PUBLIC

Code Assessment

of the Mento Liquity v2 Smart Contracts

February 17, 2026

Produced for

by

Contents

1 Executive Summary

Dear Mento Team,

Thank you for trusting us to help Mento with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Mento Liquity v2, according to Scope, to support you in forming an opinion on their security risks.

Mento implements a fork of the Liquity v2 protocol, extending its functionality to enable users to open collateralized debt positions (CDPs) by depositing the USDm stablecoin and borrowing non-USD stablecoins against it. Key additions include governance-controlled SystemParams, an upgradable Stability Pool that exposes liquidity to external strategies for rebalance operations, and an FX price feed.

The most critical subjects covered in our audit are functional correctness, system parameters, oracles and implications of the new rebalance functionality. Functional correctness is now high after addressing Batch Manager is not deleted in kickFromBatch. Correctness regarding boundaries for system parameters has been improved after addressing minDebt Bounds Do Not Allow Configuring Low-value Currencies Correctly. Governance choosing correct values continues to be essential to ensure the security of the system. The oracle implementation is secure. However, it is intentionally not available over weekends, which can cause delays in liquidations and leads to Cannot add collateral if market closed. Finally, the new rebalance functionality as well as the permissionless oracle price relaying introduce additional risks and break assumptions of Liquity v2, such as Redistributions Are More Likely To Happen. These risks should be actively monitored.

We have also provided Notes on important considerations which can aid in understanding the system.

In summary, we find that the codebase provides a satisfactory level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	1
• Code Corrected	1
Medium-Severity Findings	0
Low-Severity Findings	4
• Code Corrected	3
• Risk Accepted	1

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Mento Liquity v2 repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	20 October 2025	e964b6a9637df990a92136d6455f90151ab507b5	Initial Version
2	2 February 2026	ea31dab1701be2d83096d3574c4bd18891961bb5	Fixes
3	17 February 2026	073a25f049f423b3336e3785d5f1e13235ced07b	Final Version

For the solidity smart contracts, the compiler version `0.8.24` was chosen.

The following contracts were in scope for the review:

```
contracts/src/Dependencies/Constants.sol
contracts/src/Dependencies/LiquityBaseInit.sol

contracts/src/Dependencies/AddRemoveManagers.sol
contracts/src/Dependencies/AggregatorV3Interface.sol
contracts/src/Dependencies/Constants.sol
contracts/src/Dependencies/LiquityBase.sol
contracts/src/Dependencies/LiquityBaseInit.sol
contracts/src/Dependencies/LiquityMath.sol
contracts/src/Dependencies/Ownable.sol

contracts/src/PriceFeeds/FXPriceFeed.sol

contracts/src/tokens/patched/ERC20PermitUpgradeable.sol
contracts/src/tokens/patched/ERC20Upgradeable.sol
contracts/src/tokens/StableTokenV3.sol

contracts/src/Types/BatchId.sol
contracts/src/Types/LatestBatchData.sol
contracts/src/Types/LatestTroveData.sol
contracts/src/Types/TroveChange.sol
contracts/src/Types/TroveId.sol

contracts/src/ActivePool.sol
contracts/src/AddressesRegistry.sol
contracts/src/BatchManagerOperations.sol
contracts/src/BorrowerOperations.sol
contracts/src/CollateralRegistry.sol
contracts/src/CollSurplusPool.sol
contracts/src/DefaultPool.sol
contracts/src/GasPool.sol
contracts/src/SortedTroves.sol
```

```
contracts/src/StabilityPool.sol
contracts/src/SystemParams.sol
contracts/src/TroveManager.sol
contracts/src/TroveNFT.sol
```

In Version 2 the following contracts were removed from the scope:

```
contracts/src/tokens/patched/ERC20PermitUpgradeable.sol
contracts/src/tokens/patched/ERC20Upgradeable.sol
contracts/src/tokens/StableTokenV3.sol
```

2.1.1 Excluded from scope

All other contracts are out of scope.

