Security Audit Report

Turtle turtle-vault
Ethereum smart contract

by ARDA

on November 3, 2026

Table of Contents

Disclaimer **3**

Terminology **4**

Objective **4**

Summary **5**

Inherent Risks **6**

Code Issues & Recommendations **9**
C1: A compromised valuation manager can spam "setPricePerShare" and dilute all holders 9
C2: A corrupted valuation manager can temporarily brick the vault by making the upper geometric bound overflow 11
C3: Anyone can make a price update revert by advancing "priceContextNonce" 13
C4: Users have no protection against fee and PPS increases in "deposit", "mint" and "requestRedeem" 15
C5: Rounding of fees does not always favor holders 17
C6: "settleRedeems" bypasses the valuation manager by setting PPS to 1e18 when supply reaches zero 18
C7: "previewDeposit" and "previewMint" don’t take the fee exemption into account 19
C8: No event emitted when "priceNonce" and "priceContextNonce" change 20

Test Issues & Recommendations **21**
T1: Missing tests on "claimRedeem" reentrancy protection and on owner-only restrictions 21

Disclaimer

The report makes no statements or warranties, either expressed or implied, regarding the security of the code, the information herein or its usage. It also cannot be considered as a sufficient assessment regarding the utility, safety and bugfree status of the code, or any other statements.

This report does not constitute legal or investment advice. It is for informational purposes only and is provided on an "as-is" basis. You acknowledge that any use of this report and the information contained herein is at your own risk. The authors of this report shall not be liable to you or any third parties for any acts or omissions undertaken by you or any third parties based on the information contained herein.

Terminology

Code: The code with which users interact.

Inherent risk: A risk for users that comes from a behavior inherent to the code's design.

Inherent risks only represent the risks inherent to the code's design, which are a subset of all the possible risks. **No inherent risk doesn't mean no risk.** It only means that no risk inherent to the code's design has been identified. Other kind of risks could still be present. For example, the issues not fixed incur risks for the users, or the upgradability of the code might also incur risks for the users.

Issue: A behavior unexpected by the users or by the project, or a practice that increases the chances of unexpected behaviors to appear.

Critical issue: An issue intolerable for the users or the project, that must be addressed.

Major issue: An issue undesirable for the users or the project, that we strongly recommend to address.

Medium issue: An issue uncomfortable for the users or the project, that we recommend to address.

Minor issue: An issue imperceptible for the users or the project, that we advise to address for the overall project security.

Objective

Our objective is to share everything we have found that would help assessing and improving the safety of the code:

1. The **inherent risks** of the code, labelled R1, R2, etc.
2. The **issues** in the **code**, labelled C1, C2, etc.
3. The **issues** in the **testing** of the code, labelled T1, T2, etc.
4. The **issues** in the **other** parts related to the code, labelled O1, O2, etc.
5. The **recommendations** to address each issue.

Summary

Blockchain: Ethereum

Initial scope

- **Repository:** https://github.com/turtle-dao/turtle-vaults
- **Commit:** `1cbcfb7f36bfb43211e1769d7c300a130d63224d`
- **Path(s):**
  `./src/TurtleVault.sol`
  `./src/interfaces/`
  `./src/libraries/`
  `./src/storage/`

Final scope

- **Repository:** https://github.com/turtle-dao/turtle-vaults
- **Commit:** `9935295aeafca5026448e4e3a4c376387cea6346`
- **Path(s):**
  `./src/TurtleVault.sol`
  `./src/interfaces/`
  `./src/libraries/`
  `./src/storage/`

5 inherent risks in the final scope

1 issue in the final scope

9 issues reported in the initial scope and 1 remaining in the final scope.

Severity	Reported			Remaining		
	Code	Test	Other	Code	Test	Other
Critical	0	0	0	0	0	0
Major	1	0	0	0	0	0
Medium	4	0	0	0	0	0
Minor	3	1	0	1	0	0

Inherent Risks

R1: Users have no guarantee on their withdrawal: on whether they will be able to withdraw, on the amount, and on the duration before they can withdraw.

This is because the three parts of a withdrawal each depend on an actor the contract does not constrain.

Ability to withdraw: full trust in the custodian. If the custodian is down or compromised, or simply unable to provide the funds, users cannot withdraw.

