Security Assessment
Final Report

Veda Yield Streaming

December 2025

Prepared for Veda

Table of content

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Veda	https://github.com/Veda-Labs/boring-vault/tree/feat/accountant-tests	568d946 (initial commit)	Solidity

Project Overview

This document describes the manual code review findings of **Veda**. The following contract list is included in our scope:

```
src/base/Roles/TellerWithMultiAssetSupport.sol
src/base/Roles/TellerWithYieldStreaming.sol

src/base/Roles/AccountantWithRateProvider.sol
src/base/Roles/AccountantWithYieldStreaming.sol
```

The work was undertaken from **December 25, 2025,** to **January 5, 2026**. During this time, Certora's security researchers performed a manual audit of all the Solidity contracts and discovered several bugs in the codebase, which are summarized in the subsequent section.

Protocol Overview

Veda is a cross-chain, multi-asset vault protocol designed to manage deposits, yield, and liquidity modularly, leaning on its trusted roles for executing sensitive operations. At its core, it combines its `BoringVault` with extensible user-facing `Teller` contracts, yield integrations with external protocols, cross-chain messaging powered by LayerZero and dynamic exchange rates, following the yield generation.

Assessment Methodology

Our assessment approach combines design level analysis with a deep review of the implementation to ensure that a protocol is secure, economically sound, and behaves as intended under realistic conditions.

At the design level, we evaluate the architecture, the economic assumptions behind the protocol, and the safety properties that should hold independently of a specific chain or environment. This process includes reviewing internal and cross protocol interactions, state transition flows, trust boundaries, and any mechanism that could be exploited to extract value, deny service, or alter core system behavior. At this stage, a focused threat modelling exercise helps identify key attack surfaces and adversarial capabilities relevant to the system. Design level issues often relate to incentive structures, governance implications, or systemic behavior that emerges under adversarial conditions.

Implementation analysis focuses on the concrete behavior of the code within the execution model of the target chain. This involves reviewing the correctness of logic, access control, state handling, arithmetic behavior, and the nuanced behaviors of the chain environment. Familiar classes of vulnerabilities such as reentrancy conditions, faulty permission checks, precision issues, or unsafe assumptions often surface at this layer. These findings require context aware reasoning that takes into account both the code and the architectural intent.

To support this analysis, the codebase is examined through repeated manual passes and supplemented by automated tools when appropriate. High-risk logic areas receive deeper scrutiny, invariants are validated against both design intent and actual implementation, and potential vulnerability leads are thoroughly investigated. Automated techniques such as static analysis, fuzzing, or symbolic execution may be used to complement manual review and provide additional insight.

Collaboration with the development team plays an important role throughout the audit. This helps confirm expected behaviors, clarify design assumptions, and ensure an accurate understanding of the protocol's intended operation. All findings are documented with clear reasoning, reproducible examples, and actionable recommendations. A follow up review is conducted to validate the applied fixes and verify that no regressions or secondary issues have been introduced.

Findings Summary

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

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	-	-	-
Low	2	2	0
Informational	4	4	0
Total	6		

Severity Matrix

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

Detailed Findings

ID	Title	Severity	Status
L-01	`postLoss()` calls can truncate already vested small amounts of assets, potentially compounding into bigger losses	Low	Acknowledged
L-02	Users can avoid losses by frontrunning `postLoss()` with a withdrawal	Low	Acknowledged
I-01	Inconsistent style on `postLoss()`	Informational	Acknowledged
I-02	`SupplyObservation.cumulativeSupply` might eventually overflow	Informational	Acknowledged
I-03	A large deposit prior to posting a loss can result in losses larger than 1% of the current supply which don't pause the protocol	Informational	Acknowledged
I-04	Virtual calculation of fees based on current supply may result in the "over-calculation" of the accumulated fees resulting in less vested yield	Informational	Acknowledged

Low Severity Issues

L-01 postLoss() calls can truncate already vested small amounts of assets, potentially compounding into bigger losses

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: [AccountantWithYieldStreaming.sol]()	Status:	

Description:

The currently introduced virtual price was meant to tackle the problem of having too small vested amounts of assets over a time period, leading to direct losses of share price.

