CANTINA

Magma: Monad Liquid Staking Protocol

Security Review

Cantina Managed review by:

Optimum, Lead Security Researcher
Arno, Associate Security Researcher

March 2, 2026

Contents

1 Introduction **3**
1.1 About Cantina . . . 3
1.2 Disclaimer . . . 3
1.3 Risk assessment . . . 3
1.3.1 Severity Classification . . . 3

2 Security Review Summary **4**

3 Findings **5**
3.1 High Risk . . . 5
3.1.1 `finishedLastRebalance` Flag Set to False but Never Reset to True, Potentially Locking Rebalances . . . 5
3.1.2 `gVault`: removing a validator in the middle of a rebalance will cause loss of funds and a denial of service . . . 5
3.1.3 `CoreVault`: Potential "deadlock" between validator addition and removal leading to locked funds . . . 6
3.1.4 Storage state of validators is not deleted properly leading to accounting issues . . . 6
3.2 Medium Risk . . . 7
3.2.1 Precision Loss in gVault User Entitlement Calculation Due to Large Scale Factor . . . 7
3.2.2 Unbounded `gVault` scale growth causes overflow and DoS in admin rebalancing . . . 7
3.2.3 Interim double-counting in `totalAssets` during validator removal initiation . . . 10
3.2.4 Lack of Maximum Validator Count Cap Risks Gas Exhaustion in Loops . . . 12
3.2.5 Early withdrawers from a slashed validator incur significantly smaller losses than others 13
3.2.6 Withdrawn amount after undelegation might be less than expected, potentially causing reverts and unexpected behavior . . . 14
3.2.7 `gVault.undelegate()`: Only the original buyers of validators shares will be able to redeem . . . 14
3.2.8 `gVault.claimAndCompoundRewards()` to revert in case `_remaining = 0` . . . 15
3.3 Low Risk . . . 16
3.3.1 Inconsistent Updates to `lastRebalanceTimestamp` in Rebalancing Functions . . . 16
3.3.2 Delegation Cap Not Enforced on Last Validator During Stake Distribution . . . 16
3.3.3 Missing Enforcement of Minimum Interval in `setDelegatorInfoUpdateInterval` . . . 17
3.3.4 Incorrect Return Value on Failed Rewards Fee Transfer . . . 17
3.3.5 Incorrect Owner and Shares in Withdraw Event Emission . . . 18
3.3.6 Missing cache update on rewards compounding/injection . . . 18
3.3.7 `gVault.delegate()` will revert for validators with a default cap value and `magma().totalAssets()` is 0 . . . 18
3.3.8 `VaultBase`: re-registrating a validator in the middle of its removal leads to unintended consequences . . . 19
3.4 Gas Optimization . . . 19
3.4.1 `gVault.adminInitiateRebalanceBps()` calls `setLastRebalanceTimestamp()` redundantly . . . 19
3.4.2 `_distributeToNextValidator()`: Redundant double array sorting . . . 20
3.4.3 Remove unused imports of `BitMapLib` . . . 20
3.4.4 `gVault._claimAndCompoundRewards()` optimizations . . . 20
3.4.5 Avoid unnecessary state updates when no undelegation occurs . . . 21
3.4.6 `refreshCache()` gas optimizations . . . 21
3.5 Informational . . . 21
3.5.1 Importance of Validator Activity Monitoring . . . 21
3.5.2 Uncompounded rewards excluded from total assets calculation may lead to fairness issues . . . 22
3.5.3 `_claim` Function Reverts on Non-Critical Failures, causing Unnecessary Reverts in Reward Handling . . . 22
3.5.4 `_completeUserWithdrawal()`: violation of the "*No writes after a call*" best practice . . . 23
3.5.5 The `AdminInitiatedRebalance` event is missing parameters . . . 24
3.5.6 Fixes to inline comments . . . 24
3.5.7 `Magma`: User experience improvements . . . 24
3.5.8 Unused Referral Event Definition . . . 25
3.5.9 `_allocateAdminWidAndUndelegate` Returns Default Value Instead of Allocated WID . . . 25

3.5.10 Validator Caps Bypassed in Rewards Compounding and Injection 25
3.5.11 Rounding mismatch: code uses Floor while comment specifies Ceil 25
3.5.12 `Magma`: Use two-step admin transfer process 26
3.5.13 Calls to `delegate()` and `undelegate()` will revert in case rewards were not claimed for a duration of `_maxRewardsClaimDelay` 26
3.5.14 `claimAndCompoundRewards()`: missing reentrancy guard 26
3.5.15 Magma token holders can sybil and bypass the constraint of only one redeem request at a time 27
3.5.16 Redundant check of `_exists` 27
3.5.17 `RebalanceInitiated` is an empty event 27
3.5.18 Redundant complexity in the accounting of validator removal 27
3.5.19 `VaultBase` lacks a `receive()` function 28
3.5.20 `redelegateToValidators()` is missing event emission 28

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

Magma is a Liquid Staking Protocol on Monad building a Distributed Validator and low-latency MEV client.

From Oct 7th to Oct 12th the Cantina team conducted a review of contracts-public on commit hash 92702eca. The team identified a total of **46** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	4	4	0
Medium Risk	8	6	2
Low Risk	8	7	1
Gas Optimizations	6	6	0
Informational	20	12	8
Total	**46**	**35**	**11**

3 Findings

3.1 High Risk

3.1.1 `finishedLastRebalance` Flag Set to False but Never Reset to True, Potentially Locking Rebalances

Severity: High Risk

Context: CoreVault.sol#L122

Description: The `finishedLastRebalance` flag is initialized to `true` in `VaultBase.__VaultBase_init` and set to `false` in `CoreVault.adminRebalanceInitiate`to prevent concurrent rebalances. However, it is never reset to `true` anywhere in `CoreVault.sol` or the base `VaultBase.sol`, even at rebalance completion (e.g., in `_redelegateRedistribute`, line 421, where a comment mentions marking it as finished but no code does so). This leaves the flag stuck at `false` after initiation, causing subsequent calls to `adminRebalanceInitiate` to revert permanently.

No other functions in the contract or base set it to `true` (only set to `false` in `adminRebalanceInitiate` and initialized to `true` once).

```
function adminRebalanceInitiate() external onlyAdmin onlyAfterEpoch {
    _refreshCache();
    if (!finishedLastRebalance()) revert ErrRebalanceInProgress();
    setFinishedLastRebalance(false);
    _redelegateInitiate();
    setLastRebalanceTimestamp(block.timestamp);
}
```

(In `VaultBase.sol: $._finishedLastRebalance = true;` only during initialization).

Recommendation: Add `setFinishedLastRebalance(true);` at the end of `_redelegateRedistribute()` (`CoreVault.sol`), where the comment already indicates marking the rebalance as finished.

Hydrogen Labs: Fixed in commit b0b747de.

Cantina Managed: Fix verified.

3.1.2 `gVault`: removing a validator in the middle of a rebalance will cause loss of funds and a denial of service

Severity: High Risk

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

Description: Similarly to finding "`CoreVault`: Potential "deadlock" between validator addition and removal leading to locked funds", if the admin attempts to remove a validator that is already participating in an ongoing rebalance, the undelegated funds for that validator - which are meant to be redelegated to the `CoreVault` - will remain locked in the Monad staking contract. Additionally, from that point onward, any attempt to complete the validator removal will revert.

Scenario:

1. The admin calls `adminInitiateRebalanceBps()`, and validator A is included in the rebalance operation.
2. Before completing the rebalance, the admin calls `initiateValidatorRemoval(A)` to remove validator A.

At this stage, calling `adminCompleteRebalance()` will only process other validators, leaving A's undelegated funds stuck in the Monad staking contract. Subsequent calls to `executeValidatorUndelegation()` will revert because the admin withdrawal ID is already allocated for validator A.

Recommendation: Ensure that the rebalance and removal processes are mutually exclusive. The proposed fix of finding "`CoreVault`: Potential "deadlock" between validator addition and removal leading to locked funds" should solve this issue as well.

Hydrogen Labs: Fixed in commit bf7a705e.

Cantina Managed: Fixed by implementing the auditor's recommendation.

3.1.3 `CoreVault`: Potential "deadlock" between validator addition and removal leading to locked funds

Severity: High Risk

Context: CoreVault.sol#L429

Description: The `CoreVault` contract allows the admin to add new validators. Unlike `gVault`, which performs this in a single step, `CoreVault` uses a two-step process:

1. `addValidator()` / `addValidators()`.
2. `redelegateToValidators()`.

In this design, the first step (`addValidator()`) undelegates funds from currently active validators in preparation for redelegation to the new ones after a delay. The second step (`redelegateToValidators()`) then completes the process by delegating those funds to the new validators.

