Security Assessment

Final Report

Silo Router

February 2025

Prepared for Silo

Table of Contents

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
silo-contracts-v2	https://github.com/silo-finance/silo-contracts-v2/tree/feature/silo-router-for-all	69998658	EVM

Project Overview

This document describes the manual review of the “silo-core: Silo router redesign” Pull Request. The work was undertaken from 04.02.25 to 10.02.25.

The following contract list is included in our scope:

```
silo-core/contracts/silo-router/SiloRouter.sol
silo-core/contracts/silo-router/SiloRouterImplementation.sol
```

The team performed a manual audit of all the smart contracts. During the manual audit, the Certora team discovered bugs in the code, as listed on the following page.

Protocol Overview

Silo Router is a utility contract that aims to improve UX. It can batch any number or combination of actions (Deposit, Withdraw, Borrow, Repay) and execute them in a single transaction.

Threat model

The `SiloRouter` and `SiloRouterImplementation` contract are UX utilities. While the user can interact with the protocol either through the Router or directly with the Silo Vaults, the level of risk for these present contracts is expected to be fairly low. Indeed, these contracts aren’t expected to hold any funds, and the users are expected to not leave left-over funders. Users aren’t expected to interact with the Router outside of the official UI for the suggested flows. Additionally, the important protections already exist at the Silo Vault.

certora

Silo

Findings Summary

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

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

Severity Matrix

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

Informational Severity Issues

I-01 Type Confusion: withdraw Returns Shares, Not Assets

Severity: **Informational**	Impact: **N/A**	Likelihood: **N/A**
Files: SiloRouterImplementation.sol	Status: Fixed	

Description:

- The function `_silo.withdraw` returns `shares`, but the router treats it as `assets`, which may lead to logic errors.
- This issue exists at SiloRouterImplementationFlat.sol#L2155-L2162

```
File: SiloRouterImplementationFlat.sol
2155:     function withdraw(
2156:         ISilo _silo,
2157:         uint256 _amount,
2158:         address _receiver,
2159:         ISilo.CollateralType _collateral
2160:     ) external payable virtual returns (uint256 assets) {
2161:         assets = _silo.withdraw(_amount, _receiver, msg.sender,
_collateral);  //@audit-issue this returns shares, not assets
2162:     }
```

As proof, see the interface:

- https://github.com/silo-finance/silo-contracts-v2/blob/3b4c23ca8872e90f29ffadb84968c00d2a916a35/silo-core/contracts/interfaces/ISilo.sol#L298-L300

```
File: ISilo.sol
298:     function withdraw(uint256 _assets, address _receiver, address _owner,
CollateralType _collateralType)
299:         external
300:         returns (uint256 shares);
```

And see the function itself:

- https://github.com/silo-finance/silo-contracts-v2/blob/3b4c23ca8872e90f29ffadb84968c00d2a916a35/silo-core/contracts/Silo.sol#L230

```
File: Silo.sol
230:     function withdraw(uint256 _assets, address _receiver, address _owner)
231:         external
232:         virtual
233:         returns (uint256 shares)
```

Recommendation:
To gain clarity, change `_amount -> _assets` and `assets -> shares`

```
File: SiloRouterImplementationFlat.sol
2155:     function withdraw(
2156:         ISilo _silo,
- 2157:         uint256 _amount,
+ 2157:         uint256 _assets,
2158:         address _receiver,
2159:         ISilo.CollateralType _collateral
- 2160:     ) external payable virtual returns (uint256 assets) {
+ 2160:     ) external payable virtual returns (uint256 shares) {
- 2161:         assets = _silo.withdraw(_amount, _receiver, msg.sender,
_collateral);
+ 2161:         shares = _silo.withdraw(_assets, _receiver, msg.sender,
_collateral);
2162:     }
```

Alternatively, if what was meant was a `redeem()` instead of a `withdraw`, consider changing the function's name

Note: `withdrawAll` is fine as it actually calls `redeem`. However, to avoid any confusion, it'd be great to actually call it `redeemAll` instead

Silo's Response: Fixed, variable is renamed for clarity.