Amount: full trust in the price pushed by the valuation manager. If the valuation manager is compromised or stale, users could receive less than expected.

Timing:

1) The custodian can decide to settle an epoch whenever it wants. There is no guarantee at the contract level.

2) There is no guarantee that the custodian processes epochs in order, so a user who requested a redemption before another user could be settled after them.

3) Not all users incur the same redemption delay, because this duration can change, and because all users of the same epoch have the same delay no matter when they entered the epoch, at its start or at its end.

In addition, users can decide to always request a redemption once they deposit, which shortens their settlement delay compared to that of normal users, reducing it by 2x on average. Note, however, that users adopting this strategy would pay the entry and exit fees (which might be 0), and would also be unable to resell or transfer their shares.

R2: Users have no guarantee that the price per share (PPS) used at deposit and redemption is correct.

This is because the PPS is set by an external oracle, the valuation manager. If it is compromised or stale, this could lead to wrong amounts, or to no amounts at all. If no price is pushed at all, for example because the valuation manager is down or cannot compute a price, then the contract is stale, and deposits and redemption settlements are blocked until a new price is pushed.

There are, however, some mitigations in place:

- The owner can replace a valuation manager that does not work properly.
- The valuation manager cannot push just any price: there are bounds at each update, and a cooldown between consecutive updates.
- Redeem delay: an artificially low price per share could allow an attacker to mint a large number of shares through a supply operation. However, redeem operations have a delay between the request and the settlement, providing an opportunity to detect and react to the malicious valuation before funds are withdrawn.
- Custodian validation during redeem settlement: at settlement time, the custodian must provide the price nonce at which the redemption is settled. This provides an additional validation step and prevents the system from blindly accepting the valuation manager's price.

R3: Users can pay a performance fee on a gain they never realize, and it is never refunded if the price later falls.

This is because the performance fee is computed on the vault's global price increase, not per user. It is charged at every new price all-time high, on all shares outstanding at that moment, and is never refunded if the price later falls. That is, the high-water mark (HWM) is global, not per investor, so it does not track the user’s entry price and exit price, hence the following consequences for a user:

- A user can pay a performance fee on a gain never realized if the price goes down before withdrawal: a user entering at 1.0, holding through a peak at 1.5 (fee charged, diluting them) and exiting at 0.9 has paid a performance fee while losing money.

- The user's entry and exit fees are not taken into account, so they might incur the performance fee even if already below break-even.

Note, however, that this same mechanism also cuts the other way, against Turtle rather than users: a user entering during a drawdown may participate in a fee-free recovery up to the existing HWM. For example, if the price reaches an all-time high of 1.5, then the HWM is set to 1.5, and if later a user deposits while the price has gone back to 1.0, then that user will not be charged any performance fee until the price reaches 1.5 again.

R4: Users keep paying management and performance fees on the shares they have asked to redeem, until the redemption is settled, which can happen at an arbitrary later time.

This is because the shares are only burned when the redemption is settled, so until then they stay part of the supply that the management and performance fees are charged on. The user can neither cancel the request nor sell the shares while they wait, and the settlement happens whenever the custodian performs it, with no deadline.

R5: Users have no way to anticipate fee increases: entry fee, exit fee, performance fee, management fee.

This is because the entry fee, the exit fee, and the management fee are all set by the owner, who can change any of them at any time, with no timelock. Meanwhile, a change of performance fee is not immediate, rather it becomes effective at the next price publication, and users have no control when this will occur.

Users therefore have no guarantee that they will be able to anticipate a fee change before it applies to them. Moreover, even if they are informed about upcoming fee changes, they might not be able to withdraw their funds before the changes, since withdrawals incur delays and are gated by the custodian.

Code Issues & Recommendations

C1: A compromised valuation manager can spam "setPricePerShare" and dilute all holders

Severity: Major **Status:** Fixed

Location

```
src/TurtleVault.sol
   setPricePerShare
```

Description

Current behavior: Under the `GeometricBounds` validation model, nothing limits how often the valuation manager publishes the price per share (PPS) with `setPricePerShare`.

The validation compares the new PPS only with the currently stored one (link), which every publication replaces, so ten publications inside one block move the price by ten full steps.