Problem Scenario: A critical edge case occurs when the admin begins adding a new validator but, before completing the redelegation phase, initiates the removal of another validator. Example sequence:

1. The admin calls `addValidator(A)`. During this process, a portion of delegated funds is undelegated from validator B to prepare for redelegation to A.
2. Before completing the process by calling `redelegateToValidators()`, the admin calls `initiateValidatorRemoval(B)` to remove B.
3. The admin then attempts to finalize the addition of A by calling `redelegateToValidators()`.

At this point, the undelegated funds previously taken from B are stuck - they cannot be withdrawn from the Monad staking contract because B has already been removed from the active validator list in `CoreVault`. These funds become permanently locked, and recovery would require a contract upgrade.

Additional sequence leading to the same issue:

1. `adminRebalanceInitiate()`.
2. `initiateValidatorRemoval()`.

Attempted Mitigation and Deadlock: If the admin attempts to fix the issue by completing B's removal process, this will also fail. The next step (`_executeValidatorUndelegation()`) will revert inside `_checkFreeAdminWid()` since `_coreVaultDelInfo.stake > 0`. This creates a deadlock - neither the validator addition nor removal can proceed, leaving both processes in a stuck state and the funds inaccessible.

Likelihood: This issue is more likely than it appears at first glance:

- It can easily occur when the admin is trying to replace an old validator (B) with a new one (A) and attempts to perform both operations in parallel.
- It may also occur when different operators manage separate validators, obscuring dependencies between them and making the sequencing error harder to spot.

Overall, this logic flaw exposes `CoreVault` to operational deadlocks and potential permanent fund locks if these processes are not carefully sequenced.

Recommendation: There should be mutual exclusivity between the processes of withdrawing from a validator for redelegation and removing that validator. Consider adding a call to `_checkFreeAdminWid()` within `_initiateValidatorRemoval()` so it will solve both this issue as well as finding "`gVault`: removing a validator in the middle of a rebalance will cause loss of funds and a denial of service".

Hydrogen Labs: Fixed in commit bf7a705e.

Cantina Managed: Fixed by implementing the auditor's recommendation.

3.1.4 Storage state of validators is not deleted properly leading to accounting issues

Severity: High Risk

Context: gVault.sol#L275

Description: The `gVault` contract maintains three mappings - `_totalSharesByValidator`, `_delegatedSharesOf`, and `_scaledPrincipalUnits` - that store state data per validator. Each of these mappings uses

the validator ID from the `gVault` contract as a key. Validator removal in the contract occurs through a three-step process:

1. `initiateValidatorRemoval()`.
2. `executeValidatorUndelegation()`.
3. `completeValidatorRemovalWithdrawal()`.

However, none of these steps currently clear or reset the relevant mapping entries. As a result, outdated data remains in storage even after a validator has been fully removed. This behavior has two main implications:

- Inaccurate view data: Functions such as `delegatedSharesOf()`, `totalSharesByValidator()`, and `maxWithdrawableFromGVault()` may return incorrect or misleading results, since they reference stale state variables from previously removed validators.
- Severe accounting inconsistencies:
  If a validator is later re-added using the same validator ID, the leftover data from its previous lifecycle may cause misaccounting, incorrect share allocations, or potentially even locked funds due to inconsistencies between the recorded shares and actual delegated balances.

Recommendation: Consider deleting the values mappings of `_totalSharesByValidator`, `_delegatedSharesOf`, and `_scaledPrincipalUnits` upon admin removal. `_totalSharesByValidator` is easy to handle, but as for `_scaledPrincipalUnits`, and `_delegatedSharesOf` which are nested mappings, we can use versionning and invalidate the version upon removal. Let's take `_scaledPrincipalUnits` as an example:

1. Define `_scaledPrincipalUnits` as `mapping(uint64 validatorId => mapping(uint64 version => mapping(address => uint256)))` `_scaledPrincipalUnits` instead;
2. Add an additional mapping of `mapping(uint64 validatorId => uint64 version) public validatorVersion;`.
3. When you want to delete all users for a given validator - `validatorVersion[validatorId]++;`.

Hydrogen Labs: Fixed in commit bf7a705e.

Cantina Managed: Fixed by adding a `REMOVED` mode for validators which will prevent re-addition in the case of `gVault` and will return 0 values for the impacted view functions.

3.2 Medium Risk

3.2.1 Precision Loss in gVault User Entitlement Calculation Due to Large Scale Factor

Severity: Medium Risk

Context: gVault.sol#L518

Description/Recommendation: In `gVault.maxWithdrawableFromGVault`, the calculation `Math.mulDiv(units, P, S)` can floor to 0 for valid user positions when `_gVaultScaleS` grows large from repeated rescaling/full rebalances (e.g., `S > units * P`). This falsely reports 0 withdrawable assets, preventing withdrawals even before full uint256 overflow in `S`. It's a separate issue from the overflow DoS, as it arises from integer division flooring rather than arithmetic overflow, and can occur after fewer rescales (e.g., 4-5).

Hydrogen Labs: Fixed in commit ece62aa5.

Cantina Managed: Fix verified.

3.2.2 Unbounded `gVault` scale growth causes overflow and DoS in admin rebalancing

Severity: Medium Risk

Context: gVault.sol#L380

Description: In `gVault.adminInitiateRebalanceBps`, the multiplier rescaling multiplies both `P` and `S` by `1e9` when `P` falls below a floor, and the 100% (full) rebalance path multiplies `S` by `1e9` while resetting `P`. This causes exponential growth of `S` that overflows `uint256` after a small number of events, making further rebalances revert and blocking admin liquidity management. After 6 full rebalances (bps=10_000), `S` overflows and the call reverts.

Impact:

- Admins lose the ability to rebalance, leading to a protocol liveness/DoS issue during liquidity events.
- It will also cause `undelegate` to revert, effectively blocking withdrawals.

Proof of Concept:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.30;

import {Test} from "forge-std/Test.sol";
import {console2} from "forge-std/console2.sol";
import {ERC1967Proxy} from "@openzeppelin/contracts/proxy/ERC1967/ERC1967Proxy.sol";
import {Magma} from "../src/Magma.sol";
import {gVault} from "../src/gVault.sol";
import {CoreVault} from "../src/CoreVault.sol";
import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";

// Minimal mock of the Monad staking precompile used by gVault
contract MockStaking {
    struct DelInfo {
        uint256 stake;
        uint256 acc;
        uint256 rewards;
        uint256 deltaStake;
        uint256 nextDeltaStake;
        uint64 deltaEpoch;
        uint64 nextDeltaEpoch;
    }

    mapping(uint64 => mapping(address => DelInfo)) private delegatorInfo;

    function delegate(uint64 valId) external payable returns (bool) {
        delegatorInfo[valId][msg.sender].stake += msg.value;
        return true;
    }

    function undelegate(uint64 /*valId*/, uint256 /*amount*/, uint8 /*withdrawId*/) external pure returns
    ↪  (bool) {
        return true;
    }

    function withdraw(uint64 /*valId*/, uint8 /*withdrawId*/) external pure returns (bool) {
        return true;
    }

    function claimRewards(uint64 /*valId*/) external pure returns (bool) {
        return true;
    }

    function getDelegator(uint64 valId, address delegator)
        external
        view
        returns (uint256, uint256, uint256, uint256, uint256, uint64, uint64)
    {
        DelInfo memory del = delegatorInfo[valId][delegator];
        return (del.stake, 0, 0, 0, 0, 0, 0);
    }

    function getWithdrawalRequest(uint64 /*validatorId*/, address /*delegator*/, uint8 /*withdrawId*/)
        external
        pure
        returns (uint256 withdrawalAmount, uint256 accRewardPerToken, uint64 withdrawEpoch)
    {
        return (0, 0, 0);
    }
}

contract GVaultRescaleOverflowPOC is Test {
    address internal constant PRECOMPILE = address(uint160(0x1000));

    Magma private magma;
    gVault private gvault;
    CoreVault private core;

    function setUp() public {
```

```
        // Etch MockStaking into the staking precompile address
        MockStaking staking = new MockStaking();
        vm.etch(PRECOMPILE, address(staking).code);

        // Deploy Magma implementation and proxy-initialize
        Magma magmaImpl = new Magma();
        bytes memory magmaInitData = abi.encodeWithSelector(
            Magma.initialize.selector,
            Magma.InitializeParams({
                asset: IERC20(address(0)),
                name: "Magma",
                symbol: "MAG",
                admin: address(this),
                rewardsFee: 0,
                withdrawalFee: 0,
                feeReceiver: address(this),
                redeemDelay: 0,
                mevRewardsInjector: address(this)
            })
        );
        ERC1967Proxy magmaProxy = new ERC1967Proxy(address(magmaImpl), magmaInitData);
        magma = Magma(payable(address(magmaProxy)));

        // Deploy CoreVault (not used in the PoC, but required to set non-zero addresses in Magma)
        CoreVault coreImpl = new CoreVault();
        bytes memory coreInitData = abi.encodeWithSelector(CoreVault.initialize.selector, address(magma), 0,
        ↪  10);
        ERC1967Proxy coreProxy = new ERC1967Proxy(address(coreImpl), coreInitData);
        core = CoreVault(payable(address(coreProxy)));

