Security Assessment & Formal Verification *Final* Report

SiloCore v2

November 2024

Prepared for
Silo Team

Table of content

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
silo-contracts-v2	https://github.com/silo-finance/silo-contracts-v2/tree/develop/silo-core	e538933	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 **November 4th** to **November 25th 2024**

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 for 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	0	-	-
High	2	2	2
Medium	2	2	1
Low	2	2	2
Informational	4	4	2
Total	10	10	7

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
H-01	Faulty hook location can lead to drains from legitimate hooks.	High	Fixed.
H-02	Max repay front run grift	High	Fixed
M-01	Transition Collateral fails when the user is insolvent.	Medium	Fixed
M-02	Silo router might be susceptible to advanced attack vectors	Medium	Redesign planned.
L-01	withdrawFees is not protected by the reentrancy guard.	Low	Fixed
L-02	_beforeTokenTransfer for debtToken does not reduce currentAllowance.	Low	Fixed
I-01	Borrow and Repay within the same block might Grief some actions for that block.	Informational	Acknowledged. Fix not needed.
I-02	Implement a view alternative to beforeQuote.	Informational	Acknowledged. Fix not needed.
I-03	LiquidationCall Might want to turn on the reentrancy guard.	Informational	Fixed
I-04	Possible gas optimization in accrueInterestForAsset	Informational	Fixed

High Severity Issues

H-01 Faulty hook location can lead to drains from legitimate hooks.		
Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: Actions.sol	Status: Not Fixed	Violated Property:

Note: This finding has the initial severity of Critical but its severity was lowered due to the customer's threat model.

Description: Many of the Actions inside Action.sol follow the following pattern:

1. **Check requirements.**
2. **Hook before the call.**
3. **Turn on the reentrancy guard**.

This pattern can be used to fully drain a silo given a hook configuration that allows re-entrency, for example, a hook that transfers an unrelated erc777 token would be exploitable with this attack.,

Exploit Scenario:

1. An attacker sends a borrow action to an exploitable silo.

```
117     function borrow(ISilo.BorrowArgs memory _args)
118         external
119         returns (uint256 assets, uint256 shares)
120     {
121         ISiloConfig siloConfig = ShareTokenLib.siloConfig();
122       IhorSF, 5 months ago • silo-core: borrow fn refactoring (#560) …
123
124         require(!siloConfig.hasDebtInOtherSilo(address(this), _args.borrower), ISilo.BorrowNotPossible());
125
126         _hookCallBeforeBorrow(_args, Hook.BORROW); // msg.sender.function
127
128         siloConfig.turnOnReentrancyProtection();
129         siloConfig.accrueInterestForBothSilos();
130         siloConfig.setOtherSiloAsCollateralSilo(_args.borrower);
131
```

2. The check in line 124 is passed, and then in line 126 the hook is triggered.
3. The attacker creates a new debt position by depositing and borrowing a significant position.
4. Line 130 sets the other silo as collateral as opposed to this silo, effectively freeing the locked collateral created in step 3.
5. The attacker borrows an insignificant amount in this silo, passing the solvency check for that insignificant debt.
6. The attacker withdraws the significant collateral, as it is no longer tied to any debt, draining the silo.

Recommendations: Throughout the codebase, it is recommended to hook before any checks for any method, as well as to move all checks to be within the re-entrency guarded block of code.

Perhaps through the use of modifiers, the API can be simplified by adding a hookable modifier and a reentrecy guarded modifier.

Customer's response: Acknowledged.

Fix Review: Properly fixed https://github.com/silo-finance/silo-contracts-v2/pull/845.

H-02 Max repay front run grift		
Severity: **High**	Impact: **Medium**	Likelihood: **High**
Files: Actions.sol	Status: Not Fixed	Violated Property:

Description: The repay action can be front-run, by repaying the dust amount. The function itself would try to overpay more than the actual debt present, causing a reversion. This can effectively make the repay API almost unusable.

Exploit Scenario:

- Alice has a large debt that she wishes to repay.
- Eve front runs and repays a single share of Alice's debt.
- Alice reverts.