Therefore, a corrupted valuation manager can move the PPS to any value by performing sufficiently many updates.

Expected behavior: The valuation manager is not an entity that has the same trust level as the owner or the custodian, and therefore the contract should be protected if it gets compromised.

For this, two consecutive publications by the valuation manager should be separated by a minimum delay, so that the price cannot move by more than one validated step at a time, leaving sufficient time for the owner to rotate a compromised valuation manager.

Worst consequence: A single compromised valuation-manager key shrinks the PPS to a very low value and then mints a huge amount of shares, making all other holders' shares worth almost nothing. This can occur within one block and with no way back.

Example: Alice holds 10,000 shares at a price of 1 USDC. In one block the corrupted valuation manager Bob reduces the price by 100x, then deposits 900

USDC at the resulting price, and receives 90,000 shares. In this way, Bob has effectively stolen 9,000 USDC worth of shares from Alice.

Recommendation

We recommend enforcing a cooldown, a minimum delay between two consecutive PPS publications in `setPricePerShare`. For this, we can introduce a new owner-set cooldown storage `priceUpdateCooldown`, enforced in `setPricePerShare`. Here are the constraints on the cooldown:

- `pricePerShareDuration >= priceUpdateCooldown`. This is to ensure that the vault can have a fresh price at all times.
- The cooldown should be neither too short (to mitigate the threat of a compromised valuation manager), nor too long (to ensure that the price can be updated fast enough in a high-volatility market), typically between 2 and 6 hours. Moreover, we can also add an entry point restricted to the owner or the custodian that publishes a PPS without waiting for the cooldown.

C2: A corrupted valuation manager can temporarily brick the vault by making the upper geometric bound overflow

Severity: Medium **Status:** Fixed

Location

`src/TurtleVault.sol`

Description

Current behavior: `_validateGeometricBoundsModel` computes both the lower and the upper allowed price per share (PPS) before comparing the published price against them (link). For any non-zero upper tolerance `positiveToleranceBps`, the upper bound (link) stops fitting in 256 bits once the PPS gets close enough to `2^256 - 1`, and thus reverts with an arithmetic panic before any direction check runs.

In particular, a publication that lowers the price also reverts, although it only needs the lower bound.

Expected behavior: Only the bound on the side the price is moving should be computed, so that a downward publication within the negative tolerance is accepted whatever the stored price per share is.

Worst consequence: A corrupted valuation manager pushes the price very close to `2^256 - 1` such that even a tolerance `positiveToleranceBps` of 1 BPS would make the upper geometric bound overflow. Note that in the current model, the valuation manager can reach this state quickly by spamming `setPricePerShare` (see C1: A compromised valuation manager can spam "setPricePerShare" and dilute all holders).

From there, subsequent price publications revert and the vault is unusable for deposits or withdrawals. When the owner is informed of the issue, he can rotate the valuation manager, but he then has no satisfactory way to fix the PPS:

- In the geometric model, a zero `positiveToleranceBps` prevents him from changing the PPS, but any non-zero `positiveToleranceBps` makes `setPricePerShare` revert.
- If the owner switches to the annualized return model, he can set a zero upper tolerance and a non-zero lower tolerance, in order to reduce the

price. Unfortunately, on most chains, `currentPricePerShare * elapsed` would overflow, since PPS updates require `elapsed > 0`, which usually implies that `elapsed >= 2` since block time is ~12s on Ethereum and ~2s on Base and Optimism. But then `currentPricePerShare * elapsed >= currentPricePerShare * 2 > 2^256` overflows.

- The owner can just wait for management fees to accrue and sufficiently reduce the PPS until updates can be made, but this could take a very long time if fees are low (possibly zero). And if the owner decides to increase the management fees to speed up the recovery, this would make users lose funds.

Note: For some assets, it might be impossible to publish a price close to `2^256 - 1` if it would make `_requireRepresentableTotalAssets` fail due to an overflow of `shares * pps / WAD`. However, for high-value tokens with small decimals (typically BTC or USDC), the supply amount `shares` would likely be significantly smaller than `WAD`, hence `shares * pps / WAD` would fit in a `uint256`.

Recommendation

We recommend computing, in `_validateGeometricBoundsModel`, only the bound on the side the price is moving, both for the soft and hard bound checks. The bound of the unused direction must never be recomputed on the executed path, including in events and errors.