        // Deploy gVault implementation and proxy-initialize
        gVault gvImpl = new gVault();
        bytes memory gvInitData = abi.encodeWithSelector(gVault.initialize.selector, address(magma), 0);
        ERC1967Proxy gvProxy = new ERC1967Proxy(address(gvImpl), gvInitData);
        gvault = gVault(payable(address(gvProxy)));

        // Wire vaults into Magma (CoreVault and gVault addresses must be non-zero)
        magma.initVaults(address(core), address(gvault));

        // Add a single validator to gVault so admin rebalances iterate over at least one validator
        gvault.addValidator(1);
    }

    // Fast path PoC: 6 full rebalances (bps = 100%) overflow S on the 6th due to S *= 1e9 each time
    function test_FullRebalance_OverflowOnSixth() public {
        // 5 successful full rebalances
        for (uint256 i = 0; i < 5; ++i) {
            gvault.adminInitiateRebalanceBps(10_000);
            // allow next rebalance
            gvault.adminCompleteRebalance();
        }

        // 6th full rebalance should revert on S *= 1e9 overflow (Solidity panic on arithmetic overflow)
        vm.expectRevert();
        gvault.adminInitiateRebalanceBps(10_000);
    }


    function test_PartialRebalance_OverflowAfterThirdRescale() public {
        uint16 bps = 1000; // 10%
        uint256 rescaleCount = 0;
        uint256 lastS = gvault.gVaultScaleS();


        for (uint256 i = 0; i < 100000 && rescaleCount < 3; ++i) {
            gvault.adminInitiateRebalanceBps(bps);
            uint256 sNow = gvault.gVaultScaleS();
            if (sNow > lastS) {
                unchecked {
                    ++rescaleCount;
                }
                lastS = sNow;
            }
            gvault.adminCompleteRebalance();
        }
        assertEq(rescaleCount, 3, "Expected to reach 3 rescale events first");
```

```
        vm.expectRevert();
        gvault.adminInitiateRebalanceBps(bps);
    }
}
```

Recommendation: Consider replacing the current accounting that's based on multiplication with accounting that's storing and increasing the logarithm itself. Upon `mulDiv` operations, the difference between the exponents of `P` (`_gVaultMultiplierP`) and `S` (`_gVaultScaleS`) can be used instead.

Hydrogen Labs: Fixed in commit ece62aa5.

Cantina Managed: Fix verified.

3.2.3 Interim double-counting in `totalAssets` during validator removal initiation

Severity: Medium Risk

Context: VaultBase.sol#L501-L519

Description: The totalAssets() function temporarily overcounts the total value by the stake of the removed validator after calling `initiateValidatorRemoval`. This happens because _initiateValidatorRemoval eagerly adds the validator's stake to `_totalPendingRedelegation` without an offsetting subtraction in `_cachedTotalNetPendingDelegations`, causing inflation (e.g., from 100 ETH to ~200 ETH in test, or to 100 + `stake(v1)`). The overcount corrects after executeValidatorUndelegation due to pending readjustment or after a cache refresh, but the interim window can mislead upstream contracts (e.g., under-minting shares in Magma).

Proof of Concept:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.30;

import {Test} from "forge-std/Test.sol";
import {console2} from "forge-std/console2.sol";
import {CoreVault} from "../src/CoreVault.sol";
import {VaultBase} from "../src/VaultBase.sol";
import {ERC1967Proxy} from "@openzeppelin/contracts/proxy/ERC1967/ERC1967Proxy.sol";
import {IMonadStaking} from "../interfaces/IMonadStaking.sol";
import {DelInfo} from "../src/MagmaDelegationModule.sol";
import {IMagma} from "../interfaces/IMagma.sol";
import {Magma} from "../src/Magma.sol";
import {IGVault} from "../interfaces/IGVault.sol";
import {gVault} from "../src/gVault.sol";
import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";

// Local enum for ValidatorStatus (workaround)
enum ValidatorStatus {
    NONE,
    PAUSED,
    UNDELEGATING
}

// (No Magma mock: use real Magma)

// Simplified Mock Staking for stake setting and getDelegator
contract MockStaking {
    mapping(uint64 => mapping(address => DelInfo)) private delegatorInfo;

    function delegate(uint64 valId) external payable returns (bool) {
        delegatorInfo[valId][msg.sender].stake += msg.value;
        return true;
    }

    function getDelegator(uint64 valId, address delegator)
        external
        view
        returns (uint256, uint256, uint256, uint256, uint256, uint64, uint64)
    {
        DelInfo memory del = delegatorInfo[valId][delegator];
```

```
        console2.log("getDelegator called for valId", valId, "delegator", delegator);
        console2.log("Returning stake:", del.stake);
        return (del.stake, 0, 0, 0, 0, 0, 0);
    }

    function setInitialStake(uint64 valId, address delegator, uint256 stake) external {
        delegatorInfo[valId][delegator].stake = stake;
    }
}

// (No gVault mock: use real gVault)

contract ValidatorInitiationInflationPOC is Test {
    CoreVault private vault;
    Magma private magma;
    gVault private gvault;
    address internal constant PRECOMPILE = address(uint160(0x1000));

    uint64 constant TEST_VAL_ID_1 = 1;
    uint64 constant TEST_VAL_ID_2 = 2;
    uint256 constant INITIAL_STAKE = 100 ether;

    function setUp() public {
        // Etch MockStaking to PRECOMPILE (only allowed mock)
        MockStaking staking = new MockStaking();
        vm.etch(PRECOMPILE, address(staking).code);

        // Deploy Magma impl
        Magma magmaImpl = new Magma();

        // Proxy for Magma
        bytes memory magmaInitData = abi.encodeWithSelector(
            Magma.initialize.selector,
            Magma.InitializeParams({
                asset: IERC20(address(0)), // Mock asset, but since we use depositMON, perhaps not needed
                name: "Magma",
                symbol: "MAG",
                admin: address(this),
                rewardsFee: 0,
                withdrawalFee: 0,
                feeReceiver: address(this),
                redeemDelay: 0,
                mevRewardsInjector: address(this)
            })
        );
        ERC1967Proxy magmaProxy = new ERC1967Proxy(address(magmaImpl), magmaInitData);
        magma = Magma(payable(address(magmaProxy)));

        // Deploy CoreVault via proxy and initialize with magma
        CoreVault cvImpl = new CoreVault();
        bytes memory cvInitData = abi.encodeWithSelector(
            CoreVault.initialize.selector,
            address(magma),
            0,   // epochSeconds
            10   // maxValidatorPerBatch
        );
        ERC1967Proxy cvProxy = new ERC1967Proxy(address(cvImpl), cvInitData);
        vault = CoreVault(payable(address(cvProxy)));

        // Deploy gVault via proxy and initialize with magma
        gVault gvImpl = new gVault();
        bytes memory gvInitData = abi.encodeWithSelector(
            gVault.initialize.selector,
            address(magma),
            0 // epochSeconds
        );
        ERC1967Proxy gvProxy = new ERC1967Proxy(address(gvImpl), gvInitData);
        gvault = gVault(payable(address(gvProxy)));

        // Set vaults in Magma
        vm.prank(address(this)); // As admin
        magma.initVaults(address(vault), address(gvault));

        // First add validators before deposit
        vm.startPrank(address(this));
        vault.addValidator(TEST_VAL_ID_1);
```

```
        vault.addValidator(TEST_VAL_ID_2);
        vm.stopPrank();

        // Then deposit through Magma (natural flow): two deposits of 50 ETH
        vm.deal(address(this), INITIAL_STAKE / 2);
        magma.depositMON{value: INITIAL_STAKE / 2}(address(this), 0);
        // Force cache refresh so the second deposit sees updated stakes
        vault.refreshCache();
        vm.deal(address(this), INITIAL_STAKE / 2);
        magma.depositMON{value: INITIAL_STAKE / 2}(address(this), 0);

        // Final refresh to baseline after both deposits
        vault.refreshCache();

        // ===== Summary after deposits =====
        uint256 initialAssets = vault.totalAssets();
        assertEq(initialAssets, INITIAL_STAKE, "Initial totalAssets incorrect");
        console2.log("==== After deposits ====");
        console2.log("CoreVault.totalAssets:", initialAssets);
        (uint256 s1,,,,,,) = MockStaking(PRECOMPILE).getDelegator(TEST_VAL_ID_1, address(vault));
        (uint256 s2,,,,,,) = MockStaking(PRECOMPILE).getDelegator(TEST_VAL_ID_2, address(vault));
        console2.log("Validator 1 stake:", s1);
        console2.log("Validator 2 stake:", s2);
        console2.log("========================");
    }

    function test_InitiationCausesInflation() public {
        // Snapshot state before initiation
        uint256 initialAssets = vault.totalAssets();
        (uint256 stakeToRemove,,,,,,) = MockStaking(PRECOMPILE).getDelegator(TEST_VAL_ID_1, address(vault));
        uint256 expectedInflated = initialAssets + stakeToRemove;
        console2.log("==== Before initiateValidatorRemoval ====");
        console2.log("Initial totalAssets:", initialAssets);
        console2.log("Stake on removed validator (v1):", stakeToRemove);
        console2.log("Expected inflated totalAssets after initiation (initial + stake(v1)):", expectedInflated);

        // Initiate removal of v1
        vm.prank(address(this));
        vault.initiateValidatorRemoval(TEST_VAL_ID_1);

        // Check for inflation: Expected = initialAssets + stakeToRemove (double-count)
        uint256 assetsAfterInit = vault.totalAssets();
        console2.log("==== After initiateValidatorRemoval ====");
        console2.log("Observed totalAssets:", assetsAfterInit);
        console2.log("Inflation delta:", assetsAfterInit - initialAssets);
        console2.log("Reason: _initiateValidatorRemoval adds stake(v1) to totalPendingRedelegation while
        ↪  stake(v1) remains in cachedTotalAssets (no offset).\n");
        assertEq(assetsAfterInit, expectedInflated, "totalAssets inflated incorrectly after initiation");
    }
}
```

