// Security Assessment 09.26.2025 - 10.02.2025

EVM Managed Vaults PR

Renzo Protocol

HALBORN

EVM Managed Vaults PR - Renzo Protocol

Prepared by: HALBORN

Last Updated 11/10/2025

Date of Engagement: September 26th, 2025 - October 2nd, 2025

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
14	1	1	1	6	5

TABLE OF CONTENTS

1. Summary

2. Introduction

Renzo engaged our security analysis team to perform a comprehensive security audit of the LiquidVaults smart contract ecosystem. The primary objective was to rigorously evaluate the security posture of the vault, strategy, and withdrawal queue contracts, identify vulnerabilities, assess protocol robustness, and provide actionable recommendations to enhance the safety and operational integrity of the protocol. Our review was strictly limited to the smart contracts and codebase provided, ensuring a focused and in-depth analysis of their security and economic mechanisms.

3. Assessment Summary

Our engagement with Renzo spanned a 4-day period, during which a dedicated security engineer with deep expertise in DeFi, ERC4626 vaults, and smart contract security conducted the assessment. The goals of this audit were to:

- Validate the correct and intended functionality of the vault, strategy, and withdrawal queue contracts.
- Identify and analyze potential security vulnerabilities, including economic attacks, access control issues, and edge-case failures.
- Provide clear, actionable recommendations to strengthen the protocol's security and efficiency.

4. Test Approach And Methodology

Our methodology combined manual and automated techniques to ensure comprehensive coverage and depth. The process included:

- In-depth architectural review of the LEZyVault, WithdrawQueue, and strategy contracts, with a focus on ERC4626 compliance, upgradeability, and integration patterns.
- Manual code review to uncover logic errors, access control lapses, and economic vulnerabilities, including the ERC4626 inflation attack vector.
- Automated static analysis and call graph mapping to identify reentrancy surfaces, DoS risks, and integration edge cases.
- Custom test case generation and scenario-based PoC development using Foundry, targeting both functional correctness and exploitability.
- Fuzzing and invariant testing to validate accounting integrity, asset conservation, and liveness properties across deposit, withdrawal, and strategy flows.

5. Key Findings And Recommendations

The assessment identified several areas of strength, including robust modularity, clear event emission, and adherence to upgradeable contract patterns. However, the review also surfaced critical and high-priority issues:

- **ERC4626 Inflation Attack Risk:** The vault and strategy contracts were susceptible to donation-based inflation attacks due to reliance on balance-based accounting. We recommend implementing internal accounting and “dead shares” initialization, and following OpenZeppelin’s best practices for ERC4626 vaults.
- **Access Control Gaps:** Certain functions (e.g., approve/transfer in strategies) lacked proper access restrictions, increasing the risk of unauthorized asset movement. We recommend enforcing strict role-based access control.
- **Event Emission and State Consistency:** Some events were emitted in misleading contexts or without sufficient state validation. We recommend refining event logic to ensure accurate protocol observability.
- **Withdrawal Queue Logic:** The queue lacked certain boundary checks and could be affected by asset rebalancing or edge-case timing. We recommend additional validation and invariant tests for withdrawal flows.
- **Preview Function Exposure:** Preview functions could leak sensitive information or be used for griefing. We recommend overriding or restricting these functions as appropriate.

By addressing these findings and implementing the recommended mitigations, Renzo can significantly enhance the security, reliability, and trustworthiness of its LiquidVaults platform.

6. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

6.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

6.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (C:N) Low (C:L) Medium (C:M) High (C:H) Critical (C:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

6.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

7. SCOPE

REPOSITORY

(a) Repository: LiquidVaults

(b) Assessed Commit ID: a0becb6

(c) Items in scope:

- contracts/LEZyVault/LEZyVault.sol
- contracts/Strategies/EulerSwap/EulerSwapEzEthDelegateStrategy.sol
- contracts/WithdrawQueue/WithdrawQueue.sol
- contracts/LEZyVaultFactory.sol
- contracts/Strategies/Aave/AaveV3DelegateStrategy.sol
- contracts/Permissions/RoleManager.sol
- contracts/Strategies/Morpho/MorphoDelegateStrategy.sol
- contracts/WithdrawQueue/WithdrawQueueStorage.sol
- contracts/Errors/Errors.sol
- contracts/LEZyVault/LEZyVaultStorage.sol
- contracts/LEZyVault/ILEZyVault.sol
- contracts/Strategies/BaseDelegateStrategy.sol
- contracts/WithdrawQueue/IWithdrawQueue.sol
- contracts/LEZyVaultFactoryStorage.sol
- contracts/Permissions/IRoleManager.sol
- contracts/Permissions/RoleManagerStorage.sol

