Hydrogen Labs: Liquid Staking

Security Review

Cantina Managed review by:
Optimum, Lead Security Researcher
Arno, Associate Security Researcher

February 27, 2026

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**
- 2.1 Scope . . . 3

3 Findings **4**
- 3.1 Low Risk . . . 4
  - 3.1.1 Incorrect Return Values in `maxDeposit()` and `maxWithdraw()` break EIP-4626 Compliance . . . 4
  - 3.1.2 Inaccurate Loop Validation in `adminInitiateRebalanceBpsValId()` Allows Skipping Account Processing . . . 4
  - 3.1.3 Incorrect Rounding Direction in `undelegate` Burns Fewer Shares Than Expected . . . 5
  - 3.1.4 `gVault`: `pauseValId()` and `unpauseValId()` should only be allowed to be called for active validators . . . 5
  - 3.1.5 Unconditional rewards-claim timestamp update bypasses overdue guard . . . 5
- 3.2 Informational . . . 6
  - 3.2.1 Code Quality & Maintainability . . . 6
  - 3.2.2 `BaseVault._claimValidatorRewards()` Missing `RewardsClaimed` Event Emission . . . 6
  - 3.2.3 `BaseVault`: Consider refactoring how validator state machine is stored in storage . . . 7
  - 3.2.4 `gVault.adminCompleteRebalance()` Lacks an explicit ”paused” check . . . 7
  - 3.2.5 Asymmetric Gas Griefing via Unbounded Rebalance Loop . . . 7

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

From Feb 10th to Feb 14th the Cantina team conducted a review of contracts-public on commit hash b342d63c. The team identified a total of **10** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	5	3	2
Gas Optimizations	0	0	0
Informational	5	3	2
Total	**10**	**6**	**4**

2.1 Scope

The security review had the following components in scope for contracts-public on commit hash b342d63c:

```
src
├── BaseVault.sol
├── CoreVault.sol
├── gVault.sol
├── MagmaErrorsModule.sol
├── MagmaV2.sol
└── PausableValId.sol
```

3 Findings

3.1 Low Risk

3.1.1 Incorrect Return Values in `maxDeposit()` and `maxWithdraw()` break EIP-4626 Compliance

Severity: Low Risk

Context: MagmaV2.sol#L220

Description: The current implementation of several EIP-4626 view functions in `gVault` does not fully adhere to the EIP-4626 standard. The standard requires these functions to provide a highly accurate, real-time reflection of the vault's underlying assets and the maximum possible user interactions.

Specifically, the following discrepancies were identified:

- Underestimation of `totalAssets()`: According to the EIP-4626 specification, `totalAssets()` should reflect the total amount of managed underlying assets. This includes **unclaimed staking rewards** and should account for **reward fees** (though it should excluding withdrawal fees). Excluding these from the calculation leads to an undervalued share price and inaccurate reporting of the vault's TVL.
- Inaccurate `max` Functions: `maxDeposit()`, `maxMint()`, `maxWithdraw()`, and `maxRedeem()` fail to account for:
  - Pause States: They do not return `0` when the vault is paused (via `MagmaV2` or internal `gVault` mechanisms).
  - Global Caps: The return values do not reflect deposit/mint limits implemented in the `gVault` logic.
  - Validator Status: The functions do not return `0` if a validator is paused or removed, which halts operations.

Recommendation: Update the EIP-4626 view functions to incorporate all internal logic and pending rewards:

1. `totalAssets()`: Modify the logic to include all accrued but unclaimed staking rewards and ensure reward-related fee logic is applied as per the specification.
2. `max` functions: Update these to return `0` instead of reverting during any condition that would cause a transaction failure (e.g., pauses, reaching the global cap, or inactive validator status).

Hydrogen Labs: Acknowledged.

Cantina Managed: Acknowledged.

3.1.2 Inaccurate Loop Validation in `adminInitiateRebalanceBpsValId()` Allows Skipping Account Processing

Severity: Low Risk

Context: gVault.sol#L503

Description: The function `adminInitiateRebalanceBpsValId` is designed to update user share balances in batches during a validator rebalance. The current validation logic for the batch range is incorrectly implemented:

```
if (_start == _stop) revert ErrInvalidStop();
```

This logic is flawed for two reasons:

1. Prevents Single-Account Batches: It reverts if an admin attempts to process a batch containing only one account `(_start == _stop)`, which is a valid operational case.
2. Allows Skipping Logic: It fails to check if _start > _stop. Since the loop condition is i <= _stop, a start index greater than the stop index will cause the loop to skip entirely while still allowing the tracking index to advance via `$._lastRebalancedStartIndex[_valId] = _stop + 1;`.

Recommendation: Update the validation to permit single-account batches and strictly forbid inverted ranges:

```
// Replace:
if (_start == _stop) revert ErrInvalidStop();

// With:
if (_start > _stop) revert ErrInvalidStop();
```

Hydrogen Labs: Fixed in commit 5793bdb9.

Cantina Managed: Fix verified.

