bps of the higher rate. Accept opposite invert flags, because providers may quote a pair in opposite orientations. The check is partial: a correlated asset trading within tolerance (for example WBTC against BTC) still passes.

Apply together with I-02; see also L-03.

Customer Response:

Acknowledged.

I-02: Oracles cannot follow RedStone to a new adapter object without per-vault repointing

Files:

packages/oracle/sources/state/structs.move
packages/oracle/sources/state/oracle.move
packages/oracle/sources/providers/redstone_core.move
packages/oracle/sources/module/admin.move
packages/vaults/sources/module/admin.move
packages/vaults/sources/invokes/oracle.move

Description:

Each RedStone route pins its PriceAdapter object ID ({{SourceType::Redstone { adapter_id }}}). Only oracle::create sets it, and no admin function can change it. redstone_core::price_data asserts the adapter matches. That check correctly blocks callers from substituting their own adapter, but it also stops an existing Oracle from following RedStone to a new adapter object. The new adapter aborts with 30001, including inside try_read_source, so there is no fallback to the guard route. The old adapter aborts with 30003 once it goes stale. As a result, if RedStone switched to a new adapter object and stopped writing to the old one, operate quotes on every RedStone-route vault would fail 600 s after the last write, and borrowing, withdrawal, partial repayment and top-up would all revert. Dual-route markets can still liquidate through Pyth if the caller passes the old adapter, while RedStone-only markets also lose liquidation after 7,200 s.

Recovery requires a new Oracle per market and a vaults::admin::update_oracle call for every affected vault. No funds are lost directly, and no untrusted party can cause this. Because RedStone currently migrates in place and its readers do not check versions, a replacement adapter object is unlikely. If it happens, the result is a temporary freeze of core borrower actions. RedStone announces adapter migrations in advance, so governance can create replacement oracles and repoint vaults before the cutover; the impact is therefore Low and the finding is Informational. On RedStone-only markets, a sharp price move before governance repoints every vault can leave unhealthy positions unliquidated. The repoint is also instant and unvalidated (L-03).

Recommendations:

Keep the adapter-identity check. Add an AdminRole-only function that changes only a RedStone route's `adapter_id` and keeps the feed, invert, route slot and the Oracle's ID, so vault bindings survive without `update_oracle`. In the same transaction, read the new adapter at operate grade so that a stale or unwritten adapter, an unknown feed or a zero price aborts the change. The fix is partial: the outage until the rotation transaction lands remains. Do not add a delay; after the cutover, any delay is extra downtime.

See also L-03. Apply together with L-02 and I-01.

Customer Response:

Acknowledged.

I-03: Oracle and vault do not share a price domain: scale_and_invert can emit rates the vault rejects

Files:

packages/oracle/sources/oracle_calcs/rate.move
packages/vaults/sources/utils/liquidate.move
packages/vaults/sources/logic/liquidate.move
packages/vaults/sources/utils/operate.move
packages/vaults/sources/constants.move
packages/library/sources/tick_math.move

Description:

scale_and_invert (rate.move lines 81-93) does not bound its output. An inverted route can emit rates up to 1e30, but the vault rejects any rate above max_oracle_price (1e24) with abort 6031. Below about 26,728 (81% LT, parity exchange prices), the tick ratio falls under MIN_RATIOX48. Liquidation and absorb then abort with 14004. Code comments in liquidate.move and operate.move acknowledge both edges. As a result, a vault whose rate crosses either edge freezes: above the cap, top-ups, partial repay/withdraw and liquidation abort, though full Max repay still works and positions stay overcollateralized, so no loss is established. Among live feeds, only BABYDOGE/USD can reach either edge, the lower one after a further ~97% fall and the upper one only through an inverted route that the linked Pyth deployment does not carry, and u64 totals cap such a vault at about $18. Nothing at listing time rejects a pair whose rate sits near either edge, such as Pyth exponent -11 or below, or a non-inverted rate above 1e9.

Recommendations:

Keep the vault's `max_oracle_price` and tick lower edge as the only authority on the valid price domain, and document that the oracle's output is intentionally unbounded. Before `init_vault_config`, before every `update_oracle` and whenever the liquidation threshold changes, run an off-chain listing check that requires the live rate to sit a documented margin inside both edges.

Do not clamp or reject rates above 1e24 in `scale_and_invert`. Rejecting the final quote blocks the full Max repay exit, and clamping can liquidate healthy positions.

See also L-09.

Customer Response:

Acknowledged.

I-04: Last-admin guards can count an unusable role record

Files:

packages/oracle/sources/state/auth.move
packages/oracle/sources/module/admin.move
packages/oracle/sources/state/registry.move

Description:

The Oracle's last-admin guard counts stored AdminRole records, not usable signers. auth::grant is one-step and accepts any address, including @0x0, and increments admin_count (packages/oracle/sources/state/auth.move#L35, packages/oracle/sources/module/admin.move#L39). An admin can grant AdminRole to @0x0 or a mistyped address, which raises the count to two, and then revoke the only real admin. The guard allows this. Only a trusted admin's handoff mistake triggers it.