Impact:

- Users depositing during the interim window (after initiateValidatorRemoval and before executeValidatorUndelegation or a cache refresh) can receive fewer shares than they should, since totalAssets is inflated and ERC-4626-style minting divides by a larger denominator.
- External systems relying on CoreVault's totalAssets (e.g., TVL reporting, risk checks, caps) will observe a temporarily inflated value.

Recommendation: Call `_trackCachedUndelegation(_totalStakedToValidator)` there to net the accounting to zero during the queue.

Hydrogen Labs: Fixed in commit 3bd3e832.

Cantina Managed: Fix verified.

3.2.4 Lack of Maximum Validator Count Cap Risks Gas Exhaustion in Loops

Severity: Medium Risk

Context: CoreVault.sol#L92

Description: The contracts `CoreVault` and `gVault` allow unlimited validators to be added via `addValidator` and `addValidators` (in CoreVault). Several functions loop over all validators, such as:

- In `CoreVault`:
  - `_claimAndCompoundRewards` (claims from each validator).
  - `_redelegateInitiate` (calculates and undelegates excess per validator).
  - `_completeAllPendingRedelegationWithdrawals` (processes withdrawals per validator).
  - Various getters and distributors like `getTotalDelegated` and `_distributeToNextValidator`.
- In `gVault`:
  - Similar patterns in reward claiming and delegation functions (e.g., `claimAndCompoundRewards`, loops over validators).

Each iteration incurs gas costs (e.g., storage reads, precompile calls). With too many validators (e.g., >100-200, depending on operations), these could hit block gas limits (~30M gas), causing reverts and preventing critical operations like rebalancing or reward compounding.

Example vulnerable loop in `CoreVault._claimAndCompoundRewards`:

```
for (uint256 _i = 0; _i < _validators.length; ++_i) {
    // Claim per validator - gas scales with validator count
}
```

Without a cap, an excessive number of validators could cause functions with O(n) loops (e.g., reward claiming, rebalancing) to exceed block gas limits, leading to permanent DoS for those operations. This is caused by admins adding too many validators, potentially bricking vault functionality.

Recommendation: While the obvious and simple fix would be to add a configurable cap to the function that adds validators, another approach would be to avoid iterating through all validators. Instead, add an option to either iterate over all validators or only those provided as input parameters to the function.

Hydrogen Labs: Acknowledged.

Cantina Managed: Acknowledged.

3.2.5 Early withdrawers from a slashed validator incur significantly smaller losses than others

Severity: Medium Risk

Context: Magma.sol#L475

Description: The `Magma` contract compensates withdrawers who receive fewer assets than expected due to slashing. The intent is to "socialize" losses by minting additional shares to the withdrawer. Conceptually, this makes sense - but the current implementation results in uneven loss distribution.

Specifically, the number of shares minted is calculated as `convertToShares(request.assets - totalWithdrawn)`. This causes early withdrawers to incur significantly smaller losses than other `Magma` token holders, meaning losses are not uniformly socialized across all stakers.

Proof of Concept:

- Two users deposit 100 MON each - Alice into `gVault`, Bob into `CoreVault`.
- Magma total supply = 200.
- Magma total assets = 200.
- A 10 MON slashing occurs from the `gVault` validator (no fees).

Alice withdraws first:

- She receives 90 MON immediately.
- Compensation shares = 10 * (100 / 100) = 10.

If she later redeems those shares:
`assets = 10 * 100 / 110 = 9.09 MON`.

Alice's total = 90 + 9.09 = 99.09 MON, meaning she only lost 0.91 MON (While Bob had lost 10-0.91=9.09) instead of her fair share of 5 MON.

Another unintended consequence of the current compensation mechanism is that users who stake after a slashing event may also bear part of the associated loss, even though it occurred before their staking period. This results in an unfair redistribution of losses. Addressing this issue is considered out of scope for this engagement.

Recommendation: Consider storing the total assets value in the time of creating the redeem request inside the `RedeemRequests` object. This value should be then used to determine the amount of shares for compensation instead. Please note the edge case of 0 total assets in which the shares calculation should indeed be `convertToShares(request.assets - totalWithdrawn)`.

Hydrogen Labs: Fixed in commit e1434775.

Cantina Managed: Fix verified.

3.2.6 Withdrawn amount after undelegation might be less than expected, potentially causing reverts and unexpected behavior

Severity: Medium Risk

Context: CoreVault.sol#L441

Description: As per the Monad staking documentation, undelegated funds remain subject to slashing during the withdrawal delay period. However, in several parts of the code, the logic references the requested withdrawal amount rather than the actual withdrawn amount. This discrepancy can lead to reverts or unexpected behavior if the contract attempts to transfer more funds than are available.

```
// _completeAllPendingRedelegationWithdrawals()
// ...
(bool _exists, uint256 _amount,,) = _getWithdrawalRequest(_valId, address(this), ADMIN_WID);
// ...
```

```
// adminCompleteRebalance()
// ...
(bool exists, uint256 amt,,) = _getWithdrawalRequest(_valId, address(this), ADMIN_WID);
// ...
```

```
// _completeValidatorRemovalWithdrawal()
// ...
(bool _exists, uint256 _withdrawalAmount,,) = _getWithdrawalRequest(_valId, address(this), ADMIN_WID);
// ...
```

```
// _completeUserWithdrawal()
// ...
(, uint256 _availableAmount,,) = _getWithdrawalRequest(_valId, address(this), _withdrawalId);
// ...
```

Recommendation: Consider calculating the delta between the balance before and after the call to `_withdraw()` and use it instead.

Hydrogen Labs: Fixed in commit bf7a705e

Cantina Managed: Fixed by implementing the auditor's recommendation.

3.2.7 `gVault.undelegate()`: Only the original buyers of validators shares will be able to redeem

Severity: Medium Risk

Context: gVault.sol#L274

Description: The `Magma.requestRedeemGVault()` function allows holders of Magma tokens - who also own shares in a specific validator - to redeem their position back into the underlying `asset()` tokens.

However, since Magma tokens are transferable, an issue arises when these tokens change hands. A recipient of transferred Magma tokens who attempts to redeem through `requestRedeemGVault()` will fail, because during the call to `gVault.undelegate()`, only the original buyer (the address that made the delegation) is authorized to perform the undelegation.

This restriction effectively locks liquidity for secondary holders of Magma tokens tied to a specific validator. While they can still redeem through the `CoreVault` by calling `requestRedeem()`, doing so reduces the total pool available for other Magma token holders to claim.

The impact is somewhat limited, as the admin can eventually mitigate the situation by removing the affected validator from the `gVault`. When this occurs, the validator's delegated funds are transferred to the `CoreVault`, restoring liquidity.

However, this approach requires the admin to actively monitor for affected validators and intervene manually when such liquidity lockups occur - a process that is reactive, operationally heavy, and prone to delays. Moreover, each removal results in the loss of that validator's staking participation, which can impact network performance and yield distribution.

Recommendation: Consider removing the restriction that only the original delegator can call `undelegate()`. Allowing any current holder of Magma tokens to redeem through `gVault` would ensure liquidity consistency, reduce administrative burden, and prevent user experience issues when tokens are transferred.

Hydrogen Labs: Acknowledged. This is intended functionality.

Cantina Managed: Acknowledging this issue as "intended functionality" implies accepting an operational trade-off: the system must be continuously monitored to detect and manually undelegate from validators whose liquidity becomes locked, even if those validators are otherwise performing well. This introduces ongoing maintenance overhead and the risk of delayed user redemptions.