REMEDIATION COMMIT ID:

- https://github.com/Renzo-Protocol/LiquidVaults/pull/10/commits/c156bea1a2d8d2cc7b61c5471695aef0b145456c
- https://github.com/Renzo-Protocol/LiquidVaults/pull/10/commits/61521ece666bd1a6c1ca47b821527e84a01f6778
- https://github.com/Renzo-Protocol/LiquidVaults/pull/10/commits/31af0f8c748b7ea23dbbd8df5bd98ea677b23763#diff-5d10057372f67214228eb7deb5a30b4c3d237deef6394311c4744fc4236c6d04R32
- https://github.com/Renzo-Protocol/LiquidVaults/pull/7/commits/eff6fa61880adb2194ea67880ad2c4dd71ae1375
- https://github.com/Renzo-Protocol/LiquidVaults/pull/10/commits/31af0f8c748b7ea23dbbd8df5bd98ea677b23763
- https://github.com/Renzo-Protocol/LiquidVaults/pull/10/commits/0691906c1bc4095b93cbdf48441a81f9394e6fa1
- https://github.com/Renzo-Protocol/LiquidVaults/pull/7/commits/6bd013c186688a722b93c4cee85c4f3ddd59fca1

- https://github.com/Renzo-Protocol/LiquidVaults/pull/10/commits/0833124c878749b69d2c837e2ca9ad4a2ef3b16d
- https://github.com/Renzo-Protocol/LiquidVaults/pull/9/commits/8fd5e0d91989df25c27c8275458e9cce2062ef9d

Out-of-Scope: New features/implementations after the remediation commit IDs.

8. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
1	1	1	6

INFORMATIONAL
5

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
UNRESTRICTED TOKEN TRANSFER AND APPROVAL IN STRATEGY	CRITICAL	SOLVED - 10/09/2025
INCONSISTENT CONVERSION LOGIC FOR RATE AND FEE CALCULATIONS	HIGH	SOLVED - 10/08/2025
INACCURATE QUEUE FILLING WITH REBALANCING ASSETS	MEDIUM	RISK ACCEPTED - 10/08/2025
ERC4626 INFLATION RISK VIA DIRECT TOKEN DONATION	LOW	SOLVED - 10/08/2025
INCONSISTENT PREVIEW FUNCTIONS FOR RESTRICTED ACTIONS	LOW	SOLVED - 10/08/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
OUTDATED TOTAL ASSET REPORTING FOR INTEGRATORS	LOW	RISK ACCEPTED - 10/09/2025
MISSING MAX WITHDRAW LIMIT CHECK IN WITHDRAW QUEUE	LOW	SOLVED - 10/08/2025
OVERFILLING WITHDRAW QUEUE DUE TO STATIC FILLAT CHECK	LOW	SOLVED - 10/08/2025
STRATEGY REMOVAL CAN BE BLOCKED BY EXTERNAL DEPOSITS	LOW	RISK ACCEPTED - 10/09/2025
ETH TRANSFER RESTRICTIONS CAN BE BYPASSED	INFORMATIONAL	ACKNOWLEDGED - 10/09/2025
INCONSISTENT ASSET DECIMALS IN STRATEGY REPORTING	INFORMATIONAL	SOLVED - 10/08/2025
REDUNDANT ASSIGNMENT IN INITIALIZATION	INFORMATIONAL	SOLVED - 10/08/2025
EMITTING EVENT FOR UNCHANGED DELEGATE STRATEGIES	INFORMATIONAL	ACKNOWLEDGED - 10/09/2025
DIRECT REDEMPTION COULD INTEGRATE WITH WITHDRAW QUEUE	INFORMATIONAL	ACKNOWLEDGED - 10/09/2025

9. FINDINGS & TECH DETAILS

9.1 UNRESTRICTED TOKEN TRANSFER AND APPROVAL IN STRATEGY

// CRITICAL

Description

In the `BaseDelegateStrategy` contract, the `approve` and `transfer` functions are externally accessible without any access control. This allows any address to approve arbitrary spenders or transfer tokens from the strategy contract to any recipient. As a result, malicious actors could drain or manipulate the strategy's token balances, leading to loss of funds and undermining the intended strategy logic.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:C/D:C/Y:C (10.0)

Recommendation

Restrict access to the `approve` and `transfer` functions in the `BaseDelegateStrategy` contract by implementing appropriate access control, such as only allowing calls from authorized contracts or designated roles. This will prevent unauthorized parties from moving or approving tokens and protect the strategy's assets. Strategies should have a way to recover funds or donations, create internal tracking of actual deposits and donations.