C3: Anyone can make a price update revert by advancing "priceContextNonce"

Severity: Medium **Status:** Fixed

Location

`src/TurtleVault.sol`

Description

The entry point `setPricePerShare` accepts a new price only if the `expectedPriceContextNonce` the valuation manager pinned when it built its transaction is exactly equal to the vault's live `priceContextNonce` (link).

However, `priceContextNonce` is advanced by calls anyone can make:

- Every non-zero mint or burn increments `priceContextNonce` in `_update` (link).
- `accrueManagementFee` increments `priceContextNonce` when the accrual is non-zero (link), which can happen at most once per block.

Consequently, any account can move the nonce between the valuation manager's off-chain read and its transaction's inclusion, and the publication then reverts with `UnexpectedPriceContextNonce`.

If this prevents the valuation manager from pushing prices over an extended period of time, it would eventually make the stale price invalid, and then users would be unable to deposit and withdraw.

Recommendation

We recommend keeping the contract code unchanged, and rather:

- Making `minimumDeposit` big enough on each vault, to sufficiently deter deposit spamming.
- Implementing the following capability off-chain in the valuation manager, to mitigate spamming of `accrueManagementFee`: preparing 2 transactions on consecutive `priceContextNonce` values for the same block, ensuring that exactly one of them will land since accruals can bump the nonce by at most 1 within a single block.

- Implementing the following capability off-chain in the valuation manager, to mitigate spamming of `deposit` or `mint`: preparing an `expirePricePerShare` transaction followed by a `setPricePerShare` transaction, ensuring that no deposits can be made in between.
- Monitoring repeated `UnexpectedPriceContextNonce` reverts on publication, as well as nonce advances not attributable to known operations, which can also benefit from the recommendation of C8: No event emitted when "priceNonce" and "priceContextNonce" change.

Alternatively, if the team prefers a mitigation at the smart contract level, we suggest reaching out to the auditor.

C4: Users have no protection against fee and PPS increases in "deposit", "mint" and "requestRedeem"

Severity: Medium **Status:** Fixed

Location

`src/libraries/FeeMath.sol`

Description

Current behavior: `deposit`, `mint` and `requestRedeem` take no argument bounding the terms of the caller's own transaction (link), while the stored value deciding those terms can be changed by a privileged role with immediate effect. In such cases, the user could receive less than they would agree to:

- If the entry fee is increased just before a deposit or mint;
- If the exit fee is increased just before a redeem request;
- If the fee recipient is changed just before a deposit, mint or redeem request, and the caller is the old fee recipient;
- If the price per share (PPS) is changed just before a deposit or mint.

Consequently, a user's transaction can execute on terms materially worse than the ones they previewed, without reverting.

Expected behavior: `deposit` and `mint` should let the caller state a maximum effective entry fee rate and a maximum PPS, and `requestRedeem` a maximum exit fee (or a minimum net share amount), the call reverting when the values read at execution time fall outside those bounds. In particular, `deposit` and `mint` should take two protecting parameters because:

- With only a protection on the maximum PPS, the asset value of the user's shares could be smaller than expected due to a fee increase.
- With only a protection on the maximum entry fee, the user is not protected against a PPS change that they do not agree with, e.g. a PPS that does not accurately reflect the AUM.

Recommendation

We suggest overloading the three entry points `deposit`, `mint` and `requestRedeem` with versions that take protection arguments:

- `deposit` and `mint` would take a maximum entry fee and PPS,
- `requestRedeem` would take a maximum exit fee.

C5: Rounding of fees does not always favor holders

Severity: Medium **Status:** Fixed

Location

`src/libraries/FeeMath.sol`

Description

Current behavior: The four fees (entry, exit, performance, management) do not round in the same direction:

- The entry fee and the exit fee are both **rounded up**, because `splitGrossShares` rounds the user's net shares down and gives the whole remainder to the fee (link).
- The performance fee has no fixed direction at all: which way it rounds depends on the values of the moment (link). The amounts are small — at most one share unit per operation; the problem is the absence of a provable direction in the fee arithmetic.
- The management fee is rounded down (link).