3.2.8 `gVault.claimAndCompoundRewards()` to revert in case `_remaining = 0`

Severity: Medium Risk

Context: gVault.sol#L483

Description: `gVault.claimAndCompoundRewards()` allows anyone to claim and compound rewards for all validators registered in the `gVault` contract. During its execution, it calls `_claimAndCompoundRewards()` in a loop as we can see in the following code snippet:

```
function claimAndCompoundRewards() external {
    uint64[] memory _list = getValidators();
    for (uint256 i = 0; i < _list.length; ++i) {
        uint64 _valId = _list[i];
        _claimAndCompoundRewards(_valId);
    }
    _updateLastRewardsClaimTimestamp();
}
```

In case any of the calls revert the entire call reverts and rewards will not be claimed. Now let's examine scenarios in which `_claimAndCompoundRewards()` will fail - the main unexpected scenario is when `_remaining=0`, in this case the call to `_delegate()` will revert with the error of `ErrZeroAssets` as we can see in the following code snippets:

```
function _claimAndCompoundRewards(uint64 _valId) internal {
    uint256 _startingBalance = address(this).balance;

    uint256 _before = address(this).balance;
    _claim(_valId);
    emit RewardsClaimed(_valId, address(this).balance - _before);
    uint256 _endingBalance = address(this).balance;
    uint256 _rewards = _endingBalance - _startingBalance;

    uint256 _fee = _calculateRewardsFeeAndSend(_rewards);

    uint256 _remaining = _rewards - _fee;
    _delegate(_valId, _remaining);
}
```

```
function _delegate(uint64 valId, uint256 amount) internal {
    if (amount == 0) revert ErrZeroAssets();
    bool success = STAKING.delegate{value: amount}(valId);
    if (!success) revert ErrDelegateFailed();
}
```

`_remaining` will be 0 either in case there are no rewards for at least one validator at the time of calling, or in case `_rewards = _fee`. The main impact here is that rewards can not be claimed unless by removing validators and in addition, functions like `gVault.delegate()` and `gVault.undelegate()` might revert on `ErrRewardsClaimOverdue` until `claimAndCompoundRewards()` can be called successfully again.

Recommendation: Consider skipping the call to `_delegate()` without reverting in case `_remaining == 0`.

Hydrogen Labs: Fixed in commit bf7a705e.

Cantina Managed: Fixed by implementing the auditor's recommendation.

3.3 Low Risk

3.3.1 Inconsistent Updates to `lastRebalanceTimestamp` in Rebalancing Functions

Severity: Low Risk

Context: CoreVault.sol#L49

Description: Functions `addValidator` and `addValidators` initiate rebalancing via `_redelegateInitiate()` but do not update `lastRebalanceTimestamp`. This bypasses the `onlyAfterEpoch` modifier's time guard, potentially allowing multiple initiations within the same epoch and disrupting rebalance timing.

```
function addValidator(uint64 _valId) external onlyAdmin onlyAfterEpoch {
    _registerValidator(_valId);
    _refreshCache();
    _redelegateInitiate();
}
```

(*Similarly for `addValidators`*).

Additionally, in the admin rebalance path (`adminRebalanceInitiate` followed by `redelegateToValidators`), `setLastRebalanceTimestamp(block.timestamp)` is called twice: once at initiation (*line 124 in `adminRebalanceInitiate`*) and again at completion (*line 420 in `_redelegateRedistribute`*). This redundancy overwrites the initiation timestamp with the completion one, which may be intentional to reset the epoch timer but creates inconsistency with the addValidator paths (no updates).

Recommendation: For consistency, update `lastRebalanceTimestamp`.

Hydrogen Labs: Fixed in commit f05c4e35.

Cantina Managed: Fix verified.

3.3.2 Delegation Cap Not Enforced on Last Validator During Stake Distribution

Severity: Low Risk

Context: CoreVault.sol#L578-L596

Description: In `CoreVault.sol`, the `_distributeStakeToValidatorsAscending` function (around lines 578-595) attempts to cap the delegation amount to each validator using `_maxAllowedFromValidator`. However, the cap is only enforced if the amount exceeds the max and it's not the last validator in the sorted list (_i < _sortedValidators.length - 1). This allows the last validator to receive an uncapped amount, potentially violating the intended per-validator limit and leading to imbalanced delegations.

Example:

- Setup:
  - TotalActiveStake = 2,000 $\Rightarrow$ cap per validator per call = 2,000/20 = 100.
  - Sorted validators (by active stake): V1=600, V2=700, V3=700.
  - Injected `_amount` = 450.
- Loop:
  - Step 1 ($i = 0$, $V1$): remaining=450 $\rightarrow$ `maxAllowed= min(450, 100) = 100` $\rightarrow$ delegate 100 $\rightarrow$ remaining=350.

- Step 2 ($i = 1$, $V2$): remaining=350 → `maxAllowed= min(350, 100) = 100` → delegate 100 → remaining=250.
- Step 3 ($i = 2$, $V3$ last): remaining=250 → `maxAllowed= min(250, 100) = 100` → last-index bypasses cap → delegate 250 → remaining=0.

- Result:
  - V1: +100 (to 700), V2: +100 (to 800), V3: +250 (to 950).
  - Issue: V3 received 250 > cap 100 (exceeds cap by 150) in a single call.

Recommendation: Always apply the cap to every validator:

```
if (_amountToValidator > _maxAllowedFromValidator) {
    _amountToValidator = _maxAllowedFromValidator;
}
```

Hydrogen Labs: Acknowledged.

Cantina Managed: Acknowledged.

3.3.3 Missing Enforcement of Minimum Interval in `setDelegatorInfoUpdateInterval`

Severity: Low Risk

Context: VaultBase.sol#L348-L350

Description: The function `setDelegatorInfoUpdateInterval` in `src/VaultBase.sol` is documented to require an `_interval` between 1 minute and 24 hours. However, the code only checks the upper limit (`if (_interval > 24 hours) revert`), allowing values below 1 minute (including 0). This discrepancy could enable invalid configurations, potentially causing excessive cache updates or other unexpected behavior.

Recommendation: Update the function to enforce the minimum limit by adding a check, such as `if (_interval < 1 minutes) revert ErrInvalidAmount(_interval);`.

Hydrogen Labs: Fixed in commit 9f6a7ec6.

Cantina Managed: Fix verified.

3.3.4 Incorrect Return Value on Failed Rewards Fee Transfer

Severity: Low Risk

Context: VaultBase.sol#L762

Description: In `VaultBase.sol::_calculateRewardsFeeAndSend(uint256 _totalRewards)`, the function calculates a `_fee` and attempts to transfer it to the fee receiver. If the transfer fails (e.g., due to the receiver rejecting the funds), it emits `RewardsFeeTransferFailed` but still returns the calculated `_fee` value. This misleads callers (e.g., `_claimAndCompoundRewards` in `gVault.sol`), which subtract the returned `_fee` from rewards before re-delegating, even though the fee was not actually sent. As a result, the contract retains the untransferred fee amount but delegates a reduced remainder, leading to funds being stuck in the contract.

Recommendation: Adjust the return value on failure to reflect reality.

```
// src/VaultBase.sol
function _calculateRewardsFeeAndSend(uint256 _totalRewards) internal returns (uint256 _fee) {
    VaultBaseStorage storage $ = _getVaultBaseStorage();

    _fee = Math.mulDiv(_totalRewards, $._magma.rewardsFee(), BASE_BPS, Math.Rounding.Ceil);
    if (_fee > 0) {
        (bool _ok,) = $._magma.feeReceiver().call{value: _fee}("");
        if (!_ok) {
            emit RewardsFeeTransferFailed(_fee);
            _fee = 0;  // Adjust to reflect failed transfer
        } else {
            emit RewardsFeeTransferSuccess(_fee, $._magma.feeReceiver());
        }
    }
```

```
    return _fee;
}
```

Hydrogen Labs: Fixed in commit 00412cc8.

Cantina Managed: Fix verified.

3.3.5 Incorrect Owner and Shares in Withdraw Event Emission

Severity: Low Risk

Context: Magma.sol#L497

Description: In `Magma.sol::_redeem()`, the `Withdraw` event is emitted with `address(this)` as the `owner` parameter and incorrect Shares, but the actual owner whose withdrawal is being completed is the `owner` retrieved from the pending request (`$._pendingRedeemRequests[controller][requestId].owner`). This misrepresents the event data, violating EIP-4626 conventions where `owner` should be the account whose shares are burned/redeemed.

Recommendation: Update the event emission to use the correct `owner` and shares:

```
// src/Magma.sol
// ... existing code ...
emit Withdraw(_msgSender(), receiver, owner, totalWithdrawnAfterFee, request.shares);
// ... existing code ...
```

Hydrogen Labs: Fixed in commit c9e0de09.

Cantina Managed: Fix verified.

3.3.6 Missing cache update on rewards compounding/injection

Severity: Low Risk

Context: gVault.sol#L483

Description: `delegate()` calls `_trackCachedDelegation(amount)` after `_delegate`, but `_claimAndCompoundRewards()` and `injectRewards(uint64 _valId)` do not. This causes `IGVault.totalAssets()` (and thus `Magma.totalAssets()`) to be understated until the next cache refresh.