Recommendations: In SiloLendingLib, if a repay is requested for more shares than the balance of a given borrower, cap the shares to the borrower maximum.

Note:

- The assets might need to be recalculated in that case.
- In the case of liquidation, some leftover assets might need to be returned to the msg.sender in that case.

Customer's response: Acknowledged.

Fix Review: Properly fixed https://github.com/silo-finance/silo-contracts-v2/pull/847.

Medium Severity Issues

M-01 Transition Collateral fails when the user is insolvent.		
Severity: **Medium**	Impact: **Medium**	Likelihood: **High**
Files: Actions.sol	Status: Not Fixed	Violated Property:

Description: transitionCollateral uses the wrong solvency check in cases where the user is insolvent but through a different silo.

Exploit Scenario: N/A

Recommendations: use siloConfig.getConfigsForWithdraw in order to determine if solvency check is needed.

Customer's response: Acknowledged.

Fix Review: Properly fixed https://github.com/silo-finance/silo-contracts-v2/pull/846.

M-02 Silo router might be susceptible to advanced attack vectors

Severity: **Medium**	Impact: **Medium**	Likelihood: **Low**
Files: SiloRounter.sol	Status: Not Fixed	Violated Property:

Description: SiloRounter Calls user-provided address. Because this contract is used by every silo user, it is preferred to add a couple checks to make sure that advanced attacks on this contract are impossible.

Note: No exploit vector was found for this bug.

Recommendations:

1. Use the factories isSilo to tie the router with a specific factory, making sure that all provided silos at least were created by the factory.
2. Create a unique router for each silo pair.
3. Register each siloConfig with the router, and check that at no point the router is left with debtShare tokens of any of the registered silos.

Customer's response: Acknowledged

Fix Review: Router will be redesigned and current code will not be used.

Low Severity Issues

L-01 withdrawFees is not protected by the reentrancy guard.		
Severity: **Low**	Impact: **Low**	Likelihood: **high**
Files: silo.sol	Status: Not Fixed	Violated Property:

Description: withdrawFees should use the reentrancy guard.

Exploit Scenario: N/A

Recommendations: turn the reentrancy guard during the execution of withdrawFees.

Customer's response: Acknowledged.

Fix Review: Properly fixed https://github.com/silo-finance/silo-contracts-v2/pull/849

L-02 _beforeTokenTransfer for debtToken does not reduce currentAllowance.

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: ShareDebtToken.sol	Status: Not Fixed	

Description: When this token is entered through transferWithNoChecks the currentAllowance is never reduced, This might limit some hooks implementations.

Exploit Scenario: N/A

Recommendations: If this is a feature you wish hooks to be able to use, you might want to change the API or allow for hooks to reduce allowance as part of transfers.

Customer's response: Acknowledged.

Fix Review: Properly fixed by adding callOnBehalfOfShareToken for hooks https://github.com/silo-finance/silo-contracts-v2/pull/848

Informational Severity Issues

I-01. Borrow and Repay within the same block might grief some actions for that block.

Description: Currently there is no cost (except gas) to borrow and repay within the same block, this would hog all the liquidity for that block.

Recommendation: This is only relevant for small silos. If you wish to fix this behavior perhaps not allowing for borrowed funds to be transferred or redeemed for one block would be able to stop this.

Customer's response: Acknowledged. The issue is not present in a healthy market with good liquidity.

I-02. Implement a view alternative to beforeQuote.

Description: current view methods have no way to make sure that the oracle is not in the middle of a different operation, which might yield faulty results from the quote.
Recommendation: A view method that reads the reentrancy flag from the oracle and reverts if it is set would fix this.
This can also be fixed by the Oracle providers, it's best to at least comment on this topic in the docs or the code.

Customer's response: Acknowledged. This issue can be resolved by implementing required logic in quote() function itself when needed.

I-03. LiquidationCall Might want to turn on the reentrancy guard.

Description: LiquidationCall follows a pattern of

1. Calculate values.
2. Repay debt.
3. Receive reward.

Because There could be a discrepancy between the calculation done in step 1 and the reward received in step 3, this area might be sustainable for reentrancy attack, (as seen on the audit Certora did on April 2024).