The economic model and the choice of values for configurable parameters are also out of scope.

2.2 System Overview

This system overview describes Version 2 of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Mento offers a fork of the Liquity v2 protocol, extending its functionality to enable users to open collateralized debt positions (CDPs) by depositing the USDm stablecoin and borrowing non-USD stablecoins against it. Protocol parameters can now be adjusted through governance, and multiple contracts are now upgradable.

In addition, the fork introduces a rebalancing mechanism designed to maintain liquidity equilibrium in the Mento FPMM. This mechanism leverages the Stability Pool from Liquity v2, enabling swaps between collateral tokens from the FPMM and debt tokens from the Stability Pool.

This system overview focuses on the components that were modified from Liquity v2. A detailed description of Liquity v2 can be found in the system overview of our Liquity v2 audit report.

2.2.1 Changes from Liquity v2

1. **SystemParams**: The newly added SystemParams contract manages system-wide parameters of the protocol. Implementing an upgradable storage for all protocol parameters, it enables modifications of the parameters through governance, instead of them being immutable after deployment in Liquity v2.
2. **Upgradability**: The StabilityPool contract is now upgradable, as it extends the new LiquityBaseInit contract, which is an upgradable version of the base contract. The Bold token is replaced with the StableTokenV3 contract from Mento v3 core, which is also upgradable by governance.
3. **BatchManagerOperations**: some of the BorrowerOperations contract implementation was split to the BatchManagerOperations contract to avoid exceeding the Solidity contract size limit following the SystemParams modifications. The logic remains the same.
4. **FXPriceFeed**: The FXPriceFeed contract replaces the direct oracle price feeds with a Mento v3's OracleAdapter feed.
5. **StabilityPool**: The StabilityPool contract was extended with a rebalancing functionality.

2.2.2 Rebalancing

The StabilityPool was extended with the `swapCollateralForStable()` and `_swapCollateralForStable()` functions, which enable a Mento v3 core LiquidityStrategy to swap collateral tokens from an unbalanced market maker against debt tokens from the StabilityPool. The amounts of collateral tokens to receive and of debt tokens to send are set by the LiquidityStrategy and are not verified. The LiquidityStrategy is therefore trusted to provide values that preserve the value of LP positions, and its address is set during the initialization of the contract.

The mechanism is the same as a regular Liquity v2 liquidation: the removed debt tokens are removed from the positions of the Liquidity Providers (LPs) of the StabilityPool, and the received collateral tokens are added to the LP's collateral gains. The `MIN_BOLD_AFTER_REBALANCE` parameter (the minimum amount of debt tokens which must remain in the pool after a rebalance) is introduced to limit the rate of scale changes of the pool's `P` product.

The CollateralRegistry was extended with function `redeemCollateralRebalancing`, which enable a Mento v3 core LiquidityStrategy to redeem collateralized debt positions (CDP's) similar to regular redemptions except that the liquidity strategy pays the fee to the owner, not the registry.

2.2.3 FXPriceFeed

All prices are now fetched from the ownable and upgradable FXPriceFeed contract, which in turn fetches prices from Mento v3's OracleAdapter. The OracleAdapter's address is set during initialization and can be updated by the owner. The `fetchPrice()` function returns the FX rate as a `uint256`, optionally inverting it when the `invertRateFeed` flag is set. Due to the implementation of OracleAdapter in Mento v3 core, the `fetchPrice()` function will revert when called outside of market hours. By extension, all operations relying on the oracle price, such as liquidations, opening a new trove, or adding collateral, are not possible outside of market hours.

The price feed queries an oracle for the status of the L2 Sequencer and reverts once sequencer downtime is detected.

The contract implements an emergency shutdown functionality via the `shutdown()` function which can only be called by a `watchdog`, and only when the oracle is not yet shut down. The `watchdog` address is set during initialization and can be updated by the owner with `setWatchdogAddress()`. When triggered, the shutdown acts as a `shutdownFromOracleFailure`.