Recommendation: `_trackCachedDelegation` with the actually delegated amount:

- After `_delegate(_valId, _remaining);` in `_claimAndCompoundRewards`:

  ```
  _trackCachedDelegation(_remaining);
  ```

- After `_delegate(_valId, msg.value);` in `injectRewards(uint64 _valId)`:

  ```
  _trackCachedDelegation(msg.value);
  ```

Hydrogen Labs: Fixed in commit 2e808af8.

Cantina Managed: Fix verified.

3.3.7 `gVault.delegate()` will revert for validators with a default cap value and `magma().totalAssets()` is 0

Severity: Low Risk

Context: gVault.sol#L219

Description: The `gVault` contract allows the admin to configure a deposit cap for each validator. If the cap is explicitly set, it limits the maximum amount that can be delegated to that validator. If left unset (`_validatorCap[_valId] == 0`), the system is designed to fall back to a default cap derived as a percentage of `Magma`'s total assets.

However, when `magma().totalAssets() == 0` - such as during early deployment or after a full withdrawal - this fallback logic fails. In this scenario, the computed cap remains zero, and any call to `delegate()` for that validator reverts with `ErrCapZero`. As a result, the validator becomes undelegatable until the admin

manually sets a nonzero cap, defeating the intended behavior of having a dynamic, asset-based default limit.

Recommendation: Handle the edge case where both `magma().totalAssets()` and `_validatorCap[_valId]` are zero by returning a large default value (e.g., `type(uint256).max` or a high predefined limit). This ensures delegation remains functional during initialization or low-liquidity periods while maintaining safe default behavior.

Hydrogen Labs: Fixed in bf7a705e.

Cantina Managed: Fixed by implementing a default cap of `DEFAULT_MAX_CAP = 100_000 ether` for the edge case described in the issue.

3.3.8 `VaultBase`: re-registrating a validator in the middle of its removal leads to unintended consequences

Severity: Low Risk

Context: VaultBase.sol#L486

Description: The `VaultBase` contract allows the admin to manage validators through a lifecycle that includes registration and removal. A validator can be registered using `_registerValidator()`, and later removed via a three-step process:

1. `_initiateValidatorRemoval()`.
2. `_executeValidatorUndelegation()`.
3. `_completeValidatorRemovalWithdrawal()`.

However, there is a subtle edge case that could lead to corrupted state data. If the admin calls `_initiateValidatorRemoval()` to begin removing a validator but later changes their mind and calls `_registerValidator()` again for the same validator ID, the system will behave unexpectedly.

Specifically:

- The validator will be added back to the validator list and whitelist successfully.
- However, it will remain in a PAUSED state, since `_initiateValidatorRemoval()` already set its status as such.
- Additionally, the mappings `_pendingRedelegateByValidator` and `_totalPendingRedelegation` may still hold non-zero values from the previous removal attempt, resulting in corrupted or inconsistent accounting data.

Recommendation: Consider changing `_registerValidator()` to revert in case of a validator with `PAUSED` status.

Hydrogen Labs: Fixed in commit 71054e65.

Cantina Managed: Fixed by changing `gVault.addValidator()` and `CoreVault._addValidator()` to revert for validators with `PAUSED` status.

3.4 Gas Optimization

3.4.1 `gVault.adminInitiateRebalanceBps()` calls `setLastRebalanceTimestamp()` redundantly

Severity: Gas Optimization

Context: gVault.sol#L418

Description: The function `setLastRebalanceTimestamp()` is called redundantly within `adminInitiateRebalanceBps()`, even though `_lastRebalanceTimestamp` is never referenced elsewhere in the contract.

Recommendation: Consider removing the unnecessary `setLastRebalanceTimestamp()` call to simplify the logic and reduce gas usage.

Hydrogen Labs: Fixed in commit bf7a705e.

Cantina Managed: Fixed by implementing the auditor's recommendation.

3.4.2 `_distributeToNextValidator()`: Redundant double array sorting

Severity: Gas Optimization

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

Description: The `_distributeToNextValidator()` function sorts the validators array in ascending order by staked amount to achieve balanced fund distribution among `CoreVault` validators. However, before this ascending sort, the array is redundantly sorted in descending order, as shown below:

```
// _distributeToNextValidator()
(ValidatorAmount[] memory _sortedValidators, uint256 _totalActiveStake) =
    _getSortedValidatorsByActiveStakeDescendingWithTotal();
_sort(_sortedValidators);
```

This double sorting introduces unnecessary computation and gas costs without affecting the final outcome.

Recommendation: Remove the redundant descending sort (`_getSortedValidatorsByActiveStakeDescendingWithTotal()`) to simplify logic and reduce gas consumption.

Hydrogen Labs: Fixed in commit bf7a705e.

Cantina Managed: Fixed by implementing the auditor's recommendation.

3.4.3 Remove unused imports of `BitMapLib`

Severity: Gas Optimization

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

Description: Both `gVault` and `CoreVault` import `BitMapLib` although it is unused.

Recommendation: Consider removing those imports.

Hydrogen Labs: Fixed in commit bf7a705e.

Cantina Managed: Fixed by implementing the auditor's recommendation.

3.4.4 `gVault._claimAndCompoundRewards()` optimizations

Severity: Gas Optimization

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

Description:

```
function _claimAndCompoundRewards(uint64 _valId) internal {
    uint256 _startingBalance = address(this).balance;

    uint256 _before = address(this).balance;
    _claim(_valId);
    emit RewardsClaimed(_valId, address(this).balance - _before);
    uint256 _endingBalance = address(this).balance;
    uint256 _rewards = _endingBalance - _startingBalance;

    uint256 _fee = _calculateRewardsFeeAndSend(_rewards);

    uint256 _remaining = _rewards - _fee;
    _delegate(_valId, _remaining);
}
```

As we can see, `_before` is redundant, and `_startingBalance` can be used instead. In addition, `RewardsClaimed` can use the value of `_rewards` instead of re-calculating the difference.

Hydrogen Labs: Fixed in commit bf7a705e.

Cantina Managed: Fixed by implementing the auditor's recommendation.

3.4.5 Avoid unnecessary state updates when no undelegation occurs

Severity: Gas Optimization

Context: CoreVault.sol#L396

Description: In the `_redelegateInitiate()` function, `setTotalPendingRedelegation(totalPendingRedelegation() + _totalToUndelegate)` and `_trackCachedUndelegation(_totalToUndelegate)` are executed even when `_totalToUndelegate` equals zero. This leads to unnecessary gas consumption due to redundant storage operations and function calls.

Recommendation: Execute these calls only when `_totalToUndelegate` is greater than zero to save gas.

Hydrogen Labs: Fixed in commit bf7a705e.

Cantina Managed: Fixed by implementing the auditor's recommendation.

3.4.6 `refreshCache()` gas optimizations

Severity: Gas Optimization

Context: CoreVault.sol#L407

Description:

1. During the execution of `redelegateToValidators()`, `_refreshCache()` is called twice redundantly, and therefore can be removed.
2. `CoreVault._redelegateInitiate()`: `_getCachedTotalStakedToValidator()` can be removed and `cachedDelegatorInfo()` can be used instead.

Hydrogen Labs: Fixed in commit bf7a705e.

Cantina Managed: Fixed by implementing the auditor's recommendation.

3.5 Informational

3.5.1 Importance of Validator Activity Monitoring

Severity: Informational

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

Description: Validator uptime, performance, and behavior have a direct impact on reward accuracy and user trust in the Magma Liquid Staking product on Monad. Continuous monitoring ensures that validator-related incidents-such as missed attestations, downtime, or slashing risks-are detected and mitigated early. Because staked assets are pooled and rewards are distributed proportionally among users, even short periods of validator underperformance can reduce aggregate yield and create fairness discrepancies in reward accounting.

Recommendation: Implement comprehensive off-chain monitoring to continuously track validator activity and health. At a minimum, the following aspects should be monitored:

- Uptime and block proposal rate: Detect when a validator goes offline or fails to propose/validate blocks within expected intervals.
- Attestation or participation rate: Identify performance degradation in consensus participation.
- Slashing and penalties: Track on-chain events or conditions that could trigger slashing, double-signing, or downtime penalties.
- Stake and delegation balance: Monitor changes in staked amounts or validator commission rates that could affect reward distribution.
- Network performance metrics: Observe latency, missed blocks, or increased message propagation times that may indicate connectivity or performance issues.
- Alerting and reporting: Set up automated notifications (e.g., via webhook, Telegram, or email) for any abnormal behavior and maintain a dashboard for ongoing visibility.

Such monitoring helps operators react quickly to incidents, maintain fair reward allocation, and preserve the reliability and reputation of the Magma Liquid Staking system.