It currently does so, considering normal scenarios where small vestings compound inside of the virtual price, until they are large enough to get reflected on the real price, used by the rate.

However, those values can get truncated during loss recordings inside postLoss(), because the function derives the new virtual price from the current price, creating a discrepancy:

```javascript
uint128 cachedSharePrice = vestingState.lastSharePrice;

lastVirtualSharePrice = (totalAssets() - principalLoss).mulDivDown(RAY, currentShares);

vestingState.lastSharePrice = _calculateSharePriceFromVirtual();

uint256 lossBps =
    uint256(cachedSharePrice - vestingState.lastSharePrice).mulDivDown(10_000,
cachedSharePrice);

//verify the loss isn't too large
if (lossBps > maxDeviationLoss) {
    accountantState.isPaused = true;
    emit Paused();
```

```
}
```

As it can be seen, we derive the new virtual price from the new assets after the loss. For that we call totalAssets(), which are derived from lastSharePrice, thus these compounding assets are permanently lost.

Exploit Scenario:

For example, consider the following scenario (trying to keep it as realistic as possible):

```javascript
Starting state:
- totalAssets = 1_000_000e6 USDC
- totalSupply = 1_000_000e18 shares
- starting price = 1e6 (has to be in the same decimals as the asset)
- virtual price = 1e6 * 1e27 / 1e18 = 1e15

APY is set to 10% a year -> 100_000e6 assets over 1 year
-> 3168 wei of assets per second

Let's say derived from this, they vest for 3 days with this rate
1 second passes, someone invokes an update to the exchange rate:

1. lastVirtualSharePrice = 1e15 + (3168 * 1e27 / 1_000_000e18) =
= 1e15 + 3168 * 1e3 = 1000000003168000
2. lastSharePrice = 1000000003168000 * 1e18 / 1e27 = 1000000.003168

Thus, the lastSharePrice rounds to 1000000 and those
3168 wei starts compounding until it is large enough

Now assume a call to postLoss(), where the value doesn't matter.
What matters is the line:
lastVirtualSharePrice = (totalAssets() - principalLoss).mulDivDown(RAY, currentShares)

which in our example would be:
totalAssets = lastSharePrice * 1_000_000e18 / 1e18 = 1e6 * 1_000_000
lastVirtualSharePrice = (1e12 - loss) * 1e27 / 1_000_000e18
```

```
Even with a minimal loss, the resulting number will be less than
the previous virtual price of 1000000003168000, since the result
of the above calculation can at most be 1e15 if the loss is 0

We effectively override the 3168 wei we had.
```

This is directly dependent on both the frequency of exchange rate updates, APY, if any losses get recorded while we have compounding assets in the virtual price, etc, but it does create an edge-case scenario where data is lost, since the share price will stop reflecting all available assets.

Recommendations:

Calculate the difference between lastVirtualSharePrice and lastSharePrice, and add it back after adjusting the price upon postLoss().

Customer's response:

Acknowledged. Given that sub-wei precision is not that critical upon postLoss(), the team has decided not to implement this fix.

L-02 Users can avoid losses by frontrunning postLoss() with a withdrawal		
Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: [AccountantWithYieldStreaming.sol]()	Status: Acknowledged	

Description:

AccountantWithYieldStreaming::postLoss() instantly decreases the share price. This design, along with the fact that user withdrawals are processed instantly, allows users to dodge losses by frontrunning postLoss() calls with a withdrawal - shifting the burden of the loss to the remaining users.

Exploit Scenario:

Users can avoid losses the following way:

- Assume a vault with 2 shareholders (Alice and Bob), each with 1,000 shares (total 2,000 shares)
- The vault currently has 20,000 assets, i.e. each share is worth 10 assets
- The vault suffers a loss of 2,000 assets (decreasing total assets to 18,000)
- Under normal circumstances, the new positions will be worth:
  - Shares will decrease from 10 assets/share to 9 assets/share
  - Alice's 1,000 shares will be worth 9,000 assets
  - Bob's 1,000 shares will be worth 9,000 assets
- However, if Alice withdraws right before postLoss() is called
  - Alice's 1,000 shares will be redeemed for 10,000 assets
  - Bob's 1,000 shares will now be worth 8,000 assets, bearing all the loss

Recommendations:

Implement a queued withdrawal process, requiring users to:

- First, queue a withdrawal request
- Then, wait a predetermined period (e.g. 1 day)
- Finally, process the withdrawal at the current exchange rate (not the one when the withdrawal was requested)

Additionally, using a private RPC to submit postLoss() transactions reduces the visibility of such transactions, helping to mitigate this issue.