2.2.4 Migration

A number of non-USD stablecoins have already been deployed previously with an early version of the protocol. Mento plans to migrate these tokens to the new CDP system by creating an equivalent CDP position and burning the debt tokens.

2.3 Changes in Version 2

- LiquidityStrategy now calls the `redeemCollateralRebalancing` function when redeeming to use a custom fee. Previously it used the standard redemption path.
- Liquidations are disabled when the sequencer is down (and for a grace period after it comes back up).
- FXPriceFeed now supports an `invertRateFeed` flag that computes the inverse of the oracle rate when enabled.
- FXPriceFeed now has owner-only setter functions for the OracleAdapter address, the rate feed ID, and the `invertRateFeed` flag.
- Rebalancing via `swapCollateralForStable()` is now disabled when the system is shut down.

2.4 Trust Model

The Liquity system was designed to be immutable with limited trust assumptions. However, multiple contracts are now upgradable, and the system parameters are now modifiable, which introduces additional trusted roles.

The Mento governance (fully trusted) can upgrade the StabilityPool and the StableTokenV3, modify all system parameters, and reconfigure the FXPriceFeed. Malicious parameter values or a malicious OracleAdapter could lead to all funds becoming stuck or lost. Additionally, governance can assign the Minter, Burner, and Operator roles in StableTokenV3, enabling a compromised governance to mint arbitrary amounts of unbacked stablecoins or seize any user's tokens. The roles are expected to be held only by the Liquity smart contracts.

The OracleAdapter (fully trusted) is trusted to provide price updates within the expected threshold and return correctly formatted prices, with the expected number of decimals. It is expected that it reverts outside of market hours. Incorrect prices could lead to a loss of all value in the system.

The LiquidityStrategy (fully trusted) which is whitelisted in the StabilityPool is trusted to provide fair values for the amounts of collateral tokens to receive and of debt tokens to send, as there are no additional checks on those values. A malicious LiquidityStrategy could steal all tokens from the StabilityPool. Further, trove owners must trust the liquidity strategy to calculate a fair fee for the rebalancing transaction as a malicious strategy could set this fee to zero and trade with the trove owner at the oracle price.

The `watchdog` role (partially trusted) of the FXPriceFeed is trusted, as an unintended shutdown of the price feed would lead to a shutdown of the Liquity system, which would lead to losses and denial of service.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

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

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Open Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	1

- Redistributions Are More Likely To Happen Risk Accepted

5.1 Redistributions Are More Likely To Happen

Design Low Version 1 Risk Accepted

CS-MELI-004

Liquity V2 treats redistributions as a last resort: they only trigger when the Stability Pool is depleted and large undercollateralized troves exist simultaneously. Redistributions are considered very unlikely, and a high number of them could lead to the mechanism collapsing due to rounding (see Repeated redistribution can eventually result in zero stake Troves). Deliberately triggering sizable redistributions is considered difficult to engineer, since they require both the Stability Pool to be empty and large liquidatable Troves to be available.

Mento's extensions invalidate both assumptions:

- Anyone can create a new trove, then push a new price update permissionlessly that makes the trove liquidatable.
- Rebalancing can empty the Stability Pool at any moment, even when the system is healthy, and can also be triggered permissionlessly.

An attacker can therefore drain the StabilityPool through rebalancing, publish a new low FX price, and liquidate large troves. Those liquidations now redistribute debt/collateral to other troves without paying the gas compensation that direct liquidations provide, and the zero stake redistribution path becomes more common because the attacker can repeat the sequence.

See also Stability Pool can be emptied out.