Remediation Comment

SOLVED: The **Renzo Protocol team** solved this finding by removing the BaseDelegateStrategy contract, resulting in the removal of unrestricted transfer and approval in strategy. Instead, approval is used in DelegateStrategies wherever required.

Remediation Hash

https://github.com/Renzo-Protocol/LiquidVaults/pull/10/commits/c156bea1a2d8d2cc7b61c5471695aef0b145456c

9.2 INCONSISTENT CONVERSION LOGIC FOR RATE AND FEE CALCULATIONS

// HIGH

Description

In the `LEZyVault` contract, the `_recordRate` function uses `convertToAssets` to compute the share-to-asset rate, while `_accrueFee` uses a custom `_convertToSharesWithTotals` function that mimics the `convertToShares` logic. This inconsistency can lead to incorrect rate calculations and fee accruals, as the conversion logic for shares and assets should be consistent and follow the ERC4626 standard. The current implementation may result in inaccurate performance fee calculations or rate tracking, especially when the vault's supply or asset balances change.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:H/D:N/Y:N (7.5)

Recommendation

Refactor the contract to introduce both `_convertToSharesWithTotals` and `_convertToAssetsWithTotals` functions, each following the correct ERC4626 logic for converting between shares and assets. Use `_convertToAssetsWithTotals` for rate calculations (as in `_recordRate`) and `_convertToSharesWithTotals` for determining the number of shares to mint for performance fees (as in `_accrueFee`). This will ensure consistent and accurate conversions throughout the contract and align with ERC4626 expectations.

Remediation Comment

SOLVED: The **Renzo Protocol team** solved this finding.

Remediation Hash

https://github.com/Renzo-Protocol/LiquidVaults/pull/10/commits/61521ece666bd1a6c1ca47b821527e84a01f6778

9.3 INACCURATE QUEUE FILLING WITH REBALANCING ASSETS

// MEDIUM

Description

If the vault's `asset()` is a rebalancing or interest-bearing token, the `WithdrawQueue` contract may not accurately account for yield or balance changes that occur while assets are held in the queue. The queue uses static values to track how much needs to be filled, but if the asset generates yield or is rebalanced, the actual value in the queue may increase independently. This can result in the vault overfilling the queue, providing more assets than necessary to fulfill withdrawal requests and reducing capital efficiency.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:M/D:N/Y:N (5.0)

Recommendation

Avoid using rebalancing or interest-bearing tokens as the vault's `asset()` unless the queue logic is updated to dynamically account for balance changes due to yield or rebalancing. If such assets must be used, implement logic in the `WithdrawQueue` to track real-time balances and accrued yield, ensuring that only the required amount is filled to satisfy withdrawal requests. Document this limitation for integrators and review the queue's accounting model before supporting rebalancing assets.

Remediation Comment

RISK ACCEPTED: The **Renzo Protocol team** accepted the risk of this finding as they do not intend on using rebasing tokens as underlying in the vault and have added a warning that rebasing tokens should not be used as underlying in the vault.

Remediation Hash

https://github.com/Renzo-Protocol/LiquidVaults/pull/10/commits/31af0f8c748b7ea23dbbd8df5bd98ea677b23763#diff-5d10057372f67214228eb7deb5a30b4c3d237deef6394311c4744fc4236c6d04R32

9.4 ERC4626 INFLATION RISK VIA DIRECT TOKEN DONATION

// LOW

Description

The `LEZyVault` contract, following the ERC4626 standard, is vulnerable to the classic inflation attack due to how it calculates the vault's total underlying assets. In the `_trackUnderlying` function, the contract sums `super.totalAssets()` and the `balanceOf` of the vault's asset held by the `WithdrawQueue` contract. This approach is problematic because anyone can donate tokens directly to the vault or the `WithdrawQueue` without going through the intended deposit flow.

When an attacker donates a large amount of the vault's asset directly to the `WithdrawQueue` or the vault, the next call to `trackUnderlying` (by an admin or through a management function) will inflate the `totalUnderlying` value. This artificially increases the exchange rate between shares and assets, causing new depositors to receive fewer or even zero shares for their deposits. Existing shareholders, especially those who deposited before the inflation, can then redeem their shares for a disproportionate share of the vault's assets, effectively "stealing" the donated funds.