Customer's response: Acknowledged. This was discussed by the team, and the single step withdrawal is intentional. Losses are expected to be extremely rare on yield streaming vaults and the team already uses/recommends private RPCs for this type of transaction.

Informational Issues

I-01. Inconsistent style on postLoss()

Description:

accountantState is first used directly, and only cached afterwards:

```javascript
216 function postLoss(uint256 lossAmount) external requiresAuth {
217 @1> if (accountantState.isPaused) revert AccountantWithRateProviders__Paused();
218
219 @1> if (block.timestamp < lastStrategistUpdateTimestamp +
accountantState.minimumUpdateDelayInSeconds) {
220         revert AccountantWithYieldStreaming__NotEnoughTimePassed();
221     }
...
248             if (lossBps > maxDeviationLoss) {
249 @1>             accountantState.isPaused = true;
250                 emit Paused();
251             }
...
256 @2> AccountantState storage state = accountantState;
257 @3> state.exchangeRate = uint96(vestingState.lastSharePrice);
258
259     //update state timestamp
260     lastStrategistUpdateTimestamp = uint64(block.timestamp);
261
262     emit LossRecorded(lossAmount);
263 }
```

postLoss() accesses the state variable accountantState directly on the 3 instances marked with @1> above. Then, on line 256, it is cached, and then only used once on line 257.

Recommendation:

The style would be more consistent by either:

- Caching the variable by the beginning of the function and accessing the cached variable every time (on every @1> instance)

- Or not caching the variable at all, removing line 256 and accessing the variable directly on line 257

Customer's response:
Acknowledged. Won't fix it for now, but will save this issue for the next code change.

I-02. SupplyObservation.cumulativeSupply might eventually overflow

Description:
SupplyObservation.cumulativeSupply is a uint256 state variable that tracks the cumulative supply, by incrementing itself by vault.totalSupply() * timeElapsed whenever the exchange rate is updated.

Considering type(uint256).max = 1.15e77:

- An average supply of 3.64e69 would overflow in 1 year
- An average supply of 3.64e67 would overflow in 100 years

Considering these are unreasonable values for most legitimate tokens, and that the listing of tokens is controlled by the protocol, this issue is not likely to trigger.

Recommendation:
Wrap cumulativeSupply in an unchecked block, using the overflow as a feature (and calculating the differences accordingly).

Customer's response: Acknowledged. There are no vaults for which this is a realistic average supply.

I-03. A large deposit prior to posting a loss can result in losses larger than 1% of the current supply which don't pause the protocol

Description:

The current design of the `maxDeviationLoss` check mechanism in `postLoss()` isn't based on the cumulative supply, but rather the current one.

Considering that this would allow a 1% deviation (according to default settings), a situation could occur in which a large deposit prior to `postLoss()` call takes place, and the protocol isn't paused due to the large deposit diluting the loss as less than 1% of the total shares.

Recommendation:

Pause the protocol (manually) in all cases prior to posting loss, in order to prevent such scenarios.

Customer's response: Acknowledged. The team will pause the protocol before taking a loss.

I-04. Virtual calculation of fees based on current supply may result in the "over-calculation" of the accumulated fees resulting in less vested yield

Description:

Considering that the fees are calculated virtually based on the current supply rather than the cumulative one, and aren't directly taken from the deposit, but are to be removed from the yield afterwards - this could lead to a few possible problems:

- The first one being that if a deposit is present for a short time within the protocol and wasn't able to generate yield due to not being deployed to underlying strategies and/or other reasons, the calculated fee which included this deposit will be socialized from other users;
- Second, the platform fee always needs to be less than the APY of the vault since it's a cumulative % of the total supply. If it's more than the APY, it will result in the vault operating at a loss.

Recommendation:

Make sure that the platform fee is always less than the APY generated from the strategy.

Customer's response: Acknowledged. The team won't have a platform fee higher than the apy.

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.