Risk accepted:

Mento accepted the risk, but has decided to keep the code unchanged.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Open Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	1

- Batch Manager Is Not Deleted in kickFromBatch Code Corrected

Medium-Severity Findings	0
Low-Severity Findings	3

- Incorrect StabilityPool Interface Code Corrected
- Rebalances Are Partially Allowed During Shutdown Code Corrected
- Swapping Collateral Does Not Accrue Interest Code Corrected

Informational Findings	7

- Insufficient Check in SystemParams Code Corrected
- LiquidityStrategy Cannot Be Changed Code Corrected
- Missing Sanity Checks Code Corrected
- Redistribution Penalty Should Be Smaller Than MCR Code Corrected
- Whitepaper Deviations Specification Changed
- minDebt Bounds Do Not Allow Configuring Low-Value Currencies Correctly Code Corrected
- swapCollateralForStable Event Indistinguishable From Liquidations Code Corrected

6.1 Batch Manager Is Not Deleted in kickFromBatch

Correctness High Version 1 Code Corrected

CS-MELI-001

Due to an oversight in moving functionality to `BatchManagerOperations`, `kickFromBatch()` in `BorrowerOperations` does not delete `interestBatchManagerOf[_troveId]`.

As a result, `TroveManager` thinks the trove is standalone, while `BorrowerOperations` still routes it through the batched code paths. The implications are:

- Any subsequent call to `closeTrove` takes the batch branch and subtracts the trove's debt/collateral from `LatestBatchData`. This might allow a user to exit the protocol and repay no debt.
- The borrower can't adjust their interest rate or assign a new batch manager because `_requireIsNotInBatch()` now always reverts.
- The mapping is never corrected elsewhere, so the trove remains permanently stuck unless the contract is upgraded.

Code corrected:

Mento added the missing `delete interestBatchManagerOf[_troveId];` statement in the `kickFromBatch()` function.

6.2 Incorrect StabilityPool Interface

Correctness | Low | Version 1 | Code Corrected

CS-MELI-002

In IStabilityPool, the `swapCollateralForStable` function lists its arguments in the opposite order from the implementation. The interface should take `amountCollIn` first, followed by `amountStableOut`, to match the contract.

Code corrected:

Mento fixed the parameters order in the `IStabilityPool` interface to match the implementation.

6.3 Rebalances Are Partially Allowed During Shutdown

Design | Low | Version 1 | Code Corrected

CS-MELI-003

In StabilityPool, the `swapCollateralForStable` function continues working during shutdown. This means it can still be used by an FPMM to rebalance. However, the other rebalance direction using redemptions is disabled during shutdown, as redemptions are disabled. Only urgent redemptions can be done during shutdown.

The `swapCollateralForStable` function should also be disabled during shutdown.

Code corrected:

Mento added a `_requireNoShutdown()` check to the `swapCollateralForStable` function.

6.4 Swapping Collateral Does Not Accrue Interest

Correctness | Low | Version 1 | Code Corrected

CS-MELI-005

The function `StabilityPool.swapCollateralForStable` does not accrue interest when collateral is swapped, but reduces the values `totalBoldDeposits` and `P` that are later used to distribute interest. The next time interest is accrued (e.g., when `provideToSp` or `batchLiquidate` is called), the interest will be distributed using the reduced values.

In detail, the function `_updateYieldRewardsSum` updates the yield reward running sum as:

$$scaleToB = yield \cdot \hat{P} / \hat{totalDeposits}$$

where $\hat{P}$ and $\hat{totalDeposits}$ denote the new state variables after interest accrual is called the next time. These variables will generally be smaller than the correct values, since rebalancing reduced the total deposits and running variable P.

These variables will then be used to calculate the yield gain for each depositor in `getDepositorYieldGain()`:

$$\hat{yieldGain} = \frac{yield \cdot \hat{P}}{\hat{totalDeposits} \cdot P_{initial} \cdot SCALE\_FACTOR}$$

Since these values will be incorrect, we can estimate the relative error for $\hat{yieldGain}$ compared to $yieldGain$ that would use the correct state variables:

$$error = \frac{\hat{yieldGain}}{yieldGain} - 1$$

Interestingly, since the rebalancing simply reduces the total deposits by some amount Δd. We have $\hat{totalDeposits} = totalDeposits - \Delta d$ and by definition of P, $\hat{P} = P \cdot (totalDeposits - \Delta d)/totalDeposits$, plugging these in yields:

$$error = \frac{P \cdot (totalDeposits - \Delta d)/totalDeposits}{P} \cdot \frac{totalDeposits}{totalDeposits - \Delta d} - 1 \approx 0$$

In conclusion, while it is incorrect, the error is negligible.