Hydrogen Labs: Acknowledged.

Cantina Managed: Acknowledged.

3.5.2 Uncompounded rewards excluded from total assets calculation may lead to fairness issues

Severity: Informational

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

Description: The variable `_delInfo.rewards` is not included in the total assets calculation. This is likely intentional, as rewards are periodically compounded into the total assets through a dedicated function. However, this design introduces a fairness concern: if the compounding function is not called frequently enough, new depositors may indirectly benefit from rewards that were generated before their deposit, despite not participating in the staking period that produced those rewards.

Recommendation: Although this behavior does not present a direct vulnerability, it is recommended to:

- Ensure the compounding function is invoked regularly, either automatically or via enforced operational procedures.
- Document the expected compounding frequency and its impact on fairness and yield distribution.

Maintaining consistent compounding intervals will help ensure a fair distribution of rewards among all participants.

Hydrogen Labs: Acknowledged. We will invoke the compounding function regularly.

Cantina Managed: Acknowledged.

3.5.3 `_claim` Function Reverts on Non-Critical Failures, causing Unnecessary Reverts in Reward Handling

Severity: Informational

Context: MagmaDelegationModule.sol#L53

Description: The internal `_claim(uint64 valId)` function in `MagmaDelegationModule.sol` calls the staking precompile's `claimRewards(valId)` and reverts with `ErrDelegateFailed` if it returns `false`. However, upon review of the precompile implementation, `claimRewards` does not return `false` in any case-instead, it always returns `true` on success (including when there are no rewards to claim, i.e., zero balance) or reverts on errors (e.g., invalid input or other failures). This means the `!success` check in `_claim` is never triggered, as reverts propagate directly, causing the entire transaction to fail with the precompile's error instead of the custom `ErrDelegateFailed`.

This behavior can lead to unhandled reverts in scenarios where a graceful failure or continuation might be preferable, such as during batched operations across multiple validators. For example, a revert on a single validator could halt routine processes like reward compounding, effectively causing a temporary DoS for the operation.

Call Sites: All call sites of `_claim(uint64 valId)`:

1. `CoreVault._claimAndCompoundRewards()` (loop over validators):
    - Context: Routine compounding across all validators.
    - Impact of unhandled revert: Blocks the entire process if any single claim reverts.
2. `GVault._claimAndCompoundRewards(uint64 _valId)` (per-validator):
    - Context: Claims/compounds for a specific validator in targeted/batch ops.
    - Impact of unhandled revert: Fails the operation for that validator, potentially disrupting batch workflows.
3. `VaultBase._claimValidatorRewards(uint64 _valId)` (guarded by `rewards > 0`):
    - Context: Claims during validator removal.

- Impact of unhandled revert: Could fail the removal process if an unexpected revert occurs (though guarded by rewards > 0, which reduces likelihood).

Root Cause: The root issue is that `_claim` assumes `claimRewards` might return `false` on non-critical cases, but it actually reverts on errors and always returns `true` otherwise. As a result:

- The `bool success` assignment only captures successful calls (always `true`).
- Reverts from the precompile propagate unhandled, bypassing the custom error logic.

```
function _claim(uint64 valId) internal {
    bool success = STAKING.claimRewards(valId);
    if (!success) revert ErrDelegateFailed();
}
```

Recommendation: Update `_claim` to use a `try-catch` block to handle potential reverts from the precompile gracefully. This allows capturing errors without propagating the precompile's revert, enabling custom handling. Make `_claim` return a `bool` success value (true on success, false on revert) and let call sites handle failures (e.g., skip or log without reverting the entire operation):

```
function _claim(uint64 valId) internal returns (bool success) {
    try STAKING.claimRewards(valId) returns (bool) {
        return true;
    } catch {
        return false;
    }
}
```

In call sites like `CoreVault._claimAndCompoundRewards()`, check the returned success and continue the loop on failure to avoid blocking the entire process.

Hydrogen Labs: Fixed in commit 7835c153.

Cantina Managed: Fix verified.

3.5.4 `_completeUserWithdrawal()`: violation of the "*No writes after a call*" best practice

Severity: Informational

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

Description: The `_completeUserWithdrawal()` function performs two external calls before executing a state-changing operation.

```
// _completeUserWithdrawal()
// ...
if (_totalSuccessfulWithdrawals > 0) {
    uint256 _fee = _chargeWithdrawalFee(_totalSuccessfulWithdrawals); // audit-note: first external call
    uint256 _remaining = _totalSuccessfulWithdrawals - _fee;

    // Transfer remaining funds to Magma contract which will forward to user
    (bool success,) = address($._magma).call{value: _remaining}(""); // audit-note: second external call
    if (!success) {
        revert ErrNativeTransferFailed();
    }

    _totalWithdrawn = _totalSuccessfulWithdrawals;
    _totalWithdrawnAfterFee = _remaining;

}
// Clear all withdrawal requests for this user after processing
delete $._userWithdrawalRequests[_user]; // audit-note: writing to storage
// ...
```

In a rare scenario, if one of the called contracts can trigger a reentrant call that creates a new redeem request for the same user, that request could be lost, since the user's previous withdrawal data is deleted later in the same execution.

Recommendation: For improved safety, code clarity, and easier future maintenance, consider applying the "no writes after calls" best practice. Specifically, move the `delete $._userWithdrawalRequests[_user]` statement so that it executes before the external calls within this function.

Hydrogen Labs: Fixed in commit bf7a705e.

Cantina Managed: Fixed by implementing the auditor's recommendation.

Hydrogen Labs: Fixed in commit bf7a705e.

Cantina Managed: Fix verified.

3.5.5 The `AdminInitiatedRebalance` event is missing parameters

Severity: Informational

Context: gVault.sol#L417

Description:

```
emit AdminInitiatedRebalance(_bps);
```

The event `AdminInitiatedRebalance` only emits the `_bps` value, which provides limited visibility into the actual state changes triggered by the rebalance operation.

Recommendation: Consider including additional context in the event, such as the total amount undelegated or other key parameters affected by the rebalance, to improve transparency and traceability.

Hydrogen Labs: Acknowledged. We decided to keep the event as it is to avoid additional complexity.

Cantina Managed: Acknowledged.

3.5.6 Fixes to inline comments

Severity: Informational

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

Description:

1. `CoreVault: L89` - replace `redistributeToValidators()` with `redelegateToValidators()`.

   ```
   // @dev Add a validator and trigger excess undelegation. Redistribution must be done manually via
   ↪  redistributeToValidators()
   ```

2. `VaultBase: L647` - Comment says the underflow handled is in case of slashing but in reality it is in case of additional rewards during withdrawal period.

   ```
   setPendingUndelegateByValidator(_valId, 0); // Prevent underflow if slashed
   ```

Hydrogen Labs: Fixed in commit bf7a705e.

Cantina Managed: Fixed by implementing the auditor's recommendation.

3.5.7 `Magma`: User experience improvements

Severity: Informational

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

Description:

1. The `depositMON()` function allows users to deposit MON directly into the `CoreVault`. However, there is no equivalent function for the `gVault`, limiting flexibility for users who prefer to interact solely through it.
2. If a user stakes with a specific validator in `gVault` and that validator is later removed, the user can redeem their shares back to MON/WMON.

   However, they are still required to pay the withdrawal fees, even if they intend to immediately re-stake those funds with another validator within the `gVault`.

Hydrogen Labs: Partially fixed in bf7a705e, the first point was fixed and the second point is acknowledged.

Cantina Managed: Fix verified.

3.5.8 Unused Referral Event Definition

Severity: Informational

Context: Magma.sol#L110

Description: The `Referral` event is defined in `Magma.sol` but is never emitted anywhere in the codebase, representing dead code that unnecessarily increases contract size.

Recommendation: Remove the unused event.

Hydrogen Labs: Fixed in commit 64b56686.

Cantina Managed: Fix verified.

3.5.9 `_allocateAdminWidAndUndelegate` Returns Default Value Instead of Allocated WID

Severity: Informational

Context: VaultBase.sol#L572

Description: The internal function `_allocateAdminWidAndUndelegate` in `src/VaultBase.sol` declares a named return variable `_wid` but does not assign a value to it, causing it to always return the default `uint8` value (0). It allocates and uses the hardcoded `ADMIN_WID` (255) for undelegation, but fails to return this value as documented ("@return _wid The admin withdrawal ID (always 255)").

Recommendation: Assign the hardcoded ADMIN_WID to the return variable:

```
// src/VaultBase.sol
function _allocateAdminWidAndUndelegate(uint64 _valId, uint256 _amount) internal returns (uint8 _wid) {
    _getVaultBaseStorage()._withdrawalIdBitmaps[_valId].allocateAdminWid();
    _wid = ADMIN_WID;
    _undelegate(_valId, _amount, _wid);
    return _wid;
}
```

Hydrogen Labs: Fixed in commit ec6a0c48.

Cantina Managed: Fix verified.

