

Security Assessment Report



Fluid Timelock

October 2026

Prepared for Fluid / Mysten Labs

certora

Table of Contents

Project Summary **4**
- Project Scope 4
- Project Overview 4
- Protocol Overview 6

Assessment Methodology **7**

Threat and Security Overview **8**
- 1. System overview 8
- 2. Actors and trust assumptions 9
- 3. Security properties and results 10
- 4. Assumptions not enforced on chain 11
- 5. Areas reviewed 12
- 6. Methodology and limitations 12

Findings Summary 14

Severity Matrix 14

Detailed Findings **15**

Medium Severity Issues 18
- M-01: A guardian seated as the sole AuthCancellerRole holder cannot be removed and can veto timelocked governance indefinitely 18
- M-02: A rogue guardian can make itself unremovable and freeze liquidity during a liquidity upgrade 21

Low Severity Issues 24
- L-01: Package upgrades can skip the 12h timelock after setup if an UpgradeCap is never stored 24
- L-02: An address holding both GuardianRole and AdminCancellerRole can cancel its own removal and cannot be removed 26
- L-03: Near-zero grace period makes every later timelock operation, including the fix, executable only in a one-millisecond window 27
- L-04: An unpause approved for an earlier pause can lift a later emergency pause without a new review window 28
- L-05: A rogue guardian's pause can freeze withdrawals, repayments and vault liquidations for at least about 24 hours, not the documented 12 hours 30
- L-06: Migrating liquidity before the lending and vaults relinks commit halts lending and vault user operations, liquidations included, for at least 12 hours 32

Informational Severity Issues 34
- I-01: A TimelockAdminRole holder can raise min_delay_ms to 60 days, and the raise can only be reverted at the raised delay 34
- I-02: lending_rewards update_auth revoke aborts with a framework error for authorities granted after init 36
- I-03: Oracle timelock cap mint and disable skip the version check and emit no events, unlike the other layers 37
- I-04: Two new vaults error codes reuse existing abort codes 6078 and 6079 38

I-05: Timelock comments and docs misstate stray TimelockCap recovery, omit the DisableTimelock one-way door, and mislabel the migrate gate as permissionless..... 39
I-06: Most new timelock adapter paths, the migrator-role grants and the oracle update_admin version check have no tests..... 41
I-07: MintUserCap can operate without the vault's collateral checks..... 43
I-08: Retired timelock package versions can still invoke public state-mutating primitives..... 44
I-09: RemoveGuardian can revoke AuthCancellerRole from non-guardians without guardian veto..... 45
Disclaimer..... 46
About Certora..... 46

Project Summary

Project Scope

Project Name	Repository Link	Initial Commit Hash	Final Commit Hash	Platform
Fluid Timelock	https://github.com/Instadapp/fluid-packages-sui/pull/38	742ea21f7cb20cccff141c3cbf354a088fbfa778	a8da31c9d1caca9ed1734cf731cb3f3cc5afecad	Sui/Move

Project Overview

This document describes the security review of **Fluid Timelock**. The work was undertaken from **October 21st, 2026** to **October 28th, 2026**.

The following contract list is included in our scope:

packages/lending/sources/errors.move
packages/lending/sources/helpers/asserts.move
packages/lending/sources/module/admin.move
packages/lending/sources/state/auth.move
packages/lending/sources/state/timelock_cap.move
packages/lending/sources/types/events.move
packages/lending_rewards/sources/errors.move
packages/lending_rewards/sources/helpers/utils.move
packages/lending_rewards/sources/module/admin.move
packages/lending_rewards/sources/state/rewards_auth.move
packages/lending_rewards/sources/state/timelock_cap.move
packages/lending_rewards/sources/types/events.move
packages/liquidity/sources/errors.move
packages/liquidity/sources/module/admin.move
packages/liquidity/sources/state/auth.move

packages/liquidity/sources/state/registry.move
packages/liquidity/sources/state/timelock_cap.move
packages/liquidity/sources/types/events.move
packages/liquidity/sources/types/pause.move
packages/liquidity/sources/types/structs.move
packages/oracle/sources/errors.move
packages/oracle/sources/module/admin.move
packages/oracle/sources/state/auth.move
packages/oracle/sources/state/timelock_cap.move
packages/timelock/sources/module/guardian_gov.move
packages/timelock/sources/module/lending_gov.move
packages/timelock/sources/module/liquidity_gov.move
packages/timelock/sources/module/oracle_gov.move
packages/timelock/sources/module/rewards_gov.move
packages/timelock/sources/module/role_gov.move
packages/timelock/sources/module/timelock.move
packages/timelock/sources/module/upgrade_gov.move
packages/timelock/sources/module/vaults_gov.move
packages/timelock/sources/state/auth.move
packages/timelock/sources/state/cap_store.move
packages/timelock/sources/types/events.move
packages/timelock/sources/utils/constants.move
packages/timelock/sources/utils/errors.move
packages/timelock/sources/utils/timelock_ops.move
packages/vaults/sources/errors.move
packages/vaults/sources/module/admin.move
packages/vaults/sources/state/auth.move
packages/vaults/sources/state/position.move
packages/vaults/sources/state/registry.move
packages/vaults/sources/state/timelock_cap.move
packages/vaults/sources/types/events.move

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

The PR puts every governance and config action in the Fluid protocol packages behind a timelock. A new `fluid_timelock` package holds two timelocks, a role store and a cap store. Once a layer's `TimelockCap` is recorded, its gated admin functions need both the sender check and a ticket from that cap. The timelock module is a reduced fork of OpenZeppelin contracts-sui v1.5.1.