Certora's Response: Fix is confirmed at commit 584ef754

I-02 sendValueAll/transferAll shouldn't be up to the user to call

Severity: **Informational**	Impact: **N/A**	Likelihood: **N/A**
Files: SiloRouterImplementation.sol	Status: Acknowledged (Design Choice)	

Description:

- The router **relies on users** to call `sendValueAll`/`transferAll` correctly, but missing these calls can lead to loss of funds.
- Bots can sweep leftover funds if the user fails to execute all required actions in a single transaction.

Recommendation:

- The router UX should **automatically** append `sendValueAll`/`transferAll` for all assets withdrawn to the contract.
- Add clear comments in the code to ensure this behavior is enforced.

Silo's Response: UX will pack all necessary actions for the user. SiloRouter is an UI tool, it should not be used outside protocol website to avoid actions misconfiguration. Invalid sequence of actions may cause the loss of funds (even in ERC-20 tokens). This is a design choice.

Certora's Response: Acknowledged.

I-03 DOS or Leftover Funds Due to Fixed Amounts in unwrap/transfer/sendValue

Severity: **Informational**	Impact: **N/A**	Likelihood: **N/A**
Files: SiloRouterImplementation.sol	Status: Acknowledged (Design Choice)	

Description:

- The `withdraw -> unwrap -> sendValue` sequence requires knowing `shares` in advance, but `shares` fluctuate due to on-chain conditions.
- Failure to handle fluctuations correctly can result in stuck funds (which bots may steal) or transaction reverts.
- Attackers can manipulate exchange rates via frontrunning to cause failure.

Recommendation:

- Use `unwrapAll`, `sendValueAll`, and `transferAll` to avoid stuck funds.

Reference: [EIP-4626 Security Considerations]

> The methods totalAssets, convertToShares and convertToAssets are estimates useful for display purposes, and do not have to confer the exact amount of underlying assets their context suggests.
>
> The preview methods return values that are as close as possible to exact as possible. For that reason, they are manipulable by altering the on-chain conditions and are not always safe to be used as price oracles. This specification includes convert methods that are allowed to be inexact and > therefore can be implemented as robust price oracles. For example, it would be correct to implement the convert methods as using a time-weighted average price in converting between assets and shares.
>
> Integrators of EIP-4626 Vaults should be aware of the difference between these view methods when integrating with this standard. Additionally, note that the amount of underlying assets a user may receive from redeeming their Vault shares (previewRedeem) can be

> significantly different than the amount that would be taken from them when minting the same quantity of shares (previewMint). The differences may be small (like if due to rounding error), or very significant (like if a Vault implements withdrawal or deposit fees, etc). Therefore integrators should always take care to use the preview function most relevant to their use case, and never assume they are interchangeable.

Silo's Response: Design choice. There must be no left-overs on a router contract after user's transaction by design. To get all left-overs, users must execute transferAll, unwrapAll, sendValueAll actions.
User's transaction may revert if the sequence of actions is invalid in the system (for example, an attempt to withdraw non-existing collateral). Comments in SiloRouterImplementation.sol advise the preferred ways to interact with SiloRouter.

Certora's Response: Acknowledged.

I-04 No Slippage Protection

Severity: **Informational**	Impact: **N/A**	Likelihood: **N/A**
Files: SiloRouterImplementation.sol	Status: Acknowledged	

Description:

- EIP-4626 warns against direct EOA deposits/withdrawals without slippage protection.
- Transactions without slippage limits could suffer losses due to price changes.
- A deadline should also be set to prevent execution delays affecting outcomes.

Recommendation:

- Add a function that allows users to set slippage tolerance on deposits/withdrawals.
- Implement transaction deadlines.