Code corrected:

Mento added the `activePool.mintAggInterest()` call at the beginning of the `swapCollateralForStable` function to accrue interest before the swap.

6.5 Insufficient Check in SystemParams

Informational **Version 1** **Code Corrected**

CS-MELI-006

SystemParams only enforces that `minBoldInSP` is not zero before storing the value.

However, the rest of the system assumes that at least 1e18 collateral tokens remain in the StabilityPool at all times after the initial deposits. This can be seen in the following code comment:

```
Once totalBoldDeposits has become >= MIN_BOLD_IN_SP, a liquidation may never fully
empty the Pool - a minimum of 1 BOLD remains in the SP at all times thereafter.
```

This is important to prevent P from reaching zero and introducing catastrophic rounding errors.

Code corrected:

Mento added proper validation for `minBoldInSP` in the `SystemParams` constructor, ensuring that `minBoldInSP >= 1e18`.

6.6 LiquidityStrategy Cannot Be Changed

Informational **Version 1** **Code Corrected**

CS-MELI-007

Mento has stated that the liquidity strategy used by the StabilityPool should be changeable, but currently there is no function to change it.

Code corrected:

The liquidity strategies were made upgradeable in the mento-core repository. This allows their logic to be changed if needed.

6.7 Missing Sanity Checks

Informational | Version 1 | Code Corrected

CS-MELI-010

The contract enforces a minimum threshold amount of debt tokens after a rebalance, referred to as `MIN_BOLD_AFTER_REBALANCE`, in the `swapCollateralForStable()` function. However, there is no explicit check to ensure that the value of `MIN_BOLD_AFTER_REBALANCE` is greater than or equal to the value of `MIN_BOLD_IN_SP`, which represents the minimum amount of debt tokens required to be maintained in the Stability Pool after liquidations.

If `MIN_BOLD_AFTER_REBALANCE` is set to a value less than `MIN_BOLD_IN_SP`, the contract could allow for a situation where the total Bold deposits fall below the required minimum threshold during rebalances.

Code corrected:

Mento added a validation check in `SystemParams` to ensure that `minBoldAfterRebalance >= minBoldInSP`.

6.8 Redistribution Penalty Should Be Smaller Than MCR

Informational | Version 1 | Code Corrected

CS-MELI-011

In `Constants`, the maximum penalty for liquidation during redistributions (`MAX_LIQUIDATION_PENALTY_REDISTRIBUTION`) is set to 20%.

However, the redistribution penalty should be at most the system's overcollateralization margin (`MCR - 100%`), since a trove can only lose at most its equity.

For example, if the `MCR` is set to 110%, a trove falling slightly below MCR has at most 10% equity. A 20% penalty could therefore never be fully applied. Setting the maximum penalty higher than the available equity is inconsistent with the `MCR` chosen.

The constructor of `SystemParams` did not enforce that `MAX_LIQUIDATION_PENALTY_REDISTRIBUTION <= MCR - 100%`.

Code corrected:

Mento added validation in the `SystemParams` constructor to ensure that the redistribution penalty does not exceed `MCR - 100%`.