- `fluid_timelock`: an admin timelock with a 12h floor and an auth timelock with a 6h floor. Ready operations expire after a 7d grace period.
- **Roles** in `fluid_timelock::auth`: admin and auth proposer, admin and auth canceller, executor, timelock admin, guardian, migrate.
- `CapStore`: custody of the five `TimelockCap`s and the `UpgradeCap`s, reachable only from internal `execute_*` paths.
- **Per-layer gating**: liquidity, vaults, lending, lending_rewards and oracle take `tl: Option<TimelockTicket>` on gated functions, wrapped by `schedule_x` / `execute_x` pairs in `*_gov` modules.
- **Guardians**: instant liquidity pauses, auth-timelock cancels and admin-timelock cancels through `guardian_gov::cancel_admin`. They cannot cancel `RemoveGuardian`, which strips guardian rights here and in liquidity.
- **Setup**: the publisher's `SetupCap` grants roles instantly, then `finalize_roles`, `enable_all_timelocks` and `complete_setup` burns it. Roles then change only through 12h actions.
- **Upgrades**: `upgrade_gov` authorizes upgrades on the admin timelock. Each protocol's `migration` stays outside the timelock, gated by a migrate role.

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

1. System overview

PR #38 puts the Fluid protocol's governance and configuration actions on Sui behind a timelock. Before the PR, a privileged caller could change a protocol setting immediately. After it, most of those changes must first be scheduled publicly, wait out a fixed delay and then be executed. Users and the team can see every change before it takes effect.

The new `fluid_timelock` package contains:

- **Two timelocks.** The admin timelock has a 12-hour minimum delay and handles role changes, upgrades, unpauses and disabling the timelock. The auth timelock has a 6-hour minimum delay and handles routine configuration. A ready operation can be executed for 7 days before it expires.
- **A role store.** It holds the proposer, canceller, executor, timelock admin, guardian and migrator roles.
- **A cap store.** It holds the capability for each of the five protocol layers (liquidity, vaults, lending, lending rewards, oracle) and the package upgrade capabilities. These can only be used by executing an operation once its delay has passed.

Each layer keeps its existing permission checks. When its timelock is switched on, gated calls also need a one-time authorization from the timelock. The layers launch with the timelock off and behave as before. The timelock is then switched on through a one-way setup sequence. Each package's data migration (`migrate`) deliberately stays outside the timelock and is controlled by a dedicated migrator role.

2. Actors and trust assumptions

Actor	Trust level	What it can do
Main multisig	Trusted	Holds every role at deployment. In the intended setup it proposes admin actions, can cancel any operation and administers the timelock settings.
Ops multisig	Trusted	Proposes routine configuration changes on the 6-hour timelock and runs migrations, which take effect immediately.
Executor	Trusted for timing only	Executes ready operations and chooses when and in what order within the 7-day window. It cannot change what was approved. It needs the top permission tier in every layer it executes into.
Guardian	Semi-trusted; assumed possibly compromised	Pauses liquidity immediately and cancels pending operations. Its one limit is that it cannot cancel its own removal, which is what is meant to cap how long a rogue guardian can obstruct governance.
Migrator	Trusted	Decides when users move to a new package version by running `migrate`, which takes effect immediately and cannot be undone.
Deployer (during setup)	Trusted	Assigns roles instantly until setup is completed, after which this power is permanently destroyed.
Users and other callers	Untrusted	Can call any public function in any package version and can see every scheduled operation.

3. Security properties and results

Property the design aims for	Result	Related findings
Every gated change waits its full delay before it can take effect	Holds for every timelocked action. Package upgrades escape the delay if an upgrade capability is never handed to the timelock during setup.	L-01
An executed operation does exactly what was approved, on the object that was approved	Holds. The exception is unpause approvals, which are not tied to a specific pause, so an old approval can lift a newer emergency pause.	L-04
Only the timelock can authorize gated calls, and its capabilities cannot leak	Holds for capabilities kept in the cap store. The code comments overstate how a capability left outside the store can be recovered.	I-05
With the timelock off, every layer behaves as it did before the PR	Holds, apart from an intended change to lending rewards permissions, which left one revocation path failing with an unclear error.	I-02
A rogue guardian can delay governance by at most 12 hours and can always be removed	**Does not hold.** The guardian can become unremovable through some role setups or during a liquidity upgrade, and its pauses can last about twice as long as documented.	M-01, M-02, L-02, L-05

Property the design aims for	Result	Related findings
Timelock settings cannot leave governance unusable	Does not hold for extreme values. A near-zero grace period or a maximal minimum delay can make later operations, including the fix, practically impossible to run.	L-03, I-01
Upgrades and migrations do not interrupt users or governance	Not enforced. Running a migration too early can halt lending and vaults for at least 12 hours, or block guardian removal.	M-02, L-06
New functions follow the codebase's existing conventions: version checks, events and unique error codes	Minor gaps in the oracle and vaults packages.	I-03, I-04
Documentation and tests reflect the shipped behaviour	Several comments are out of date, and much of the new timelock code has no tests.	I-05, I-06

4. Assumptions not enforced on chain

The security of the system also depends on operational choices the code does not check:

- All protocol capabilities and upgrade capabilities are actually handed to the timelock before setup is completed.
- Roles are assigned as intended. In particular, guardians are separate from the main multisig, removal-sensitive roles are not combined on one address, and executors hold the required permissions in every layer.
- Every dependent package is relinked before liquidity is migrated during an upgrade.
- The Sui framework behaves as documented.