This vulnerability is not limited to the vault and withdraw queue. If any strategy contract (such as `AaveV3DelegateStrategy`) calculates its `underlyingValue` based on the balance of an interest-bearing token, a direct donation to the strategy contract can similarly inflate the vault's total assets, enabling the same attack.

Proof of Concept

```
    function test_underlaying_inflation() public {

        address EXCHANGE_RATE_ADMIN = makeAddr("EXCHANGE_RATE_ADMIN");

        address C = makeAddr("C");
        address D = makeAddr("D");
        address E = makeAddr("E");
        address F = makeAddr("F");
        address G = makeAddr("G");
        address H = makeAddr("H");
        address I = makeAddr("I");
        address J = makeAddr("J");
        address K = makeAddr("K");
        address L = makeAddr("L");

        vm.startPrank(PROTOCOL_MULTISIG);
        address[] memory accounts = new address[](10);
        bool[] memory status = new bool[](10);
        accounts[0] = C; status[0] = true;
        accounts[1] = D; status[1] = true;
        accounts[2] = E; status[2] = true;
        accounts[3] = F; status[3] = true;
        accounts[4] = G; status[4] = true;
        accounts[5] = H; status[5] = true;
        accounts[6] = I; status[6] = true;
        accounts[7] = J; status[7] = true;
        accounts[8] = K; status[8] = true;
        accounts[9] = L; status[9] = true;

        lezyVault.updateWhitelist(accounts, status);
        vm.stopPrank();

        vm.startPrank(TIMELOCK_CONTROLLER);
        roleManager.grantRole(roleManager.EXCHANGE_RATE_ADMIN(), EXCHANGE_RATE_ADMIN);
        vm.stopPrank();
```

```
        ////////////////////////////////////

        assertEq(lezyVault.totalAssets(), 0);
        assertEq(IERC20(lezyVault.asset()).balanceOf(address(withdrawQueue)), 0);

        uint256 depositAmount = 1;
        helper_deposit(ALICE, depositAmount);

        assertEq(lezyVault.totalSupply(), depositAmount);
        assertEq(lezyVault.balanceOf(ALICE), 1);

        // Perform a donation of 10_000 USDC to the vault
        deal(VAULT_ASSET, address(withdrawQueue), 10_000 * (10 ** underlyingDecimals));

        // The donation does inflate the totalUnderlying with the donation amount
        // similar to an standard ERC4626 vault inflation attack when the totalUnderlaying
        // gets updated.
        // However, in this case since we are using internal tracking this is the only function that
        // the totalUnderlying variable.
        // This requires the EXCHANGE_RATE_ADMIN role. The track underlying function
        // can also be called though manage function by the REBALANCE_ADMIN role
        vm.startPrank(EXCHANGE_RATE_ADMIN);
        lezyVault.trackUnderlying();
        vm.stopPrank();

        // Bob now deposits 1000 USDC
        depositAmount = 5_000 * (10 ** underlyingDecimals);
        helper_deposit(BOB, depositAmount);

        // Bob should get the deposited amount, but due to the inflation of the internal accounting
        // he gets 0
        // assertEq(lezyVault.balanceOf(BOB), depositAmount);
        assertEq(lezyVault.balanceOf(BOB), 0);

        helper_deposit(C, depositAmount);
        helper_deposit(D, depositAmount);
        helper_deposit(E, depositAmount);
        helper_deposit(F, depositAmount);
        helper_deposit(G, depositAmount);
        helper_deposit(H, depositAmount);
        helper_deposit(I, depositAmount);
        helper_deposit(J, depositAmount);
        helper_deposit(K, depositAmount);
        helper_deposit(L, depositAmount);

        // Due to donation and deposits not matching or surpasing it, supply will still be 1
        assertEq(lezyVault.totalSupply(), 1);
        // The Alice deposit + 10_000 donation + 11 * 1_000 deposits (for other users including BOB)
        assertEq(lezyVault.totalAssets(), 1 + 10_000 * (10 ** underlyingDecimals) + 11 * 5_000 * (10

        // All users except Alice have 0 shares
        assertEq(lezyVault.balanceOf(BOB), 0);
        assertEq(lezyVault.balanceOf(C), 0);
        // ....

        // They will get 0 assets when trying to withdraw
        assertEq(lezyVault.convertToAssets(lezyVault.balanceOf(BOB)), 0);
        assertEq(lezyVault.convertToAssets(lezyVault.balanceOf(C)), 0);
        // ....

        // Alice has all the shares, those she can withdraw all the assets
        assertEq(lezyVault.balanceOf(ALICE), 1);

        vm.startPrank(ALICE);
        lezyVault.approve(address(withdrawQueue), 1);

        withdrawQueue.withdraw(1);
        vm.stopPrank();


        uint256 withdrawIndex = withdrawQueue.totalUserWithdrawRequests(ALICE) - 1;
        (
            uint256 withdrawRequestID,
            uint256 amountToRedeem,
            uint256 sharesLocked,
            uint256 createdAt,
            uint256 fillAt
```

```
        ) = withdrawQueue.withdrawRequests(ALICE, withdrawIndex);

        // Log the shares locked and amount to redeem
        console.log("sharesLocked:", sharesLocked);
        console.log("amountToRedeem:", amountToRedeem);

    }
```