6.9 Whitepaper Deviations

Informational | Version 1 | Specification Changed

CS-MELI-013

The whitepaper states that, during a contraction rebalance, the CDP Liquidity Strategy can withdraw collateral tokens from the Stability Pool in exchange for repaying debt tokens:

```
Flash-swap contract – If traders have reduced the USD.m reserves of the FPMM beyond
the tolerance, the strategy reverses the above trade: it withdraws USD.m from the Stability
Pool and repays an equivalent amount of JPY.m to the Stability Pool.
```

However, this is not the case in `CDPLiquidityStrategy.sol`. The strategy can only execute contractions by using the redemption functionality of Liquity V2.

Specification changed:

Mento updated the whitepaper to accurately reflect the implementation.

6.10 minDebt Bounds Do Not Allow Configuring Low-Value Currencies Correctly

Informational | Version 1 | Code Corrected

CS-MELI-014

The constructor of `SystemParams` enforces a boundary of `0 < minDebt < 10_000e18` on the minimum debt threshold for a Trove.

While 10 000 units is enough for USD-pegged tokens, it is tiny for low-value currencies (e.g., 10 000 KRW ≈ $7). Those branches often need much higher minimum debts to keep liquidations economically viable and cover fixed gas/GasComp costs. However, the constructor does not allow any value above 10 000 tokens.

Code corrected:

Mento removed the maximum validation on the `minDebt` parameter, allowing it to be configured higher for low-value currencies.

6.11 swapCollateralForStable Event Indistinguishable From Liquidations

Informational | Version 1 | Code Corrected

CS-MELI-015

In `StabilityPool`, `swapCollateralForStable()` emits the same `StabilityPoolCollBalanceUpdated` and `StabilityPoolBoldBalanceUpdated` events that are emitted during liquidations. There is no dedicated event to signal that the change came from a liquidity strategy rebalance rather than a trove liquidation.

As a result, external monitoring cannot tell when pool assets are taken by manual rebalances versus regular liquidations.

Code corrected:

Mento added a dedicated `RebalanceExecuted` event that is emitted by the `swapCollateralForStable` function to distinguish rebalances from liquidations.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Missing Events

Informational | Version 1 | Acknowledged

CS-MELI-009

- Several modified contracts cache the address of `SystemParams` in their constructors but do not emit a corresponding event, even though events for other dependencies are emitted. A non-exhaustive list of affected contracts includes:
    - `ActivePool`
    - `BorrowerOperations`
    - `BatchManagerOperations`
    - `CollateralRegistry`
    - `StabilityPool`
    - `TroveManager`
- `BatchManagerOperations` does not emit any event when setting the addresses of its dependencies in the constructor.
- `StabilityPool` does not emit an event when setting the address of its `liquidityStrategy` in `initialize()`.

These missing events make it more difficult to verify deployments and detect misconfigurations.

Acknowledged:

Client is aware of this behavior, but has decided to keep the code unchanged.

7.2 Stability Pool Can Be Emptied Out

Informational | Version 1 | Risk Accepted

CS-MELI-012

As anyone can permissionlessly push chainlink prices in the ChainlinkRelayer, it is possible to intentionally empty out the stability pool in the following way:

1. Create a large trove with ICR equal or right above MCR
2. Push a price update that reduces the oracle price
3. Liquidate the freshly created trove

These steps can be done atomically with a flashloan.

Generally, liquidations should be profitable for the stability pool, so there is no incentive to do this. However, there may be extreme market conditions in which liquidations are unprofitable for the stability

pool and profitable for the trove owner. Once the stability pool is empty, further liquidations will be done as redistributions to other troves.

There is a similar known issue in Liquity V2. The difference is that in Liquity V2, it is not possible to permissionlessly push the Chainlink price, so the actions cannot be executed atomically, and cannot use a flashloan. For more information, reference the Liquity V2 audit report, issue CS-BOLD-019.

Risk accepted:

Mento has accepted the risk, but has decided to keep the code unchanged.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 CCR and SCR Considerations

Note Version 1

Note that the CCR and SCR must be set in such a way that they are not too close to each other to prevent intentional triggering of branch shutdown.