5. Areas reviewed

Area	What was examined	Outcome
Access control and roles	Whether any gated action can skip its delay, the order of permission checks, guardian powers and removal, protection of the last role holder, and capabilities held outside the timelock	No way to skip the delay. Guardian removal issues found.
Fund safety	Whether the PR changes any user-facing maths or gives an unprivileged caller a path to funds	User maths is unchanged and no path to funds exists. Funds can be temporarily frozen during upgrades or pauses.
Upgrades and dependencies	Custody of upgrade capabilities, ordering of package upgrades and migrations, version checks across linked packages	Upgrade ordering and setup gaps found.
Operational and economic	Public visibility of the queue, executor timing, delay and grace-period settings	Configuration edge cases found. Public visibility of the queue is accepted as a design trade-off.
Consistency with the existing code	The same changes across all five layers, error codes, events, documentation and tests	Minor inconsistencies found.

6. Methodology and limitations

This was a manual review of commit `742ea21`. No fuzzing or formal verification was performed. Real 12-hour and 6-hour delays were not waited out, oracle prices were simulated, and network congestion was not tested.

Findings Summary

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

Severity	Discovered	Acknowledged	Fixed
Critical	-	-	-
High	-	-	-
Medium	2	0	2
Low	6	4	2
Informational	9	2	7
Total	17	6	11

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
M-01	A guardian seated as the sole AuthCancellerRole holder cannot be removed and can veto timelocked governance indefinitely	Medium	Fixed
M-02	A rogue guardian can make itself unremovable and freeze liquidity during a liquidity upgrade	Medium	Fixed
L-01	Package upgrades can skip the 12h timelock after setup if an UpgradeCap is never stored	Low	Acknowledged
L-02	An address holding both GuardianRole and AdminCancellerRole can cancel its own removal and cannot be removed	Low	Fixed
L-03	Near-zero grace period makes every later timelock operation, including the fix, executable only in a one-millisecond window	Low	Fixed
L-04	An unpause approved for an earlier pause can lift a later emergency pause without a new review window	Low	Acknowledged

L-05	A rogue guardian's pause can freeze withdrawals, repayments and vault liquidations for at least about 24 hours, not the documented 12 hours	Low	Acknowledged
L-06	Migrating liquidity before the lending and vaults relinks commit halts lending and vault user operations, liquidations included, for at least 12 hours	Low	Acknowledged
I-01	A TimelockAdminRole holder can raise min_delay_ms to 60 days, and the raise can only be reverted at the raised delay	Informational	Acknowledged
I-02	lending_rewards update_auth revoke aborts with a framework error for authorities granted after init	Informational	Fixed
I-03	Oracle timelock cap mint and disable skip the version check and emit no events, unlike the other layers	Informational	Fixed
I-04	Two new vaults error codes reuse existing abort codes 6078 and 6079	Informational	Fixed
I-05	Timelock comments and docs misstate stray TimelockCap recovery, omit the DisableTimelock one-way door, and mislabel the migrate gate as permissionless	Informational	Fixed
I-06	Most new timelock adapter paths, the migrator-role grants and the oracle update_admin version check have no tests	Informational	Fixed

I-07	MintUserCap allows minting UserCaps for uninstantiated vaults outside atomic flow	Informational	Acknowledged
I-08	Retired timelock package versions can still invoke public state-mutating primitives	Informational	Fixed
I-09	RemoveGuardian can revoke AuthCancellerRole from non-guardians without guardian veto	Informational	Fixed

Medium Severity Issues

M-01: A guardian seated as the sole AuthCancellerRole holder cannot be removed and can veto timelocked governance indefinitely

Severity: Medium	Impact: High	Likelihood: Low
Files: packages/timelock/sources/module/guardian_gov.move packages/timelock/sources/state/auth.move packages/timelock/sources/module/role_gov.move packages/timelock/sources/module/timelock.move	Status: Fixed	

Description:

Affected packages: `timelock`

`RemoveGuardian` is the one admin-timelock action a guardian cannot cancel. The module documentation states that it strips every guardian role, so "a rogue guardian can block other actions for at most 12h", and that "the last guardian is removable (exempt from `auth`'s last-holder guard) so it cannot block its successor" (guardian_gov.move#L5-L7). The code does not keep this promise when the guardian is the only holder of `AuthCancellerRole`.

`execute_remove_guardian` first revokes `GuardianRole`, then revokes `AuthCancellerRole` through the same `auth::revoke_role` (guardian_gov.move#L99-L102). `revoke_role` refuses to remove the last holder of a role and exempts only `GuardianRole` (auth.move#L198-L207). If the guardian is the sole `AuthCancellerRole` holder, the second revoke aborts with `last_holder_not_removable` (4005). The abort is atomic, so the `GuardianRole` revoke and the `Done` marker set by `execute` (timelock.move#L557) are rolled back. The guardian keeps both roles and the `RemoveGuardian` operation stays pending.

