Security Assessment & Formal Verification *Final* Report

SiloCore v2

May 2025

Prepared for
Silo Team

Table of content

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Initial Commit Hash	Platform
silo-contracts-v2	https://github.com/silo-finance/silo-contracts-v2/tree/develop/silo-core	3db357d	8b56c53	EVM

Project Overview

This document describes the specification and verification of **silo contracts v2** using the Certora Prover and manual code review findings. The work was undertaken from **March 31st** to **April 7th 2025**

The following contract list is included in our scope:

`silo-core/contracts/*`

The Certora Prover demonstrated that the implementation of the **Solidity** contracts above is correct with respect to the formal rules written by the Certora team. In addition, the team performed a manual audit of all the Solidity contracts**.** During the verification process and the manual audit, the Certora team discovered bugs in the Solidity contracts code, as listed on the following page.

Please note that a few more formal rules are not included in this report, as they were proven with an unreleased version of the Certora Prover. Once those rules are proven on a released version of the Certora Prover, we will add them to the next version of this document.

Protocol Overview

Silo is a lending protocol between two assets. Each silo holds two assets that can be used as collateral to debt from either asset. Each half of the silo uses three share tokens to manage the debt, collateral and protected collateral of each user. Shares can be traded and are a wrapped ERC20

Findings Summary

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

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	1	1	1
Low	1	1	1
Informational	3	2	-
Total	5	4	2

Severity Matrix

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

certora

Silo

Detailed Findings

ID	Title	Severity	Status
M-01	Delpoyer can deny DAO funds through revert on transfer.	Medium	Fixed.
L-01	getCollateralAmountsWithInterest might underflow for huge total collateral assets.	Low	Fixed.

Medium Severity Issues

M-01 Delpoyer can deny DAO funds through revert on transfer.		
Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: Actions.sol	Status: **Fixed.**	

Description: The fix in PR-1086 does not adequately fix the issue raised by cantina #33.

The fix was to use OpenZepplin Address FunctionCall, which can still revert in transfer.

Recommendations: Wrap the `safeTransfer()` call that sends tokens to the `deployerFeeReceiver` in a `try-catch` block. Upon successful execution, proceed as normal. If the call fails, catch the revert and execute the `catch` block to send the `deployerFeeReceiver`'s fees to the DAO.

Customer's response: Properly fixed on PR#1176.

Low Severity Issues

L-01 getCollateralAmountsWithInterest might underflow for huge total collateral assets.

Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: SiloMathLib.sol	Status: **Fixed.**	

Description:

If the _collateralAssets is equal to max uint256 the collateral amount with interest will underflow.

```
70          unchecked { cap = type(uint256).max - _collateralAssets⬆ - 1; }
71
72          if (cap < collateralInterest) {
73              // avoid overflow on interest
74              collateralInterest = cap;
75          }
76
77          // safe to uncheck because of cap
78          unchecked {  collateralAssetsWithInterest = _collateralAssets⬆ + collateralInterest; }
```

Recommendations: Add a check to make sure such an underflow is not possible.

Customer's response: Properly fixed on PR#1159.

Informational Issues

I-01. Self Transfer of Share tokens might lead to unintended rewards, or interaction with hooks.

Description: Usually custom wrapping of ERC20 tokens block self transfer of tokens (where sender and receiver are the same address) to block potential issues with how hooks and rewards mechanisms are interfacing and incentivising interactions with the protocol.

Recommendation: Block the ability to transfer to the same address as the sender.

Customer's response: Acknowledged..

I-02. Adding nonReentrant to liquidationCall might not fully solve the double liquidation issue in the future.

Note: Currently this issue is not present in the code, the suggested recommendation is to future proof the contract against possible hard to find edge cases in the future.

Description: Currently adding nonReentrant to liquidation call solves the issue raised. In the future if any other contract interfacing with Silo has interaction with the solvency of a specific user, this reentrancy attack might still be active.

Recommendation: Maintain the locking cross contract reentrency guard. And write an "unsafe_repay" function that would be identical to the "repay" but without acquiring the locks. This would allow liquidationCall to maintain reentrency isolation.

Customer's response: Acknowledged would not fix.

I-03. applyFractions uses the pre-interest debt to check if additional accrument is necessary.

Note: This finding is still under active investigation; it's in this report to maintain clarity.
Note 2: Further investigation led to potential missed rewards. Impact analysis yielded that the missed rewards are significantly small and can be ignored.

Description: applyFraction might accrue additional fractional interest on debts that don't need that additional accrument, that might lead to issues with the math of how fractions are calculated.

Recommendation: Develop a Unit / Formal test to make sure that the math does not break on edge cases.

Customer's response: Acknowledged no issue.

certora

Silo

Formal Verification

Verification Notations

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

General Assumptions and Simplifications

- We work with objects inherited from the original contracts. In the inherited objects we add more view methods, flags, etc. In cases where it was not possible to collect the required information via the inherited object, we modify the original. E.g. we added flags to keep track whether some internal function has been called or not. These modifications don't affect the functionality of original contracts.
- We replaced some functions with equivalent CVL implementations. Notably *mulDiv* and most methods in *SiloConfig*. This speeds up the verification process.
  When possible, we used a simplified version of *SiloMathLib.convertToAssets*, *SiloMathLib.convertToShares* and *InterestRateModelV2.getCompoundInterestRate*. These overapproximate the originals, meaning that when a property is verified using the simplified method, it is also verified for the original implementation.

Formal Verification Properties

Silo

Module General Assumptions

- Any loop was unrolled to two iterations.
- The quoted price of any token, from any oracle, can either be 0.5, 1 or 3.
- We use basic standard ERC20 token implementations for the underlying Silo tokens.
- The sum of collateral assets never overflows, and neither the sum of assets and underlying token balances.
- “Actors” are excluded from being Silo contracts.