In particular, the CCR should be higher than the SCR by an amount that is greater than the largest expected price change in a single oracle update. Otherwise, an attacker could intentionally trigger shutdown by frontrunning the oracle update.

8.2 Cannot Add Collateral if Market Closed

Note Version 1

The FXPriceFeed reverts outside of market hours. As a result, users cannot add collaterals to their troves while the market is closed, as adding collateral requires an oracle price. This means that users cannot top up their troves in case they expect a price drop of the collateral token while the market is closed.

It should also be noted that interest continues to accrue on troves while the market is closed. As a result, it is possible that a trove is not liquidatable when the market closes, but becomes liquidatable while the market is closed due to interest accrual.

8.3 Changing Upgradeable Parameters Can Be Dangerous

Note Version 1

In Liquity V2, all parameters are constants. In Mento, most parameters were made upgradeable. Any upgrade of the parameters once the contracts are live should be carefully reviewed and simulated to ensure that it does not break the contracts.

The `SystemParams` contract is itself upgradeable, meaning that all of its stored values can be changed.

Below is a non-exhaustive collection of parameters that could be problematic to change:

The `ETH_GAS_COMPENSATION` should never be changed.

1. Increasing the `ETH_GAS_COMPENSATION` will lead to insolvency, since the amount of `ETH_GAS_COMPENSATION` deposited for each user is not cached, so users would be able to get more out than they originally paid in. Similarly, liquidators would receive more.
2. Reducing the `ETH_GAS_COMPENSATION` will lead to stuck funds.

The Critical Collateralization Ratio (CCR), Shutdown Collateralization Ratio (SCR), and Maintenance Collateralization Ratio (MCR) should only be changed with utmost care. It must be ensured that changing them does not create bad debt or trigger shutdown. Not only is their absolute size important, but also their relation to `LIQUIDATION_PENALTY_SP`, `LIQUIDATION_PENALTY_REDISTRIBUTION`, `COLL_GAS_COMPENSATION_DIVISOR`, the volatility of the collateral pairs, as well as the chainlink

deviation. Changes could result in liquidations not being profitable enough, creating bad debt, or making self-liquidation profitable.

A Buffer Collateralization Ratio (BCR) lower than a chainlink update can enable the issue CS-BOLD-019 `Upfrontfee Can Bring Troves Below MCR` from the Liquity V2 audit report.

8.4 Gas Compensation Must Be Sufficient

Note **Version 1**

The gas compensation for liquidations must be set sufficiently high to ensure that it is always profitable to liquidate troves. In particular, it must be high enough to cover the gas when the network is highly congested, not just the average gas cost.

8.5 LIQUIDATION_PENALTY_SP Must Be Large Enough

Note **Version 1**

The `LIQUIDATION_PENALTY_SP` parameter defines the penalty applied to troves when they are liquidated through the Stability Pool. It is important that this penalty is set sufficiently high, such that liquidations are profitable for the Stability Pool. If it is too low, it can be profitable to intentionally create a liquidatable position and liquidate it through the Stability Pool.

In particular, the penalty should be higher than the Chainlink deviation threshold for the FX pair. Whenever the current oracle price deviates from the true market price by more than the `LIQUIDATION_PENALTY_SP`, liquidations are unprofitable for the Stability Pool.

Note that price updates can be larger than the Chainlink deviation threshold, so this is a lower bound. The value should be high enough that an oracle deviation larger than the penalty is unlikely to happen.

Note that it is easier to create a liquidatable position in Mento than in Liquity V2, since anyone can permissionlessly push a price update. See also Stability Pool can be emptied out.

8.6 MCR Must Be Sufficient for Expected Volatility

Note **Version 1**

The MCR must be chosen in a way that accounts for the largest price swing the FX pair can experience before liquidations can react. For example, an MCR of 110 % assumes the price will not drop by more than 10 % in the time it takes to liquidate.

In particular, it must also not move by this amount between the market close and the market open. Markets could be closed for three days in a row when a weekend overlaps with a holiday.