The guardian cannot create this state by itself. Governance creates it through role seating, and nothing in the code prevents that:

- During bootstrap, the `SetupCap` holder grants `AuthCancellerRole` to the guardian and then instantly revokes it from every other holder with `role_gov::revoke_role` (role_gov.move#L63-L66). `finalize_roles` only checks that the role has at least one holder (auth.move#L147).
- After setup, governance executes `RevokeRole<AuthCancellerRole>` against every non-guardian holder. `assert_not_guardian` only refuses guardian targets (role_gov.move#L214-L220), so revoking the Main and Ops multisigs succeeds.

The intended seating gives `AuthCancellerRole` to the Main and Ops multisigs as well as to the guardians, so this needs governance to depart from that seating. It is still an easy configuration to reach, since the module documents that guardians hold `AuthCancellerRole` (guardian_gov.move#L2).

Once in this state, the only way out is to give `AuthCancellerRole` to someone else first. That is a `GrantRole<AuthCancellerRole>` operation on the admin timelock, with a delay of at least 12h (timelock_ops.move#L40). `cancel_admin` lets any `GuardianRole` holder cancel every admin operation except `RemoveGuardian` (guardian_gov.move#L48-L51), and `cancel_internal` cancels any pending operation (timelock.move#L570-L586). A rogue guardian cancels each repair grant before it becomes executable, however often governance re-schedules it. The other paths are closed. `RevokeRole<AuthCancellerRole>` against a guardian aborts with `use_remove_guardian` (3002) at both schedule and execute (role_gov.move#L178, role_gov.move#L208). The instant bootstrap paths are closed after `finalize_roles`. A package upgrade is no escape either, because `AuthorizeUpgrade` is also an admin-timelock operation the guardian can cancel.

The consequence is that governance permanently loses the ability to remove a rogue guardian, and the guardian keeps an unbounded veto instead of the documented 12h one. Through `cancel_admin` it can cancel every admin-timelock operation: role grants and revokes, `AuthorizeUpgrade`, `DisableTimelock`, timelock delay updates and the liquidity unpauses (`UnpauseGlobal`, `UnpauseToken`, `UnpauseUser`, see liquidity_gov.move#L596).

Recommendations:

Provide a narrowly scoped way for guardian removal to handle the last AuthCancellerRole holder

Customer Response:

Fixed in commit 5ecdc8f

Fix Review:

Fixed

M-02: A rogue guardian can make itself unremovable and freeze liquidity during a liquidity upgrade

Severity: Medium	Impact: High	Likelihood: Low
Files: packages/timelock/sources/module/guardian_gov.move packages/timelock/sources/module/upgrade_gov.move packages/timelock/sources/module/role_gov.move packages/timelock/sources/module/liquidity_gov.move packages/liquidity/sources/module/admin.move packages/liquidity/sources/state/auth.move packages/liquidity/sources/state/registry.move	Status: Fixed	

Description:

Affected packages: `timelock`, `liquidity`

`RemoveGuardian` is the one admin action a guardian cannot cancel. The module doc relies on it: "a rogue guardian can block other actions for at most 12h" (guardian_gov.move#L5-L6). This bound does not hold during a liquidity upgrade.

`execute_remove_guardian` (guardian_gov.move#L79-L116) first revokes the guardian's local `GuardianRole` and `AuthCancellerRole` (guardian_gov.move#L99-L102). In the same call it then always calls `liquidity_admin::update_guardians` (guardian_gov.move#L109-L115). That function starts with `auth::assert_governance` (admin.move#L94), and the first check there is `registry.assert_version()` (auth.move#L110-L119). It compares the shared `Registry.version` with `constants::version()` of the liquidity code that the deployed `fluid_timelock` links against (registry.move#L41-L43, constants.move#L72).

A protocol upgrade takes two steps: upgrade the liquidity package, then upgrade `fluid_timelock` rebuilt against it (upgrade_gov.move#L5-L9). Liquidity `migrate` is instant, not timelocked, and only moves the version forward (admin.move#L169-L172, registry.move#L48-L53). If the `MigrateRole` holder runs it before the `fluid_timelock` relink commits, the deployed `fluid_timelock` still runs the old liquidity code against a migrated `Registry`. Every call to `execute_remove_guardian` then aborts with `wrong_version` (13001). The abort is deterministic for the whole window between `migrate` and the relink. The transaction aborts atomically, so no role is revoked and the `RemoveGuardian` operation simply stays pending.

The only way to close the window is the `fluid_timelock` upgrade. When its `UpgradeCap` is held in the `CapStore`, as the two-step flow assumes, it can never be taken out again (upgrade_gov.move#L11-L13). The relink must then go through `schedule_upgrade` and `authorize_scheduled_upgrade` on the 12h admin timelock. `cancel_admin` lets any guardian cancel any admin operation except one with `RemoveGuardianParams` (guardian_gov.move#L38-L53). So the guardian can cancel every `AuthorizeUpgrade` that would restore compatibility, for as long as it wants. There is no other exit. `RevokeRole` refuses any address that holds `GuardianRole` (role_gov.move#L214-L220). Liquidity `DisableTimelock` hits the same version gate (admin.move#L204).