Expected behavior: All fees should round in favor of the maximum number of holders, as the management fee already does.

Recommendation

We suggest always rounding fees in favor of holders:

- rounding `netShares` up in `splitGrossShares`, which fixes both the entry and the exit fee since both go through it;
- computing the `grossAssets` term used to build `assetsAfterFee` rounding up (link).
- in `mint`, we can round down `grossShares` so that fees are underestimated, and round `assets` up such that the shares are correctly backed.

C6: "settleRedeems" bypasses the valuation manager by setting PPS to 1e18 when supply reaches zero

Severity: Minor **Status:** Fixed

Location

```
src/TurtleVault.sol
    settleRedeems
```

Description

`settleRedeems` rewrites the stored price per share to `1e18` and resets the performance fee high-water mark (HWM) whenever the settlement burn brings the total supply to zero. However, price updates are intended to be performed by the valuation manager in full awareness, and not automatically by the contract. This hard reset could pose a risk, in future code changes, if ever reachable from an unexpected flow. Finally, it adds complexity to the sensitive price update logic.

Recommendation

We suggest deleting the automatic zero-supply and HWM reset (link). If the supply ever reaches 0:

- If the valuation manager pushes a price while the supply is 0, it will necessarily be `1e18` and the HWM will be reset.
- If a user deposits before a new PPS is pushed, then the old PPS will be used, and it is as valid as `1e18`.

C7: "previewDeposit" and "previewMint" don’t take the fee exemption into account

Severity: Minor **Status:** Won't fix

Location

`src/TurtleVault.sol`

Description

The functions `previewDeposit` and `previewMint` quote the entry fee with the stored rate `fees.entryFeeBps` (link), while `deposit` and `mint` charge `_entryFeeBps(fees, receiver)`, which is 0 when the receiver is the fee recipient (link). Whenever the stored entry fee is non-zero, the quote therefore does not match the execution for that one address.

The divergence is intentional and in the safe direction: `previewDeposit` and `previewMint` can’t take a user as an argument, as they must stay compliant with the ERC-4626 standard; therefore, they can’t take the exemption into account. What is missing is a receiver-aware variant of these quote functions, which the exit side already provides through `previewRequestRedeem` (link).

Recommendation

We suggest adding quote functions `previewDepositFor(uint256 assets, address receiver)` and `previewMintFor(uint256 receiverShares, address receiver)`, each identical to the existing view with `fees.entryFeeBps` replaced by `_entryFeeBps(fees, receiver)`. In particular, we would leave `previewDeposit` and `previewMint` untouched, to preserve the ERC-4626 compatibility.

Resolution notes

Since the only user for which these queries would return an incorrect value is the administrative account `fee_receiver`, the team preferred not adding functions to handle this specific case.

C8: No event emitted when "priceNonce" and "priceContextNonce" change

Severity: Minor **Status:** Fixed

Location

`src/TurtleVault.sol`

Description

The vault keeps two counters, `priceNonce` and `priceContextNonce`, that off-chain operators must send back for `setPricePerShare` (link) and `settleRedeems` (link) to succeed.

However, no event carries either value: the four paths advancing `priceContextNonce` all run through the private helper `_incrementPriceContextNonce`, which emits nothing (link).

This makes it harder for the valuation manager and custodian to monitor these nonces.

Recommendation

We suggest introducing new events emitted whenever an increment of either `priceNonce` or `priceContextNonce` occurs.

Test Issues & Recommendations

T1: Missing tests on "claimRedeem" reentrancy protection and on owner-only restrictions

Severity: Minor **Status:** Fixed

Description

The test suite does not pin two protections of the code, so a future change that removes either would be uncaught by tests and would have a higher risk of reaching production:

- Reentrancy protection: `claimRedeem` resists a reentrant double claim only through the order of its statements — it zeroes the user's redeem request and decreases the epoch's balances before transferring the assets out (link). However, there is no reentrancy test witnessing that this mechanism works as intended.
- Owner-only entry points: `setValuationManager` (link), `setRedeemConfig` (link), `activate` (link) and the factory's `_authorizeUpgrade` (link) are all `onlyOwner`, but none is covered by a test checking that another caller is rejected with `OwnableUnauthorizedAccount`.

Recommendation

We recommend adding the missing tests.