Module Properties

P-01. Integrity of state-changing methods	
Status: Verified	

Rule Name	Status	Description	Link to rule report
HLP_integrityOfBorrow **HLP_integrityOfBorrowSame** **HLP_integrityOfDeposit** **HLP_integrityOfMint** **HLP_integrityOfBorrowShares** **HLP_integrityOfRedeem** **HLP_integrityOfWithdraw**	Verified	*The methods update the state as expected.*	*Report*

P-02. Methods only affect the expected users

Status: Verified	

Rule Name	Status	Description	Link to rule report
HLP_BorrowDoesntAffectOthers **HLP_BorrowSameAssetDoesntAffectOthers** **HLP_BorrowSharesDoesntAffectOthers** **HLP_DepositDoesntAffectOthers** **HLP_MintDoesntAffectOthers** **HLP_RedeemDoesntAffectOthers** **HLP_RepayDoesntAffectOthers** **HLP_RepaySharesDoesntAffectOthers** **HLP_transitionCollateralDoesntAffectOthers**	Verified	*Balances of all users are unaffected by the method except for msg.sender and the users specified in methods parameters.*	*Report*

P-03. The protocol doesn't deny access to any user.

Status: Verified	

Rule Name	Status	Description	Link to rule report
RA_anyone_may_deposit **RA_anyone_may_repay** **RA_deposit_recipient_is_not_restricted**	Verified	*Any user may deposit in favor of anyone. Any user may repay anyone's debt.*	*Report*

P-04. Only specified methods may change important variables

Status: Verified

Rule Name	Status	Description	Link to rule report
noAccountChangesBeforeAccrue	Verified	*No external method changes the balance of any Silo token for any user without calling* `AccrueInterestForAsset` *first.*	*Report*
onlyAccrueCanChangeVars	Verified	*Only* `AccrueInterestForAsset` *can change* `daoAndDeployerRevenue`	*Report*

P-05. Risk assessment properties

Status: Verified

Rule Name	Status	Description	Link to rule report
RA_Silo_repay_all_shares	Verified	*A user has no debt after being repaid with max shares amount*	*Report*
PRV_user_assets_invariant_under_accrual_interest	Verified	*Any user shares value (converted to underlying assets) doesn't change when calling accrueInterest from both Silos.*	*Report*
PRV_LtV_invariant_under_accrual_interest	Verified	*The LtV of any user doesn't change when calling accrueInterest() from both Silos.*	*Report*

solventAfterSwitch	Verified	*User is solvent after switchCollateralToThisSilo()*	*Report*

P-06. Integrity of preview_and max_ methods

Status: Verified

Rule Name	Status	Description	Link to rule report
maxWithdraw_noGreaterThanLiquidity	Verified	*The result of maxWithdraw() should never be more than the liquidity of the Silo.*	*Report*
HLP_MaxDeposit_reverts	Verified	*Trying to deposit more than the result of maxDeposit always reverts.*	*Report*
HLP_PreviewBorrowCorrectness	Verified	*PreviewBorrow must overestimate the debt shares received.*	*Report*
HLP_PreviewBorrowSharesCorrectness	Verified	*PreviewBorrowShares must underestimate the assets received.*	*Report*

P-07. Customer suggested properties

Status: Verified

Rule Name	Status	Description	Link to rule report
accrueInterest_idempotent	Verified	*accrueInterest() calling twice is the same as calling once (in a single block).*	*Report*
solventChecked	Verified	*Solvency checked on the correct user on any change that implies more debt.*	*Report*, *Report*
transferWithChecksAlwaysOn	Verified	*The flag transferWithChecks is always on at the end of all public methods*	*Report*, *Report*
borrowerCollateralSilo_neverSetToZero	Verified	*if borrowerCollateralSilo[user] is set from zero to non-zero value, it never goes back to zero*	*Report*
noDebtInBothSilos	Verified	*It's not possible to have debt in both Silos.*	*Report*, *Report*
accrueInterestConsistency	Verified	*accrueInterestForBothSilos() is equal to calling silo0.accrueInterest() and silo1.accrueInterest()*	*Report*
	Implied	*repay() any user that can repay the debt should be able to repay the debt. Implied by RA_anyone_may_repay*	
	Implied	*repay() any other user than the borrower can repay. Implied by RA_anyone_may_repay*	
	Implied	*repayShares() should reduce only the debt of the borrower. Implied by HLP_repaySharesDoesntAffectOthers*	
	Implied	*repay() should reduce only the debt of the borrower. Implied by HLP_repayDoesntAffectOthers*	
	Implied	*borrowShares() should always increase debt shares of the borrower. Implied by HLP_integrityOfBorrowShares.*	

	Implied	*borrowShares() should always increase the balance of the receiver. Implied by HLP_integrityOfBorrowShares.*	
	Implied	*borrow() should always increase debt shares of the borrower. Implied by HLP_integrityOfBorrow*	
	Implied	*borrow() should always increase the balance of the receiver. Implied by HLP_integrityOfBorrow*	

P-08. Reentrancy guard integrity

Status: Verified

Rule Name	Status	Description	Link to rule report
RA_reentrancyGuardStaysUnlocked	Verified	*Reentrancy guard stays unlocked after every public method call.*	*Report*
RA_reentrancyGuardStatus_change	Verified	*After any call from a non-privileged address the status of reentrancy guard either stays 1 or stays greater than 1.*	*Report*
RA_reentrancyGuardChecked	Verified	*Every public method checks (loads) the reentrancy guard*	*Report*

Disclaimer

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

About Certora

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