Note: Currently no attack vector was found.
Recommendation: it might be possible to make the function safer by following the checks-effects-interactions pattern.

1. Turn On the reentrancy guard.
2. Calculate all the relevant values.
3. Transfer with no checks.
4. Turn Off the reentrancy guard.
5. Repay.
6. Redeem.

Customer's response: Acknowledged.
Fix Review: Properly fixed https://github.com/silo-finance/silo-contracts-v2/pull/851

I-04. Possible gas optimization in accrueInterestForAsset

Description: accrueInterestForAsset Is checking the init condition before the more common multiple accruments within the same block. Because we expect the initial check to only trigger once, but the latter check to trigger multiple times per block, by changing the order we might be able to save on some gas.

Recommendation: switch the lines

1. if (lastTimestamp == 0)
2. if (lastTimestamp == block.timestamp)

Customer's response: Acknowledged.
Fix Review: Properly fixed https://github.com/silo-finance/silo-contracts-v2/pull/852

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.
Implied	The property is implied by some other verified properties..
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 some 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.
- We further assume that these properties hold in all reachable states :
  a. silo0Storage.protected collateral + silo0Storage.collateral <= token0.TotalSupply and the same for token1
  b. debtToken0.totalSupply() == 0 <=> silo0.Storage[Debt] == 0 and the same for the other tokens

c. if a silo has assets then interestRateTimestamp is > 0
d. There isn't more shares than assets, i.e. *silo0.totalSupply() <= silo0.Storage[Collateral]*

Formal Verification Properties

Silo

Module General Assumptions

- Any loop was unrolled to two iterations.
- The quoted price of any token, from any oracle is 1.
- We use basic standard ERC20 token implementations for the underlying Silo tokens.
- "Actors" are excluded from being Silo contracts. I.e., we verify that the properties cannot be violated by an external user.

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_integrityOfRepayShares** **HLP_integrityOfRepay** **HLP_integrityOfRedeem** **HLP_integrityOfWithdraw**	Verified	*The methods update the state as expected.*	*Report* *Report* *Report* *Report* *Report* *Report* *Report* *Report* *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**	Verified	*Balances of all users are unaffected by the method except for msg.sender and the users specified in methods parameters.*	*Report Borrow* *Report BorrowSame* *Report BorrowShares* *Report Deposit* *Report Mint* *Report Redeem* *Report Repay* *Report RepayShares*

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** **RA_repay_borrower_is_not_restricted**	Verified	*Any user may deposit in favor of anyone. Any user may repay anyone's debt.*	*Report* *Report* *Report* *Report*

P-04. Only specified methods may change important variables

Status: Verified	

Rule Name	Status	Description	Link to rule report
howTimeStampChanges	Verified	`InterestRateTimestamp` *never decreases.*	*Report*
whoCanChangeShareTokenTotalSupply	Verified	*Any increase / decrease of* `CollateralShareToken.TotalSupply()` *may only be caused by a method with increase / decrease privilege.*	*Report*
whoCanDecreaseRevenue	Verified	*Only specified methods may decrease* `daoAndDeployerRevenue`	*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_int	Verified	*Any user shares value (converted to underlying assets) doesn't change when calling accrueInterest from both Silos.*	*Report*

erest			
PRV_LtV_invariant_under_accrual_interest	Verified	*The LtV of any user doesn't change when calling accrueInterest() from both Silos.*	*Report*