As a result, nobody can grant or revoke roles (update_admin/update_auth) or call registry migrate until a package upgrade, which is possible while the publisher holds the UpgradeCap. Prices, quotes and oracle creation (gated by AuthorizeRole) keep working. The reproduction assumes, rather than proves, that @0x0 or a typo address has no usable signer.

update_admin and update_auth also skip the registry version check. Once fluid_oracle is published, the original package's setters stay callable on the shared Auth and can bypass any later role-policy fix. A version gate added before the first publish avoids this. If the UpgradeCap is burned or made immutable, the lockout is permanent, and a compromised or lost AuthorizeRole key can never be revoked.

Recommendations:

Reject @0x0 when granting either role. Make the admin grant two-step: store a pending admin and increase `admin_count` only when that address accepts in a transaction it signs; let current admins cancel pending grants. Add the OracleRegistry version check to `update_admin`, `update_auth` and the new accept and cancel functions.

Ship this in the first publish of fluid_oracle; once the one-step setters are live, they stay callable on the shared Auth and bypass a later fix.

Customer Response:

Fixed in 95ac3ec.

Fix Review:

Fix confirmed.

I-05: Oracle fixes take effect only through a relinked, version-migrated Vault upgrade

Files:

packages/oracle/sources/module/view.move
packages/oracle/sources/state/oracle.move
packages/oracle/sources/state/registry.move
packages/oracle/sources/module/admin.move
packages/vaults/sources/invokes/oracle.move
packages/vaults/sources/state/registry.move
packages/vaults/sources/module/admin.move
packages/vaults/Move.toml

Description:

The fluid_oracle quote functions (get_both_exchange_rate, get_exchange_rate_operate, get_exchange_rate_liquidate) never read the OracleRegistry, so they have no version check. The vault checks only quote.oracle_id. Because Sui keeps old package versions callable, an oracle-only upgrade does not stop a caller from using the old view in the same PTB as the vault.

A Vaults upgrade closes this only if it relinks fluid_oracle (old direct calls then fail with InvalidLinkage), bumps the Vaults version and runs migrate_vault_registry. Without the last two steps, the old Vaults package still accepts old quotes. This causes no harm in the audited code, since Oracle configurations are immutable, but it is a deployment caveat for the L-01, L-06, L-07 and L-08 fixes and for Pyth or RedStone pin changes. The upgrade runs used simplified stand-in packages. If such a fix ships without the full Vaults upgrade, the old pricing stays usable; for L-01, healthy positions keep being liquidated after the fix is believed live.

Recommendations:

Before the oracle package is first published, pass the OracleRegistry to every quote-producing function. Route all of them through one private helper that rejects only outdated code (`registry.version <= constants::version()`). Put fixes into the existing quote functions instead of adding fixed versions next to them. Do not reuse the exact-equality assert_version; it would block new code before migration and stop liquidations during that window. If the oracle is published without this gate, ship every oracle-side fix as part of the full Vaults upgrade.

See also L-01, L-06 and L-07.

Customer Response:

Fixed in 95ac3ec.

Fix Review:

Fix confirmed.

I-06: Oracle and Vaults cannot be published to Sui mainnet: manifests omit RedStone's system dependencies

Files:

packages/oracle/Move.toml
packages/oracle/Move.lock
packages/vaults/Move.toml
packages/vaults/Move.lock

Description:

A Sui publish transaction must list every package in the transitive dependency closure. The mainnet RedStone package Fluid links against (0xbd6c…6ac3, v4) links 0x1 v19, 0x2 v45, 0x3 (SuiSystem) v26 and 0xb (Bridge) v6. fluid_oracle declares only fluid_library, redstone_price_adapter and pyth, and neither lock file has SuiSystem or Bridge entries. Sui CLI 1.80.1 does not add them implicitly, so the publish lists only [0x1, Pyth 0x5530…, 0x2, Wormhole 0x99de…, RedStone 0xbd6c…] and the validator rejects it with PublishUpgradeMissingDependency.

This holds for both the audited and the final RedStone manifest pin. fluid_vaults depends on the oracle through a local path and inherits the same gap (inferred; not dry-run separately). As a result, the Oracle and the Vaults as committed cannot be deployed to mainnet with the current CLI, although a CLI version that still injects implicit system dependencies might publish them successfully. The failure is loud and happens before launch, so no funds or state are at risk. This becomes material only if operators work around the failure by republishing or re-pinning RedStone or Pyth (e.g. `--with-unpublished-dependencies`). A republished RedStone defines a different PriceAdapter type that the live adapter 0x22794c3a… cannot satisfy, so the deployed Oracle would either fail to read prices or run unreviewed provider code.

Recommendations:

Add `sui_system = { system = "sui_system" }` and `bridge = { system = "bridge" }` to the Oracle manifest's [dependencies], and remove its legacy [addresses] section (fluid_oracle = "0x0"). Move the Vaults manifest to the new format as well, since a legacy-format package cannot depend on a new-format one: remove fluid_vaults = "0x0" and add

`redstone_price_adapter` and `pyth` as direct dependencies with the same pins as the Oracle. Then regenerate and review both lock files.

This must ship before the first mainnet publish; never use `--with-unpublished-dependencies` for RedStone or Pyth.

Customer Response:

Fixed in 95ac3ec.

Fix Review:

Fix confirmed.

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. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.