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:C/D:C/Y:C (3.0)

Recommendation

To mitigate this risk in the `LEZyVault` contract:

- Mint a minimum number of "dead shares" (virtual shares) during initialization, as recommended by OpenZeppelin, to absorb the impact of small donations and make the attack economically unfeasible.
- Consider overriding the decimals offset to increase the number of virtual shares, but be aware this changes the vault token's decimals and may not be desirable for all use cases.
- Monitor and restrict when `manage` or `trackUnderlying` can be called, ensuring there is sufficient liquidity and no one is attempting to deposit a minimal amount of shares immediately before these functions are executed.
- Require a minimum number of shares (e.g., `1e9`) to exist in the vault, as suggested by OpenZeppelin, to further reduce the impact of inflation attacks.
- For all strategies, ensure that `underlyingValue` is not solely based on token balances that can be externally inflated. Use internal accounting or restrict token transfers to the strategy contract to prevent unauthorized donations.

Document this risk for integrators and administrators, and review all strategy implementations to ensure they are not susceptible to similar donation-based inflation attacks.

Remediation Comment

SOLVED: The **Renzo Protocol team** solved this finding. The previously shared remediation on **https://github.com/Renzo-Protocol/LiquidVaults/pull/7/commits/d18edccfdb007573eaaa43fae087238a209c5f31** did not solve the problem. The issue was finally solved by having a balance offset, which made the POC non-profitable anymore, overriding decimal offset to increase the number of virtual shares. The vault shares will be in 18 decimal precision.

Remediation Hash

https://github.com/Renzo-Protocol/LiquidVaults/pull/7/commits/eff6fa61880adb2194ea67880ad2c4dd71ae1375

9.5 INCONSISTENT PREVIEW FUNCTIONS FOR RESTRICTED ACTIONS

// LOW

Description

In the `LEZyVault` contract, the `mint`, `redeem`, and `withdraw` functions are intentionally disabled and always revert, requiring users to interact through the withdraw queue. However, the inherited ERC4626 preview functions such as `previewWithdraw`, `previewRedeem`, and `previewMint` remain accessible and return values as if these actions were available. This inconsistency can mislead integrators and users, as these functions suggest that direct withdrawals, redemptions, or mints are possible when they are not.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (2.5)

Recommendation

Override the `previewWithdraw`, `previewRedeem`, and `previewMint` functions in the `LEZyVault` contract to revert or return zero, making it clear that these actions are not supported directly. This will align the contract's interface with its intended usage and prevent confusion for users and integrators.

Remediation Comment

SOLVED: The **Renzo Protocol team** solved this finding. previewMint and previewWithdraw will revert as suggested. Also, overriding previewRedeem as it is getting used by the WithdrawQueue to determine the amount of assets for the given number of shares.

Remediation Hash

https://github.com/Renzo-Protocol/LiquidVaults/pull/10/commits/31af0f8c748b7ea23dbbd8df5bd98ea677b23763

9.6 OUTDATED TOTAL ASSET REPORTING FOR INTEGRATORS

// LOW

Description

In the `LEZyVault` contract, the `totalAssets` function is overridden to return the `totalUnderlying` value, which is only updated when `manage` or `trackUnderlying` is called. This means that the reported total assets may be outdated between updates, especially after deposits, withdrawals, or strategy operations. Integrators relying on `totalAssets` for up-to-date vault state may receive stale information, as the actual asset balance is calculated using `super.totalAssets()` plus balances in the withdraw queue and strategies, as well as accrued performance fees. This can affect integrations that depend on real-time asset values for accounting, risk management, or user interfaces.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (2.5)

Recommendation

Document the behavior of the `totalAssets` function in the `LEZyVault` contract and ensure that integrators are aware that the value may be outdated between explicit updates. If real-time accuracy is required, consider updating `totalUnderlying` more frequently or providing a separate view function that computes the current total assets on demand by aggregating all relevant balances and accrued fees.