Reference: EIP-4626 Security Considerations

> If implementors intend to support EOA account access directly, they should consider adding an additional function call for deposit/mint/withdraw/redeem with the means to accommodate slippage loss or unexpected deposit/withdrawal limits, since they have no other means to revert the transaction if the exact output amount is not achieved.

Silo's Response: No impact. Share-to-assets ratio can be changed only with interest rate which is limited to 10k% APR (<0.0004% / second in the worst case scenario). Silo has a protection from the first depositor and other ERC4626 specific attacks to inflate the rates. We don't expect the lending markets to have slippage.

Certora's Response: Acknowledged.

I-05 sendValueAll/transferAll shouldn't be up to the user to call

Severity: **Informational**	Impact: **N/A**	Likelihood: **N/A**
Files: SiloRouterImplementation.sol	Status: Acknowledged	

Description:

- Today, wrapped native tokens (e.g., WETH) are always 1:1 with the native currency, but this might not hold true for future implementations.
- Example: Base L2's native ETH is yield-bearing, meaning we could think of yield-bearing tokens like Base's ETH, which might shift away from a 1:1 ratio if yield distribution is introduced. , then the scenarios involving `wrap`/`unwrap` could be wrong (an`approveAll()` method would be needed here).

Recommendation:

- Future-proof the router by considering scenarios where `wrap`/`unwrap` is not strictly 1:1.

Silo's Response: Acknowledged but we consider this an impossible likelihood.

Certora's Response: Acknowledged.

Dismissed Concerns

Repayments Paused While Liquidations Enabled

On the Router, there's a `Pauseable` system. However, no other pause mechanisms were seen in the protocol. This means that, a state where repayments are paused but liquidations are enabled is possible, which is highly unfair.
Granted, users could directly call the Silo Vault to repay: but does the UI account for that or do users need to be tech-savvy?
Either the `Pauseable` system should be more granular, or there should be a sync to pause liquidations in the meantime. However, even if those are sync'd, given the competitive nature of liquidations (using Bots), the average users would get immediately liquidated once the Silo Router gets unpaused.

Silo's Response: "Rejected. It is a safety feature to pause a particular router contract, not the entire protocol. Pausing helps to migrate from the old router contract and abandon user's approvals during migration to the new router contract.
Other protocol contracts will keep working, repayments can be done by interacting with Silo contract directly."

Easily Bypassable Pause Mechanism

If an attacker deploys their own copy-pasted `SiloRouter`, they will still be able to interact with the protocol while other users would be blocked.
Even without the `SiloRouter`, it's also very possible to interact with the Silo Vaults directly. This raises the question: why is there a pause mechanism in place? It can make sense if an issue is found on the Silo Router itself, but not on Silo (which is directly accessible). Tech-savvy users aren't effectively protected/paused. This here is as effective as deactivating a button in the frontend (which would cost less gas thanks to not having to read the `paused` state variable). So: shouldn't the Pause Mechanism here only be on the front-end? (greying-out a button?)

As an additional note: the reentrancy guard can also be bypassed by an attacker deploying their own Silo Router, simply by not putting up a ReentrancyGuard. Therefore, this is probably also only consuming additional gas.

Silo's Response: "Rejected. Router is an UI utility contract to simplify complicated interactions with the protocol. Pausing is a safety feature to abandon user's approvals when we migrate to the new version.
When paused, all router actions can be done in Silo contract directly, it is intended."

Repayment/deposits will revert with Fee-On-Transfer tokens

Some tokens take a transfer fee (e.g. STA, PAXG), some do not currently charge a fee but may do so in the future (e.g. USDT, USDC).