The same liquidity leg gives a second way into this state. `assert_governance` also requires the transaction sender to hold liquidity `AdminRole` (auth.move#L94-L96). An executor that holds the timelock `ExecutorRole` but not liquidity `AdminRole` gets `admin_module_only_governance` (10002) on the same call. In both cases the removal of timelock roles fails because of a liquidity check that has nothing to do with them.

The result is more than a governance delay. While it keeps cancelling the relink, the guardian keeps a veto over every admin-timelock operation, and through `AuthCancellerRole` over every auth-timelock operation. Every liquidity action executed through the old `fluid_timelock` aborts on the same gate. This includes `UnpauseGlobal` (liquidity_gov.move#L617-L637, admin.move#L263-L279). A guardian that is also a liquidity guardian, which `RemoveGuardian` is designed to strip, can call `pause_global` directly on the new liquidity package, where the version check passes (admin.move#L250-L260). Supply, withdraw, borrow and payback are then blocked (user.move#L85-L95), and no unpause can execute. Depositors and borrowers in liquidity lose access to their funds, and governance loses control of the protocol, until the guardian stops cancelling.

The window exists only if the `MigrateRole` holder runs liquidity `migrate` before the `fluid_timelock` relink commits. The docs leave that timing to the migrator and do not require this order. Before `migrate`, `RemoveGuardian` still executes and the 12h bound holds. A guardian must also be rogue or compromised during that window.

Recommendations:

Decouple local guardian-role revocation from version-gated liquidity cleanup so removal can succeed during migrations; make the cleanup separately authorized and retryable.

Customer Response:

Fixed in commit e0d7414f

Fix Review:

Fixed

Low Severity Issues

L-01: Package upgrades can skip the 12h timelock after setup if an UpgradeCap is never stored

Severity: Low	Impact: High	Likelihood: Low
Files: packages/timelock/sources/module/role_gov.move packages/timelock/sources/state/auth.move packages/timelock/sources/state/cap_store.move packages/timelock/sources/module/upgrade_gov.move packages/liquidity/sources/state/auth.move	Status: Acknowledged	

Description:

Affected packages: `timelock`, `liquidity`

`complete_setup` verifies custody of protocol `TimelockCaps` but does not require the `UpgradeCaps` for governed packages or `fluid_timelock` to be stored in `CapStore`. A publisher retaining an `UpgradeCap` can upgrade its package without timelocked authorization and potentially change or remove the protections enforcing delays, despite setup being marked complete.

Recommendations:

Before completing setup, verify custody of the canonical `UpgradeCaps` for every governed package and `fluid_timelock`, or verify an approved immutable or independently timelocked configuration.

Customer Response:

Acknowledged - "*We accept the finding. No code changes are planned, as setup is a trusted phase and an on-chain check would not prevent a malicious deployer.*

Mitigation: Before launch, verify on-chain that all six UpgradeCaps are owned by CapStore. This remains publicly verifiable.
"

Fix Review:

Acknowledged

L-02: An address holding both GuardianRole and AdminCancellerRole can cancel its own removal and cannot be removed

Severity: Low	Impact: High	Likelihood: Low
Files: packages/timelock/sources/state/auth.move packages/timelock/sources/module/role_gov.move packages/timelock/sources/module/guardian_gov.move packages/timelock/sources/module/timelock.move	Status: Fixed	

Description:

Affected packages: `timelock`

An `AdminCancellerRole` holder can cancel pending admin-timelock operations, including a `RevokeRole` operation targeting itself, and there is no exception for self-revocation. Setup grants the publisher both `GuardianRole` and `AdminCancellerRole`, and role-grant paths do not prevent the roles from overlapping. A guardian with cancellation authority can therefore veto its own removal, while the described guardian role-revocation paths also prevent that role combination from being cleared.

Recommendations:

Prevent a holder from cancelling a `RevokeRole` operation targeting itself, or provide a non-cancellable revocation path or separate quorum for these roles. Enforce mutual exclusion where appropriate and ensure guardian removal clears applicable cancellation authority.

Customer Response:

Fixed in commit b7fd2cfda

Fix Review:

Fixed

L-03: Near-zero grace period makes every later timelock operation, including the fix, executable only in a one-millisecond window

Severity: Low	Impact: Medium	Likelihood: Low
Files: packages/timelock/sources/module/timelock.move packages/timelock/sources/utils/timelock_ops.move packages/timelock/sources/state/cap_store.move	Status: Fixed	

Description:

Affected packages: `timelock`

Configuration checks permit `min_delay_ms` to be zero and require only a positive grace period, allowing values as small as one millisecond. A very short grace period can make operations practically impossible to execute within their ready window. Since the value is stored when an operation is scheduled, subsequent corrective configuration operations can inherit the same unusable window.

Recommendations:

Enforce realistic minimum values for `min_delay_ms` and `grace_period_ms` in scheduling and execution paths, and consider separating configuration proposers from executors.

Customer Response:

Fixed in commit 397527b

Fix Review:

Fixed

L-04: An unpause approved for an earlier pause can lift a later emergency pause without a new review window

Severity: Low	Impact: Medium	Likelihood: Low
Files: packages/timelock/sources/module/liquidity_gov.move packages/timelock/sources/module/timelock.move packages/timelock/sources/module/guardian_gov.move packages/timelock/sources/utils/constants.move packages/liquidity/sources/module/admin.move packages/liquidity/sources/state/auth.move	Status: Acknowledged	

Description:

Affected packages: `timelock`, `liquidity`

Unpause proposals identify an action and, where applicable, a reserve or user, but not the specific pause they are intended to lift. Scheduling does not require an active pause, and execution checks only that the target is currently paused. After one approved operation lifts a pause, another can remain ready; if a guardian pauses the action again later, that stale operation can lift the new pause without a fresh proposal or review period. This applies to global, token-level, and user-level unpauses.

