macro

Seven Seas A-59

Security Audit

December 30th, 2025
Version 1.0.0

Presented by 0xMacro

Table of Contents

Introduction

This document includes the results of the security audit for Seven Seas's smart contract code as found in the section titled 'Source Code'. The audit was performed by the Macro security team from August 20th to 25th 2025. Additional changes for PR 552 were reviewed from December 1st to 19th 2025.

The purpose of this audit is to review the source code of certain Seven Seas Solidity contracts, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Macro's review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Overall Assessment

The following is an aggregation of issues found by the Macro Audit team:

Severity	Count	Acknowledged	Won't Do	Addressed
High	2	-	-	2
Medium	1	-	-	1
Low	4	2	-	2
Code Quality	2	-	-	2

Seven Seas was quick to respond to these issues.

Specification

Our understanding of the specification was based on the following sources:

- Discussions with the Seven Seas team.
- Available documentation in the repository.

Source Code

The following source code was reviewed during the audit:

- **Instant Withdraw Accountant (from PR 419)**

  Commit Hash: `98b572caf553b34891203c1aaf5dbe69d004276c`

Source Code	SHA256
src/base/Roles/AccountantWithYieldStreaming.sol	`baed6a244d40dc0cc6f5b402f47676b55dd96cd6a72fe948f48f064a10723c0e`
src/base/Roles/TellerWithYieldStreaming.sol	`04f1ea8b9d6242da7bb890c5aa2356fbcf35f3042ad755da7b404e6dbb52618a`
src/base/Roles/TellerWithBuffer.sol	`bfaf1092ad162ee4ef7271b5a3486eb4e3fc769fbf78a7aa1ba48b7ad4857eec`

- **Additional Changes for Instant Withdraw Accountant (from PR 552)**

  Commit Hash: `b779bd8af3b87ed11511c8c428bcfa77031a205d`

Source Code	SHA256
src/base/Roles/AccountantWithYieldStreaming.sol	`8dfbe352a7757b0fc6f7fa4f3e47b24a4ccb4df186dbbdd5baa4e83570f6043f`
src/base/Roles/TellerWithYieldStreaming.sol	`5ca922808be1f6d0ef1d788cbd3847fba1f5b6d36ce0c34588d7022a04cecb6f`
src/base/Roles/TellerWithBuffer.sol	`9c08473dbff552d76500c930614c0a12c74de2f6cd4b30af2b9709c09b74bf25`

Note: This document contains an audit solely of the Solidity contracts listed above. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Issue Descriptions and Recommendations

Click on an issue to jump to it, or scroll down to see them all.

Security Level Reference

We quantify issues in three parts:

1. The high/medium/low/spec-breaking **impact** of the issue:
    - How bad things can get (for a vulnerability)
    - The significance of an improvement (for a code quality issue)
    - The amount of gas saved (for a gas optimization)
2. The high/medium/low **likelihood** of the issue:
    - How likely is the issue to occur (for a vulnerability)
3. The overall critical/high/medium/low **severity** of the issue.

This third part – the severity level – is a summary of how much consideration the client should give to fixing the issue. We assign severity according to the table of guidelines below:

Severity	Description
(C-x) Critical	We recommend the client **must** fix the issue, no matter what, because not fixing would mean **significant funds/assets WILL be lost.**
(H-x) High	We recommend the client **must** address the issue, no matter what, because not fixing would be very bad, *or* some funds/assets will be lost, *or* the code's behavior is against the provided spec.
(M-x) Medium	We recommend the client to **seriously consider** fixing the issue, as the implications of not fixing the issue are severe enough to impact the project significantly, albiet not in an existential manner.
(L-x) Low	The risk is small, unlikely, or may not relevant to the project in a meaningful way. Whether or not the project wants to develop a fix is up to the goals and needs of the project.
(Q-x) Code Quality	The issue identified does not pose any obvious risk, but fixing could improve overall code quality, on-chain composability, developer ergonomics, or even certain aspects of protocol design.
(I-x) Informational	Warnings and things to keep in mind when operating the protocol. No immediate action required.
(G-x) Gas Optimizations	The presented optimization suggestion would save an amount of gas significant enough, in our opinion, to be worth the development cost of implementing it.

Issue Details

H-1 Yield updates and recording losses blocked during vesting period

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Fixed	Medium	High

In AccountantWithYieldStreaming, both `vestYield` and `postLoss` contain the same problematic time-gating check:

```
if (block.timestamp < accountantState.lastUpdateTimestamp + accountantState.mi
```

The issue stems from `accountantState.lastUpdateTimestamp` being updated in multiple contexts:

1. In `_collectFees()` , called during `updateExchangeRate()` , which is triggered during user deposits and withdrawals.
2. In `vestYield()` and `postLoss()` themselves

This creates a problem: if withdrawals and deposits occur frequently, strategists cannot post timely yield updates or loss adjustments. This leads to inaccurate share prices and prevents strategists from promptly recording losses, which masks declining share values from users.