The following logic, after `transferFrom`, may actually retain a balance of token that is less than `repayAmount` in the contract, meaning that the approval would approve more tokens than the contract's balance, and the call to `repay` would revert due to trying to move more funds than the Router's actual balance.

```
File: SiloRouterImplementation.sol
178:     function repayAll(ISilo _silo) external payable virtual returns
(uint256 shares) {
179:         uint256 repayAmount = _silo.maxRepay(msg.sender);
180:         IERC20 asset = IERC20(_silo.asset());
181:
182:         transferFrom(asset, address(this), repayAmount);
183:         approve(asset, address(_silo), repayAmount);
184:
185:         shares = repay(_silo, repayAmount);
186:     }
```

This is also a problem in the flow of deposit token using `SiloRouter.multicall`:

```
File: SiloRouterImplementation.sol
18: - deposit token using SiloRouter.multicall
19:     SiloRouter.transferFrom(IERC20 _token, address _to, uint256 _amount)
20:     SiloRouter.approve(IERC20 _token, address _spender, uint256 _amount)
21:     SiloRouter.deposit(ISilo _silo, uint256 _amount)
```

Remediation:

It'd be great to have an `approveAll` function that would use the actual balance of the contract and return it, so we can know how much to repay. A `depositAll` function would also be relevant in this scenario

Silo's Response: Rejected. Silo protocol does not support fee-on-transfer tokens.

Repayment could be made to revert

When repayments are over-repaid: the transaction reverts. If the repayment is close to the limit, say leaving 10% in, an attacker could frontrun the multicall by directly repaying on the Silo Vault on behalf of the target user, for about 11%. While economically expensive, this is still an open vector that could be exploited and might even be profitable for an attacker griefing a competitor's `multicall`.
An easy fix would be to add an upperbound to the repayment by checking `_silo.maxRepay(msg.sender)` and only repaying up to the maximum repay amount, and not possibly more.

Silo's Response: Rejected. To repay all users must use the repayAll function.

No direct mint/redeem: is that a design decision?

We have a deposit and a withdraw (which both takes `assets` as input and output shares). Besides the `redeem` on all shares: there isn't a way to specify the number of shares to redeem or to mint. Was that forgotten or on purpose? (design decision?)

Silo's Response: Rejected. It is a design choice. UX requirements do not contain this feature. SiloRouter is a minimalistic contract for the UI purpose.

Owner can renounce while system is paused

The contract `owner` is not prevented from renouncing the ownership while the contract is paused.

Silo's Response: Rejected. It is a feature to abandon old SiloRouter contract to migrate for a new version. Owner will pause contract and renounce ownership to prevent it to become live ever again in the future.

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.

Formal Verification Properties

SiloRouter

P-01. Integrity of pausing and ownership

Status: Verified

Rule Name	Status	Description	Link to rule report
consistencyOfPausing	Verified	*After calling pause(), all calls to multicall() will revert.*	*Run link*
onlyOwnerCanPause	Verified	*pause() must revert if the caller is not the owner.*	*Run link*
onlyOwnerCanUnpause	Verified	*unpause() must revert if the caller is not the owner.*	*Run link*

SiloRouterImplementation

Module General Assumptions

We assume that the Silo contract works correctly. These rules prove that methods on *SiloRouterImplementation* pass their arguments correctly to methods on *Silo*, i.e. that specified properties of *SIlo* apply to *SiloRouterImplementation* as well.

Module Properties

P-02. Methods' inverses

Status: Verified

Rule Name	Status	Description	Link to rule report
depositWithdrawInverse	Verified	*Calling deposit(); withdraw(); has no effect.*	*Run link*
borrowRepayInverse	Verified	*Calling borrow(); repay(); has no effect.*	*Run link*

P-03. Methods don't affect others

Status: Verified

Rule Name	Status	Description	Link to rule report
borrowDoesntAffectOthers	Verified	*borrow() doesn't affect balances of unrelated users.*	*Run link*
depositDoesntAffectOthers	Verified	*deposit() doesn't affect balances of unrelated users.*	*Run link*

repayDoesntAffectOthers	Verified	*repay() doesn't affect balances of unrelated users.*	*Run link*
withdrawDoesntAffectOthers	Verified	*withdraw() doesn't affect balances of unrelated users.*	*Run link*

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.