8.7 Rebalancing via Redemptions Can Fail

Note **Version 1**

Rebalancing via redemptions reverts if the total redeemable debt is lower than the requested rebalance amount. Under normal conditions this is not expected, since all debt tokens are issued by the Liquity system and `maxIterations` can be set sufficiently high.

However, the Liquity system prohibits redemptions of certain troves, which can reduce the total redeemable amount:

1. **Under-collateralized troves**: If a trove's collateralization ratio drops below 100%, the system skips it during redemption to avoid decreasing its collateral ratio further. These troves should be liquidated instead.
2. **Unapplied redistribution debt**: Troves that have been fully redeemed can receive new debt via redistributions. This debt is not redeemable until `applyPendingDebt` is called on the trove or the trove owner syncs their position.

Both scenarios are considered unlikely under normal operating conditions. If they do happen, under-collateralized troves are expected to be liquidated promptly, and `applyPendingDebt` can be called permissionlessly by anyone to make redistribution debt redeemable again.

8.8 Redemption Fee Floor Must Account for Oracle Arbitrage

Note Version 1

`FXPriceFeed` relays prices that inherit Chainlink's deviation threshold behavior: a new Chainlink price is pushed if it moves beyond the configured threshold, or when the heartbeat time has passed.

Meanwhile, the protocol allows redemption fees as low as the `redemptionFeeFloor` set in `SystemParams`. Whenever the oracle underreports the true price by more than the minimum redemption fee, a stale feed can be exploited to redeem collateral tokens at a discount.

Consider the following example, assuming a minimum redemption fee of 0.5%:

1. The FX pair moves to a price that is 0.6% higher than the last oracle price.
2. An arbitrageur acquires debt tokens, calls `redeemCollateral()`, and receives collateral at the oracle price plus the redemption fee floor (0.5%). They can immediately sell the received collateral at spot. As the oracle is stale, the arbitrageur profits from the 0.1% difference (0.6% - 0.5%).

As a result, the `redemptionFeeFloor` should be set to a sufficiently high value based on the expected volatility of the FX pair, the Chainlink deviation threshold, and the heartbeat time. In case the volatility is high, a single price update can be larger than the deviation threshold. E.g., if the price moves by 1% in a single update, and the deviation threshold is set to 0.5%, the oracle price will change by 1%.

Note that the `FXPriceFeed` requires someone to trigger the price update after it has been pushed by Chainlink, by calling the ChainlinkRelayer's `relay` function. If this is not done immediately, it can lead to greater staleness.

8.9 Redemptions Are Less Punishing in Mento

Note Version 1

In Mento, redemptions are less punishing compared to Liquity. In Liquity, redemptions generally happen only when they are profitable to do, as the redeemer is the one paying for the redemption. However, in Mento, redemptions can also be used to rebalance an FPMM. In this case, the FPMM is paying to execute the redemption even if it is not profitable to do so. In fact, the rebalance incentive is expected to be used to pay the redeemer.

This changes the game theory of redemptions. In Liquity, trove owners generally want to avoid redemption, as they will most likely lose money. If getting redeemed is profitable, this will cause trove owners to choose a rate close to the lowest possible interest rate.

8.10 Stability Pool Users Can Dodge Bad Debt

Note Version 1

Liquidations are performed against the stability pool. Generally, liquidations are profitable. However, if the price of the collateral asset drops significantly, it is possible for liquidations to be unprofitable for the stability pool. In this case, stability pool users can avoid taking on bad debt by withdrawing their stake before the liquidation occurs. This is also the case in Liquity V2.

In Mento, this is more likely to happen in case the price drop occurs while the markets are closed (while the MarketHoursBreaker reverts). During this time, most actions in the Liquity system are blocked, including liquidations. However, stability pool users can still withdraw their stake, as the Stability Pool contract does not use the price oracle. As a result, stability pool users may remove their stake over a trading weekend in case they expect a significant price drop when the markets open again.