3.1.3 Incorrect Rounding Direction in `undelegate` Burns Fewer Shares Than Expected

Severity: Low Risk

Context: gVault.sol#L424

Description: In `gVault.sol`, the share amount to burn for an asset-specified withdrawal is rounded down,Per ERC4626, for `withdraw`/`previewWithdraw`-style flows (user specifies **assets**), the required **shares** should be rounded **up** so the user burns the maximum shares for the requested assets (favoring the vault). Using `Floor` can under-burn shares, leaking value to withdrawers at the expense of remaining share holders (and can create edge-case "dust"/zero-share burns).

Recommendation: Use `Math.Rounding.Ceil` instead of `Math.Rounding.Floor` when computing shares to burn on withdrawal:

```
uint256 _shares = _sharesForAssets(_valId, _amount, Math.Rounding.Ceil);
```

Hydrogen Labs: Acknowledged.

Cantina Managed: Acknowledged.

3.1.4 `gVault`: `pauseValId()` and `unpauseValId()` should only be allowed to be called for active validators

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: Both `pauseValId()` and `unpauseValId()` can be called for non-active validators (non existent/paused/removed) which might not happen in reality since it is an owner only function but is not accurate in terms of the expected state machine transitions.

Recommendation: Short term: Consider reverting in case the validator is not marked in the `_isWhitelisted` mapping.

Long term: consider fully implementing the recommendation of finding `BaseVault`: Consider refactoring how validator state machine is stored in storage.

Hydrogen Labs: Fixed in commit 5793bdb9.

Cantina Managed: Fix verified.

3.1.5 Unconditional rewards-claim timestamp update bypasses overdue guard

Severity: Low Risk

Context: gVault.sol#L612

Description: In `claimAndCompoundRewards()` the contract updates the global rewards-claim timestamp unconditionally, even if no validator rewards were actually claimed (e.g., all validators are paused or otherwise skipped in the loop).

`delegate()` and `undelegate()` call `_checkRewardsClaimDelay()`, which only reverts if the time since `_lastRewardsClaimTimestamp` exceeds `_maxRewardsClaimDelay` (default 7 days). Because the timestamp can be refreshed without claiming, the "rewards overdue" protection becomes ineffective and delegations/undelegations can continue while rewards accumulate but remain unclaimed.

Recommendation: Only call `_updateLastRewardsClaimTimestamp()` if rewards were actually claimed (e.g., track `claimedAny` / `claimedAllEligible` inside the loop).

Hydrogen Labs: Fixed in commit 5793bdb9.

Cantina Managed: Fix verified.

3.2 Informational

3.2.1 Code Quality & Maintainability

Severity: Informational

Context: gVault.sol#L187

Description:

1. gVault.sol#L187: `_isRebalanceBatchCompleted()` should be used here instead.
2. `_isRebalanceBatchCompleted()`: Will return `true` in case a rebalance never started, the name of the function does not reflect that. You might want to change the name to `isRebalanceInactive()`.
3. `_isRebalanceBatchCompleted()`: the second part of the condition `$._lastRebalancedStartIndex[_valId] != $._accountsByValidator[_valId].length` is redundant and should be removed.
4. Remove code comments about `multiplier` from the `gVault` contract.
5. Typo in gVault.sol#L318 in the word "funcionality".
6. Missing natspec documentation for the `_magmaShares` parameter that was added to many functions in the `gVault` contract.
7. gVault.sol#L335: The comment of "// this cannot be 0, checked at checkMagmaShares modifier," is wrong since this check is missing from `checkMagmaShares`.
8. Consider renaming `adminInitiateRebalanceBpsValId()` to `adminRebalanceInitiateBpsValId()` so it will resemble the `adminRebalanceInitiate()` function of `CoreVault` which will make it easier to navigate in the code, you might want to also change `Core.redelegateToValidators()` to `adminCompleteRebalance` or some shared name for both contracts.

Hydrogen Labs: Fixed in commit 5793bdb9.

Cantina Managed: Fix verified.

3.2.2 `BaseVault._claimValidatorRewards()` Missing `RewardsClaimed` Event Emission

Severity: Informational

Context: BaseVault.sol#L712

Description: The internal function `_claimValidatorRewards()` in `BaseVault` is responsible for processing and collecting rewards from validators. However, the function does not emit the `RewardsClaimed` event upon a successful claim. While the state is updated correctly, the lack of an event makes it difficult for off-chain infrastructure to track these actions.

Recommendation: Add an `emit RewardsClaimed(...)` statement within the `_claimValidatorRewards()` function.

> **Note on Implementation:** Given that this is a non-critical improvement and does not violate any EIP standards, an immediate contract upgrade or migration may be considered "overkill." It is recommended to include this event emission in the next scheduled deployment or major architectural upgrade to improve the long-term observability of the protocol.

Hydrogen Labs: Fixed in commit 5793bdb9.

Cantina Managed: Fix verified.

3.2.3 `BaseVault`: Consider refactoring how validator state machine is stored in storage

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: `BaseVault`, `CoreVault` and `gVault` implement a state machine that represents the life-cycle of validators in Monad. In the current version of the code, the state is stored in multiple different variables in the storage, which makes it harder to maintain and prone to errors like we have seen in finding `gVault`: `pauseValId()` and `unpauseValId()` should only be allowed to be called for active validators.