P-06. Integrity of Max methods

Status: Verified

Rule Name	Status	Description	Link to rule report
maxRepay_burnsAllDebt	Verified	*Repaying with maxRepay() value burns all user share debt token balance*	*Report*
HLP_MaxRedeem_noGreaterThanBalance	Verified	*The result of maxRedeem() should never be more than share token balanceOf user*	*Report*
maxWithdraw_noGreaterThanLiquidity	Verified	*The result of maxWithdraw() should never be more than the liquidity of the Silo.*	*Report*
HLP_MaxRepayShares_reverts	Verified	*Trying to repayShares with more than the result of MaxRepayShares always reverts.*	*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*
noDebtInBothSilos	Verified	*It's not possible to have debt in both Silos.*	*Report*, *Report*
solventChecked	Verified	*Solvency checked on the correct user on any change that implies more debt.*	*Report*
getDebtAmountsWithInterest_correctness	Verified	*getDebtAmountsWithInterest() never returns lower value for debtAssetsWithInterest than _totalDebtAssets input*	*Report*
borrowerCollateralSilo_setNonzeroIncreasesBalance	Verified	*if borrowerCollateralSilo[user] is set from zero to non-zero value, user must have balance in one of debt share tokens - excluding switchCollateralToThisSilo() method*	*Report*
borrowerCollateralSilo_neverSetToZero	Verified	*if borrowerCollateralSilo[user] is set from zero to non-zero value, it never goes back to zero*	*Report*
	Implied	*The result of previewBorrow() should be equal to the result of borrow(). Implied by HLP_PreviewBorrowCorrectness_strict*	
	Implied	*The result of previewMint() should be equal to the result of mint(). Implied by HLP_PreviewMintCorrectness_strict*	
	Implied	*The result of previewWithdraw() should be equal to the result of withdraw(). Implied by HLP_PreviewWithdrawCorrectness_strict*	
	Implied	*The return value of previewRepay() should be always equal to repay(). Implied by HLP_PreviewRepayCorrectness_strict*	
	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 decrease the debt. Implied by HLP_integrityOfRepayShares*	
	Implied	*repayShares() should reduce only the debt of the borrower. Implied by HLP_repaySharesDoesntAffectOthers*	
	Implied	*repayShares() should not be able to repay more than maxRepayShares. Implied by HLP_MaxRepayShares_reverts*	
	Implied	*repay() should decrease the debt. Implied by HLP_integrityOfRepay*	
	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-8. Integrity of Preview methods

Status: Verified

Rule Name	Status	Description	Link to rule report
HLP_PreviewBorrowCorrectness	Implied	*PreviewBorrow must overestimate the debt shares received. Implied by HLP_PreviewBorrowCorrectness_strict*	
HLP_PreviewBorrowSharesCorrectness	Verified	*PreviewBorrowShares must underestimate the assets received.*	*Report*
HLP_PreviewDepositCorrectness	Verified	*PreviewDeposit must underestimate the collateral shares received.*	*Report*
HLP_PreviewMintCorrectness	Implied	*PreviewMint must overestimate the assets paid. Implied by HLP_PreviewMintCorrectness_strict*	
HLP_PreviewRedeemCorrectness	Verified	*PreviewRedeem must underestimate the assets received.*	*Report*
HLP_PreviewRepayCorrectness	Implied	*PreviewRepay must underestimate the debt shares removed. Implied by HLP_PreviewRepayCorrectness_strict*	
HLP_PreviewRepaySharesCorrectness	Verified	*PreviewRepayShares must overestimate the assets paid.*	*Report*
HLP_PreviewWithdrawCorrectness	Implied	*PreviewWithdraw must overestimate the collateral shares paid. Implied by HLP_PreviewWithdrawCorrectness_strict*	

HLP_PreviewMintCorrectness_strict	Verified	*PreviewMint tells the exact amount of assets paid*	*Report*
HLP_PreviewWithdrawCorrectness_strict	Verified	*PreviewWithdraw tells the exact amount of the collateral shares paid.*	*Report*
HLP_PreviewBorrowCorrectness_strict	Verified	*PreviewBorrow tells the exact amount of the debt shares received.*	*Report*
HLP_PreviewRepayCorrectness_strict	Verified	*PreviewRepay tells the exact amount of the debt shares removed.*	*Report*

P-09. Reentrancy guard integrity

Status: **Violated**
Reported Bug: L-01

Rule Name	Status	Description	Link to rule report
RA_reentrancyGuardStaysUnlocked	Verified	*Reentrancy guard stays unlocked after every public method call.*	*Report*
RA_reentrancyGuardStatus_change	Violated	*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*, *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.