Consider using a separate variable to track when losses or yields were last provided, allowing strategists to perform updates during active investing periods.

~~H-2~~ `bulkWithdraw` fails to update exchange rate and cumulative supply

TOPIC	STATUS	IMPACT	LIKELIHOOD
Incentive Design	Fixed	High	Medium

The `bulkWithdraw` function in TellerWithYieldStreaming isn't overridden, so it uses the standard implementation from TellerWithMultiAssetSupport. This implementation fails to call `updateExchangeRate` before withdrawals and `updateCumulative` after withdrawals.

Updating the exchange rate before deposits/withdrawals is crucial for maintaining accurate `getRate()` and `totalAssets()` values. Similarly, updating `cumulativeSupply` after deposits/withdrawals ensures correct TWAP calculations.

Consider overriding the `bulkWithdraw` function to include `updateExchangeRate` before and `updateCumulative` after `_withdraw` .

~~M-1~~ Yield deviation ignores vesting duration

TOPIC	STATUS	IMPACT	LIKELIHOOD
Incentive Design	Fixed	Medium	Medium

In `vestYield` , the calculation of `yieldBps` considers `yieldAmount` as an absolute value without accounting for the distribution `duration` . This approach fails to recognize that distributing the same `yieldAmount` over e.g. 1 day versus 7 days creates significantly different impacts.

Consider modifying the calculation to normalize `yieldAmount` to a daily rate, then using this normalized value to determine deviation.

~~L-1~~ `previewUpdateExchangeRate` function uses incorrect implementation from parent contract

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Fixed	Low	Medium

The `AccountantWithYieldStreaming` contract fails to override the `previewUpdateExchangeRate` function from its parent `AccountantWithRateProviders` contract. This causes the function to use the parent's exchange rate update logic instead of the yield streaming logic.

Consider disabling the function (if not needed) or implement a proper logic.

~~L-2~~ Price updates are still possible while contract is paused

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Fixed	Low	Low

In `AccountantWithYieldStreaming` , the `paused` state does not prevent price update functions from executing. Specifically, `isPaused` is not checked in `vestYield` , `postLoss` , and `updateExchangeRate` .

This behavior differs from `AccountantWithRateProviders` , where `updateExchangeRate` includes an `isPaused` check. As a result, even when the contract is paused, share price updates can still occur, which may be inconsistent with expected pause semantics.

L-3 Incorrect value provided for `YieldRecorded` event

TOPIC	STATUS	IMPACT	LIKELIHOOD
Events	Acknowledged	Low	Low

At the end of the `vestYield` function, the following event is emitted:

```
emit YieldRecorded(yieldAmount, vestingState.endVestingTime);
```

with the following event declaration:

```
event YieldRecorded(uint256 amountAdded, uint256 newtotalAssetsInBase);
```

The 2nd parameter should be `newtotalAssetsInBase`, but `vestingState.endVestingTime` is provided instead. Either update the event definition or provide the correct parameter when emitting the event.

L-4 Updating cumulative supply fails to emit an event

TOPIC	STATUS	IMPACT	LIKELIHOOD
Events	Acknowledged	Low	Low

The `updateCumulative()` function modifies important state variables like `cumulativeSupply` and `lastUpdateTimestamp` without emitting an event. Best practices recommend emitting events for all functions that change state to improve transparency and facilitate off-chain monitoring.

~~Q-1~~ Unnecessary state variable update in `_collectFees`

TOPIC	STATUS	QUALITY IMPACT
Unnecessary Code	Fixed	Low

In the `_collectFees` function, `state.totalSharesLastUpdate` is set to the current `totalSupply()` on each call. This is redundant because the same value is already set at the end of the `_updateExchangeRate()` function. Consider removing the redundant update to `totalSharesLastUpdate` in the `_collectFees` function.

~~Q-2~~ Nitpicks

TOPIC	STATUS	QUALITY IMPACT
Best Practices	Fixed	Low

- Unused import of BoringVault here.
- Unused import of IPausable here.
- Unused error definition `AccountantWithYieldStreaming__MaxDeviationLossExceeded` here.

Disclaimer

Macro makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Macro specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Macro will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand by any other party. In no event will Macro be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Macro has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the Seven Seas team and only the source code Macro notes as being within the scope of Macro's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Macro. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Macro is not responsible for the content or operation of such websites, and that Macro shall have no liability to your or any other person or entity for the use of third party websites. Macro assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.