Recommendations:

Bind each unpause proposal to a pause generation or equivalent unique pause identifier, and require an exact match at execution. Requiring the target to be paused at scheduling can supplement this check but does not invalidate stale approvals on its own.

Customer Response:

Acknowledged
"*Mitigation: Cancel any remaining unpause operations for the same target after an unpause executes.*"

Fix Review:

Acknowledged

L-05: A rogue guardian's pause can freeze withdrawals, repayments and vault liquidations for at least about 24 hours, not the documented 12 hours

Severity: Low	Impact: Medium	Likelihood: Low
Files: packages/timelock/sources/module/guardian_gov.move packages/timelock/sources/module/liquidity_gov.move packages/timelock/sources/utils/timelock_ops.move packages/timelock/sources/utils/constants.move packages/liquidity/sources/module/admin.move packages/liquidity/sources/state/auth.move packages/liquidity/sources/module/user.move packages/vaults/sources/logic/liquidate.move packages/vaults/sources/invokes/liquidity.move	Status: Acknowledged	

Description:

Affected packages: `timelock`, `liquidity`, `vaults`

Guardians can pause liquidity actions instantly, while unpause operations require the 12-hour admin timelock. Guardians can cancel admin-timelock operations except `RemoveGuardian`. A guardian can therefore pause withdrawals or paybacks, cancel an unpause scheduled alongside its removal, and force governance to wait for removal before scheduling a replacement unpause. That unpause then requires another 12-hour delay, allowing the freeze to last about 24 hours. A Payback pause can also block debt repayment and liquidations that use payback.

Recommendations:

Exempt unpause operations from guardian cancellation, or have `RemoveGuardian` lift pauses imposed by the removed guardian as part of its execution.

Customer Response:

Acknowledged - *"No code changes are planned, as the maximum freeze is approximately 24 hours following the timelock fixes.*

Mitigation: Schedule RemoveGuardian alongside the unpause and reschedule cancelled unpauses immediately. Update documentation to reflect the potential 24-hour freeze, including Payback pauses."

Fix Review:

Acknowledged

L-06: Migrating liquidity before the lending and vaults relinks commit halts lending and vault user operations, liquidations included, for at least 12 hours

Severity: Low	Impact: Medium	Likelihood: Low
Files: packages/timelock/sources/module/upgrade_gov.move packages/timelock/sources/utils/constants.move packages/liquidity/sources/module/admin.move packages/liquidity/sources/module/user.move packages/liquidity/sources/state/registry.move packages/liquidity/sources/state/auth.move packages/liquidity/sources/constants.move packages/liquidity/sources/errors.move packages/lending/sources/invokes/liquidity.move packages/vaults/sources/invokes/liquidity.move packages/vaults/sources/logic/liquidate.move packages/lending_rewards/Move.toml	Status:	

Description:

Affected packages: `liquidity`, `lending`, `vaults`, `lending_rewards`, `timelock`

Liquidity's version check requires an exact match, and its `migrate` is instant and cannot be undone. Lending and vaults are linked to the liquidity package they were built against, so once liquidity is migrated, every lending and vault user operation aborts inside the old liquidity code

until those packages are relinked. Each relink is a 12-hour admin-timelock upgrade that can only be scheduled after the liquidity upgrade commits. The upgrade recipe in `upgrade_gov` names only `fluid_timelock` as a dependent, so a migrator who follows it can freeze deposits, withdrawals, repayments and liquidations for at least 12 hours, during which positions that cannot be liquidated can leave bad debt with lenders.

Recommendations:

Document and enforce an upgrade order in which liquidity is migrated only after every package that depends on it has been relinked, and confirm that ordering before migrating. Decide in advance how urgent liquidity fixes will be rolled out given the resulting delay.

Customer Response:

Acknowledged - "*The upgrade procedure in upgrade_gov now documents the required dependency order and sequential 12-hour admin-timelock upgrades, resulting in a minimum 48-hour upgrade process.*

All migrations are executed atomically in a single PTB after the dependent packages have been relinked, preventing version-gate failures. During this period, urgent liquidity issues can be mitigated through the guardian's immediate pause mechanism.
"

Fix Review:

Acknowledged

Informational Severity Issues

I-01: A TimelockAdminRole holder can raise min_delay_ms to 60 days, and the raise can only be reverted at the raised delay

Files:

packages/timelock/sources/module/timelock.move
packages/timelock/sources/utils/timelock_ops.move
packages/timelock/sources/module/guardian_gov.move
packages/timelock/sources/state/cap_store.move

Description:

Affected packages: `timelock`

Updating `min_delay_ms` is gated by the timelock admin role, allowing a `TimelockAdminRole` holder to schedule and execute the change without an ExecutorRole holder. The value can be raised to `MAX_DELAY_MS` (60 days), and `timelock_ops` uses the greater of the configured class floor and the timelock minimum for protocol actions. Once raised, governance operations - including removing the role holder, removing a guardian, disabling the timelock, unpausing, and upgrades - can require up to 60 days. Correcting the setting also requires an operation subject to the inflated minimum delay.