Originally, `BaseVault` is using the `_validatorStatus` mapping that maps a validator id to the following states:

```
enum ValidatorStatus {
        NONE,
        PAUSED,
        REMOVED,
        UNDELEGATING
    }
```

However, this enum lacks many states that are actually implemented in the code, such as:

1. ACTIVE state (or ADDED): After adding a validator (when `_registerValidator()` is called), `_validatorStatus` is set to `NONE` (0) for that validator which is redundant, instead of setting it to ACTIVE. `_isWhitelisted` is an auxiliary mapping that is used here instead but should be removed in case you are adding the ACTIVE state instead.
2. PAUSED state is ambiguous: `BaseVault` uses `_validatorStatus` to store the `PAUSED` when initiating a removal of a validator, while `gVault` is also using the paused terminology (managed by the `PausableValId` contract) but it is used for rebalancing. You might want to change PAUSED to PAUSED_FOR_REMOVAL and introduce a new state that is originally meant for `gVault` only called `PAUSED_FOR_REBALANCING` but can and should actually be used for `CoreVault` as well inside `adminRebalanceInitiate()`.

Note on Completeness: The states identified above represent the immediate gaps in the current implementation; however, there might be more states needed for the protocol's specific edge cases-please take the time to verify the full lifecycle before finalization.

Hydrogen Labs: Acknowledged.

Cantina Managed: Acknowledged.

3.2.4 `gVault.adminCompleteRebalance()` Lacks an explicit ”paused” check

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: Although we could not find a concrete issue in the current code, we think it is better to add a check that restricts access to the `adminCompleteRebalance()` like the one that's implemented in `adminInitiateRebalanceBpsValId()`.

Recommendation: Short term: Consider implementing the same check of `whenPausedValId`.

Long term: Consider improving the state machine management in storage as described in finding `BaseVault`: Consider refactoring how validator state machine is stored in storage.

Hydrogen Labs: Fixed in commit 5793bdb9.

Cantina Managed: Fix verified.

3.2.5 Asymmetric Gas Griefing via Unbounded Rebalance Loop

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The `gVault` contract employs a ”Push” rebalancing model in `adminInitiateRebalanceBpsValId` that requires the admin to manually iterate through every

account in `_accountsByValidator`. While batching via `_start` and `_stop` parameters allows the admin to stay within block gas limits per transaction, the architectural design creates a permanent, asymmetric vulnerability to gas griefing.

The Asymmetric Cost Risk:

- Low Entry Cost for Attacker: An attacker can ”stuff” the array by depositing the `_minUserDepositAmount` (currently 100 MON) across numerous unique addresses. This is a one-time capital commitment to create a permanent operational burden on the owner.
- High Recurring Cost for Admin: Because the admin must process the array sequentially, every ”dust” account created by the attacker adds mandatory `SLOAD` operations and an expensive `SSTORE` to update `_magmaSharesForUserByValidator`.
- Ineffectiveness of Pruning: A malicious actor will never undelegate. Their addresses remain in the storage array indefinitely, as pruning only triggers on voluntary exits (balance reaching zero).
- Sequential Locking: `adminCompleteRebalance` is strictly locked and will revert unless `_lastRebalancedStartIndex` exactly matches the total array length. The admin is forced to spend gas to update negligible balances just to unlock the validator's capital and forward funds to the `CoreVault`.

Impact: The Frequency Multiplier: The severity of this impact is directly tied to the **frequency of rebalancing**:

- Operational ”Tax”: The cost of a rebalance is incurred **every single time** the admin initiates the process. While a single batch may be affordable, the cumulative cost of processing thousands of ”junk” accounts across multiple rebalances per year creates a significant drain on the protocol's operational budget.
- Cumulative Delay: Beyond the gas cost, the frequency of rebalancing compounds the risk of capital inefficiency. Each rebalance puts funds in a `pendingRedelegateByValidator` state (earning 0% rewards). High-frequency rebalancing under an ”array stuffing” attack means validator funds spend a disproportionate amount of time earning zero yield while waiting for the admin to clear the ”dust” batches.

Recommendation:

1. Gas Benchmarking: The team should conduct gas tests to measure the cost of `adminInitiateRebalanceBpsValId` when processing a validator with 1,000+ addresses. Specifically, compare the gas limit consumed by an iteration that performs an `SSTORE` versus one that only performs an `SLOAD`.
2. Dust Threshold Skip: Modify the loop in `adminInitiateRebalanceBpsValId` to allow the admin to skip addresses where `_magmaSharesForUserByValidator` is below a certain ”dust” threshold (e.g., 0.1 MON). This eliminates the expensive `SSTORE` gas cost for poisoned accounts, significantly reducing the recurring tax and allowing for larger, faster batches.
3. Entry Barrier Review: Re-evaluate if the `_minUserDepositAmount` provides a sufficient economic barrier to deter an attacker from adding thousands of addresses to the state.

Hydrogen Labs: Acknowledged.

Cantina Managed: Acknowledged.