3.5.10 Validator Caps Bypassed in Rewards Compounding and Injection

Severity: Informational

Context: gVault.sol#L351

Description: the `delegate(address _user, uint64 _valId)` function enforces per-validator caps by checking if the new delegation amount would exceed `_maxCapFor(_valId)` and reverting with `ErrExceedsCap` if so. However, `injectRewards(uint64 _valId)` and `_claimAndCompoundRewards(uint64 _valId)` (called by `claimAndCompoundRewards()`) delegate funds (MEV rewards or compounded staking rewards) directly to a validator via `_delegate` without any cap checks. This allows validators to exceed their configured caps, potentially leading to over-concentration of stake.

It may be intentional in `claimAndCompoundRewards`, but it might not be in `injectRewards()`.

Recommendation: Add cap enforcement to these paths, similar to `delegate()`.

Hydrogen Labs: Acknowledged.

Cantina Managed: Acknowledged.

3.5.11 Rounding mismatch: code uses Floor while comment specifies Ceil

Severity: Informational

Context: gVault.sol#L267

Description: In `src/gVault.sol` within `delegate()`, the comment says to compute units using ceiling ("Calculate units = ceil(...)") to avoid precision erosion, but the implementation uses `Math.Rounding.Floor` when adding to `_scaledPrincipalUnits`. This contradicts the stated intent.

Recommendation: Fix comment or the code.

Hydrogen Labs: Fixed in commit bf7a705e.

Cantina Managed: Fixed by implementing the auditor's recommendation.

3.5.12 `Magma`: Use two-step admin transfer process

Severity: Informational

Context: Magma.sol#L502

Description: The `Magma` contract currently allows the admin role to be transferred in a single step. This means that if an incorrect address is mistakenly set as the new admin, control over the contract could be permanently lost.

Recommendation: Consider implementing a two-step admin transfer process where the new admin must explicitly accept the role before the change takes effect. This reduces the risk of accidental loss of administrative control.

Hydrogen Labs: Fixed in commit bf7a705e.

Cantina Managed: Fixed by implementing the auditor's recommendation.

3.5.13 Calls to `delegate()` and `undelegate()` will revert in case rewards were not claimed for a duration of `_maxRewardsClaimDelay`

Severity: Informational

Context: gVault.sol#L244

Description:

```
function _checkRewardsClaimDelay() internal view virtual {
    VaultBaseStorage storage $ = _getVaultBaseStorage();
    if (block.timestamp - $._lastRewardsClaimTimestamp > $._maxRewardsClaimDelay) {
        revert ErrRewardsClaimOverdue();
    }
}
```

`delegate()` and `undelegate()` in both `gVault` and `CoreVault` will revert in case rewards were not claimed for the validator for more than the maximum duration defined as `_maxRewardsClaimDelay`, `claimAndCompoundRewards()` will need to be called first to resolve this issue but this is a suboptimal user experience that should be improved.

Recommendation: Consider changing the logic to claim the rewards instead of reverting, make sure to call `_updateLastRewardsClaimTimestamp()`.

Hydrogen Labs: Acknowledged. We will periodically claim rewards and watch to avoid this happening.

Cantina Managed: Acknowledged.

3.5.14 `claimAndCompoundRewards()`: missing reentrancy guard

Severity: Informational

Context: Magma.sol#L539

Description: While no specific reentrancy attack vector was identified, maintaining consistent protection against reentrancy across all state-changing functions is important for system robustness and clarity. The function `claimAndCompoundRewards()` currently lacks a reentrancy guard, creating an inconsistency in the contract's security pattern.

Recommendation: Consider adding a reentrancy guard to `claimAndCompoundRewards()` both in `gVault` and `CoreVault` to ensure consistent protection and align with the overall contract security model.

Hydrogen Labs: Fixed in commit bf7a705e.

Cantina Managed: Fixed by implementing the auditor's recommendation.

3.5.15 Magma token holders can sybil and bypass the constraint of only one redeem request at a time

Severity: Informational

Context: Magma.sol#L377

Description: Magma token holders are designed to be limited to a single active redemption request at a time - meaning they should only be able to call `requestRedeem()` again after the previous redemption cycle (both `requestRedeem()` and `redeem()`) has been completed.

However, in practice this restriction can be bypassed by transferring Magma tokens to another address and initiating a new `requestRedeem()` from there. This effectively allows users to make multiple concurrent redemption requests, undermining the intended limitation.

Recommendation: There is no simple on-chain mitigation for this classic Sybil-style bypass. Awareness of this behavior and its potential economic implications is advised.

Hydrogen Labs: Acknowledged. One request at a time was designed ot simplify accounting and code. We are okay with a sybil bypassing.

Cantina Managed: Acknowledged.

3.5.16 Redundant check of `_exists`

Severity: Informational

Context: gVault.sol#L436, VaultBase.sol#L430

Description: both `CoreVault._completeAllPendingRedelegationWithdrawals()` and `VaultBase._completeValidatorRemovalWithdrawal()` check both `_exists` and `_withdrawalAmount > 0` while in reality it is redundant since `exists` means `_withdrawalAmount > 0`.

Recommendation: Consider removing `_exists`.

Hydrogen Labs: Fixed in commit 5a2ec2e5.

Cantina Managed: Fix verified.

3.5.17 `RebalanceInitiated` is an empty event

Severity: Informational

Context: CoreVault.sol#L398

Description: The `RebalanceInitiated` event currently provides no contextual information about the rebalance process, making it difficult to monitor rebalancing activity or perform analytics on-chain or off-chain.

Recommendation: Add relevant parameters to the event for improved transparency and tracking:

```
event RebalanceInitiated(
    uint256 totalUndelegated,
    uint256 numValidators,
    uint256 targetPerValidator,
    uint256 totalDelegated
);
```

Hydrogen Labs: Fixed in commit bf7a705e.

Cantina Managed: Fixed by implementing the auditor's recommendation.

3.5.18 Redundant complexity in the accounting of validator removal

Severity: Informational

Context: VaultBase.sol#L464

Description: The validator removal process consists of three stages:

1. `initiateValidatorRemoval()`.

2. `executeValidatorUndelegation()`.
3. `completeValidatorRemovalWithdrawal()`.

The main purpose of `initiateValidatorRemoval()` is to ensure that all pending deposits are transitioned from `deltaStake` and `nextDeltaStake` into the active `staked` balance. This is necessary because delegation in the Monad staking contract is asynchronous and requires some time to complete.

However, `_initiateValidatorRemoval()` currently also modifies `_pendingRedelegateByValidator` and `_totalPendingRedelegation`. At this stage, these values might not yet reflect the finalized state, which can result in inaccurate intermediate data. While this inaccuracy is later compensated for in `_executeValidatorUndelegation()`, the approach adds unnecessary complexity and increases the cognitive load for code reviewers and maintainers.

Recommendation: Simplify the validator removal logic by removing the modification of `_pendingRedelegateByValidator` and `_totalPendingRedelegation` from `_initiateValidatorRemoval()`.

Instead, perform these updates directly within `_executeValidatorUndelegation()`, where the state of validator redelegation is fully known and stable. This adjustment will improve readability, maintainability, and reduce the risk of future logic inconsistencies.

Hydrogen Labs: Acknowledged.

Cantina Managed: Acknowledged.

3.5.19 `VaultBase` lacks a `receive()` function

Severity: Informational

Context: VaultBase.sol#L717

Description: The `VaultBase` contract defines a function `_claimValidatorRewards()` that interacts with the Monad staking contract to claim MON (the network's native token) and transfer it back to the vault. This operation requires the vault contract to be capable of receiving native tokens.

In the current implementation, this requirement is implicitly satisfied because both derived contracts - `gVault` and `CoreVault` - define their own `receive()` functions. However, this design introduces a potential future vulnerability: if a new contract inherits from `VaultBase` but fails to implement a `receive()` function, the reward-claiming mechanism will break. As a result, claimed MON rewards would be unable to reach the vault, potentially leading to lost or stuck funds.

Recommendation: To make the system more robust and inheritance-safe, consider adding an empty `receive()` function directly in the `VaultBase` contract, ensuring that all derived contracts can safely receive MON without requiring explicit reimplementation.

Hydrogen Labs: Fixed in commit bf7a705e.

Cantina Managed: Fixed by implementing the auditor's recommendation.

3.5.20 `redelegateToValidators()` is missing event emission

Severity: Informational

Context: CoreVault.sol#L405

Description: The `redelegateToValidators()` function redistributes stake across validators during rebalancing but does not emit any event to record this action. This makes redelegations untraceable off-chain and reduces transparency for monitoring and auditing.

Recommendation: Emit an event after successful redelegation to improve observability and auditability.

```
event RedelegationCompleted(
    uint256 totalRedelegatedAmount,
    address[] fromValidators,
    address[] toValidators,
    uint256 timestamp
);
```

Hydrogen Labs: Acknowledged.

Cantina Managed: Acknowledged.