Recommendations:

Cap `min_delay_ms` at a bounded value relative to relevant class floors, or prevent `timelock_ops` from applying values above a fixed cap. Require an independent executor or proposer/executor pair for self-configuration, and allow a decrease to be scheduled without waiting for the inflated current minimum.

Customer Response:

Acknowledged - "*TimelockAdminRole is trusted and increasing the delay requires a scheduled operation subject to the current timelock (12h admin, 6h auth). During this window, authorized cancellers can prevent execution.*"

Fix Review:

Acknowledged

I-02: lending_rewards update_auth revoke aborts with a framework error for authorities granted after init

Files:

packages/lending_rewards/sources/module/admin.move
packages/lending_rewards/sources/state/rewards_auth.move
packages/timelock/sources/module/rewards_gov.move
docs/STANDARDS.md
setup/src/init/lending_rewards.ts

Description:

Affected packages: `lending_rewards`, `timelock`

The PR widened `is_auth` to include authorities and stopped creating an `auths` record when an authority is granted, but the revoke branch of `update_auth` was not updated to match. It still relies on the widened check and then removes a record that may not exist, so revoking an authority granted after init aborts with a framework error instead of a protocol error. A timelocked `UpdateAuth` revocation for such an address waits 12 hours and then cannot execute. Access control is unaffected, since `update_authority` remains the correct way to remove an authority.

Recommendations:

Make the revoke branch of `update_auth` validate the condition it actually depends on, so it fails with the protocol's own error, and align the setup script's revocation steps with the new grant behaviour.

Customer Response:

Fixed in commit 8105313

Fix Review:

Fixed

I-03: Oracle timelock cap mint and disable skip the version check and emit no events, unlike the other layers

Files:

packages/oracle/sources/module/admin.move
packages/oracle/sources/types/events.move
packages/oracle/sources/state/registry.move
packages/oracle/sources/state/auth.move
packages/timelock/sources/module/oracle_gov.move
packages/timelock/sources/module/role_gov.move
packages/liquidity/sources/module/admin.move

Description:

Affected packages: `oracle`, `timelock`

In the oracle package, `mint_timelock_cap` and `disable_timelock` check only that the sender holds `AdminRole`. Unlike the other oracle setters and the matching functions in the other four layers, they do not check the package version and emit no events. After an oracle upgrade, the old package's versions of these functions therefore remain callable, and monitoring sees no timelock enable or disable events for the oracle. The direct security effect is small, but once the package is published, changing these signatures is no longer a compatible upgrade.

Recommendations:

Before the oracle package is first published, align these functions with the other layers by adding the version check and emitting timelock enable and disable events.

Customer Response:

Fixed in commit e516792 & commit 62b7aec3

Fix Review:

Fixed

I-04: Two new vaults error codes reuse existing abort codes 6078 and 6079

Files:

packages/vaults/sources/errors.move
packages/vaults/sources/state/auth.move
packages/vaults/sources/logic/operate.move
packages/vaults/sources/state/vault_market.move

Description:

Affected packages: `vaults`

The PR adds `vault_admin_only_migrators` and `vault_admin_timelock_ticket_required` with abort codes 6078 and 6079, which already belong to `vault_price_quote_required` and `vault_same_col_debt_type`. On-chain behaviour is unaffected, because the abort location still tells each pair apart. Off-chain scripts, monitors and runbooks that map errors by number alone will mislabel them, and the codes become part of the published interface once the package is deployed.

Recommendations:

Assign the new errors codes that no other vaults error uses before publishing, and add a check that prevents duplicate error codes within a package.

Customer Response:

Fixed in commit b0d11a9

Fix Review:

Fixed

I-05: Timelock comments and docs misstate stray TimelockCap recovery, omit the DisableTimelock one-way door, and mislabel the migrate gate as permissionless

Files:

packages/liquidity/sources/module/admin.move
packages/vaults/sources/module/admin.move
packages/lending/sources/module/admin.move
packages/lending_rewards/sources/module/admin.move
packages/oracle/sources/module/admin.move
packages/liquidity/sources/state/auth.move
packages/vaults/sources/state/auth.move
packages/lending/sources/state/auth.move
packages/lending_rewards/sources/state/rewards_auth.move
packages/oracle/sources/state/auth.move
packages/liquidity/sources/state/registry.move
packages/vaults/sources/state/registry.move
packages/timelock/sources/module/upgrade_gov.move
packages/timelock/README.md
docs/ARCHITECTURE.md

Description:

Affected packages: `liquidity`, `vaults`, `lending`, `lending_rewards`, `oracle`, `timelock`

Several code comments and documents do not match the code. The recovery advice for a stray `TimelockCap` assumes the holder can always burn it, which fails when the holder is outside the top tier or the cap is frozen, wrapped or lost, leaving a package upgrade as the only fix. The list of one-way doors omits `DisableTimelock`, although a layer cannot be re-enabled through the timelock after setup. The registry comments describe the `migrate` gate as permissionless, although it requires `MigrateRole` or the auth tier, and the architecture document and timelock README describe an earlier design. Operators who rely on these comments can choose a recovery step that does not work, or disable a layer without knowing it cannot be re-enabled.

Recommendations:

Update the affected comments and documents to describe the shipped custody, recovery, migration and timelock behaviour, including which actions cannot be reversed after setup.

Customer Response:

Fixed

Fix Review:

Fixed

I-06: Most new timelock adapter paths, the migrator-role grants and the oracle update_admin version check have no tests

Files:

docs/STANDARDS.md
packages/timelock/sources/module/vaults_gov.move
packages/timelock/sources/module/liquidity_gov.move
packages/timelock/sources/module/lending_gov.move
packages/timelock/sources/module/rewards_gov.move
packages/timelock/sources/module/oracle_gov.move
packages/timelock/sources/module/role_gov.move
packages/timelock/sources/module/upgrade_gov.move
packages/timelock/sources/state/cap_store.move
packages/timelock/sources/state/auth.move
packages/timelock/sources/utils/timelock_ops.move
packages/timelock/sources/utils/errors.move
packages/timelock/tests/roles_tests.move
packages/timelock/tests/timelock_tests.move
packages/liquidity/sources/module/admin.move
packages/lending/sources/module/admin.move
packages/lending_rewards/sources/module/admin.move
packages/vaults/sources/module/admin.move
packages/oracle/sources/module/admin.move
packages/oracle/tests/oracle_admin_tests.move

Description:

Affected packages: `timelock`, `vaults`, `liquidity`, `lending`, `lending_rewards`, `oracle`

The repository's standard requires a named test for every abort, but much of the new surface has no tests. Only 4 of the 65 timelock adapter execute wrappers are called by a test, and the timelocked upgrade path and `migrate_cap_store` are never exercised. Ten timelock abort codes have no expected-failure test, including the checks that bind an execution to its scheduled target and parameter type. The new migrator-role grants are untested in four layers, and the PR removes the oracle `update_admin` version-check test. No funds are at risk, but a later change that breaks one of these paths would ship without a failing test.

Recommendations:

Add tests covering the timelock adapter paths, the upgrade and cap-store migration flows, every abort code and the migrator-role changes, as the project's standard requires, and consider enforcing coverage in CI.

Customer Response:

Fixed

Fix Review:

Fixed

I-07: MintUserCap can operate without the vault's collateral checks

Files:

packages/timelock/sources/module/liquidity_gov.move#L67-L70

packages/timelock/sources/module/liquidity_gov.move#L495-L541

packages/liquidity/sources/module/admin.move#L211-L225

packages/vaults/sources/module/admin.move#L95-L135

Description:

MintUserCap accepts a Holder without rejecting vault object IDs and sends the minted UserCap to a chosen recipient. Since a future vault's ID is predictable, governance can mint its cap before the vault exists and send it outside the vault, bypassing the atomic mint_and_install_caps path. Once the vault's liquidity position is configured, the cap holder can operate it without the vault's collateral checks. The existing cap also causes the vault's later installation attempt to fail.

Recommendations:

Clarify whether governance may mint UserCaps for vault holders through the generic MintUserCap flow. If vault caps must be installed atomically, reject vault holders in that flow. Otherwise, document the risk of using a predicted future vault ID and require governance to verify the intended recipient and vault creation order before approval.

Customer Response:

Acknowledged - "*Liquidity cannot identify future vault IDs on-chain. MintUserCap is timelocked, publicly visible, and cancellable by guardians. We documented the required use of vaults::admin::mint_and_install_caps and governance checks against existing and pending vault IDs. Mis-minted capabilities can be revoked through set_user_authorization.*"

Fix Review:

Acknowledged

I-08: Retired timelock package versions can still invoke public state-mutating primitives

Files:

packages/timelock/sources/module/timelock.move:295-303

packages/timelock/sources/module/timelock.move:332-350

packages/timelock/sources/module/timelock.move:379-396

packages/timelock/sources/state/cap_store.move:129-132

Description:

Public timelock self-configuration primitives do not check the canonical version anchor. After an upgrade and CapStore migration, callers can still invoke historical package implementations against shared timelock state, including paths whose authorization or configuration behavior may have been tightened in the new version.

Recommendations:

Consider version-gating the four timelock self-configuration functions that schedule or execute changes to the minimum delay and grace period.

Customer Response:

Fixed in commit 211bf816e

Fix Review:

Fixed

I-09: RemoveGuardian can revoke AuthCancellerRole from non-guardians without guardian veto

Files:

packages/timelock/sources/module/guardian_gov.move#L57-L116

packages/timelock/sources/module/role_gov.move:166-220

Description:

Neither scheduling nor execution verifies that the target is a timelock guardian or a liquidity guardian. Execution unconditionally revokes the target's AuthCancellerRole where held and removes its liquidity guardian record where present. Since RemoveGuardian is not subject to the usual guardian cancellation path, an AdminProposerRole holder and an ExecutorRole holder can use it to remove AuthCancellerRole from ordinary cancellers, bypassing the cancellable normal role-revocation path. It can also affect liquidity-only guardians without a timelock GuardianRole.

Recommendations:

Require at scheduling and execution that the target holds GuardianRole or has a liquidity guardian record. Limit AuthCancellerRole revocation to valid guardian removal; use the cancellable RevokeRole path for non-guardian cancellers.se

Customer Response:

Fixed in commit dd15c6bd

Fix Review:

Fixed

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.

Finally, Certora delivers ongoing monitoring and incident response, enabling continuous threat detection and prevention, alert triage, and coordinated response to active incidents.