Remediation Comment

RISK ACCEPTED: The **Renzo Protocol team** accepted the risk of this finding.

9.7 MISSING MAX WITHDRAW LIMIT CHECK IN WITHDRAW QUEUE

// LOW

Description

In the `WithdrawQueue` contract, the `withdraw` function allows users to submit withdrawal requests by transferring shares to the contract. However, it does not enforce any `maxWithdraw` or `maxRedeem` limit checks, which are standard in ERC4626 implementations to prevent users from withdrawing more than their available balance or the vault's capacity. This omission could allow users to initiate withdrawal requests for more shares than they are entitled to, potentially leading to failed transactions or unexpected contract states.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (2.5)

Recommendation

Add a check in the `withdraw` function of the `WithdrawQueue` contract to ensure that the requested withdrawal amount does not exceed the user's `maxRedeem` or `maxWithdraw` limit, depending on the intended logic. This will align the withdrawal process with ERC4626 standards and prevent users from over-requesting withdrawals beyond their available balance.

Remediation Comment

SOLVED: The **Renzo Protocol team** solved this finding by adding a MaxRedeem check in withdrawQueue as recommended.

Remediation Hash

https://github.com/Renzo-Protocol/LiquidVaults/pull/10/commits/0691906c1bc4095b93cbdf48441a81f9394e6fa1

9.8 OVERFILLING WITHDRAW QUEUE DUE TO STATIC FILLAT CHECK

// LOW

Description

In the `WithdrawQueue` contract, the `claim` function recalculates the redeemable asset amount for a withdrawal request based on the current share-to-asset rate, which is correct. However, the function checks if the queue is sufficiently filled using the original `fillAt` value from the stored request, which may be higher than the actual amount needed if the asset value per share has decreased. This can result in the queue being overfilled with more assets than necessary to fulfill withdrawal requests, leading to inefficient asset management and potential liquidity issues.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (2.5)

Recommendation

Update the `claim` function in the `WithdrawQueue` contract to recalculate the required `fillAt` value based on the final `claimAmountToRedeem`. After determining the actual redeemable amount, adjust the `fillAt` check to reflect the true asset requirement for the request. This ensures the queue is only filled with the necessary amount of assets, improving efficiency and preventing unnecessary overfilling.

Remediation Comment

Client Note:
Added the fix which fills the withdrawQueue by the delta amount i.e. (withdrawAmountToRedeem - claimAmountToRedeem) when claimAmountToRedeem is less than withdrawAmountToRedeem. It allows the remaining delta to fill the queueDeficit for pending withdrawRequests.

Remediation Hash

https://github.com/Renzo-Protocol/LiquidVaults/pull/7/commits/6bd013c186688a722b93c4cee85c4f3ddd59fca1

9.9 STRATEGY REMOVAL CAN BE BLOCKED BY EXTERNAL DEPOSITS

// LOW

Description

In the `LEZyVault` contract, the `removeDelegateStrategies` function checks that a strategy's `underlyingValue` is zero before allowing its removal. However, this value is typically based on the on-chain balance of interest-bearing tokens (such as aTokens or protocol-specific assets) held by the strategy. An attacker or any external party can frontrun the removal by depositing tokens directly into the strategy contract, causing `underlyingValue` to be non-zero and blocking the removal. This issue also affects strategies like `MorphoDelegateStrategy` and `EulerSwapEzEthDelegateStrategy`, where the balance is determined by the protocol account's balance, making them vulnerable to the same manipulation.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:L/I:N/D:N/Y:N (2.5)

Recommendation

Implement internal accounting within each strategy to track only assets deposited through the intended vault or protocol functions, rather than relying solely on token balances. This approach prevents external actors from interfering with the removal process and ensures that only authorized deposits and withdrawals affect the strategy's reported value. Review and update all affected strategies to use internal tracking for `underlyingValue` calculations. Another option is to have a function to enforce the removal without verifying the `underlyingValue` amounts but monitoring the contract values before triggering the remove function.

Remediation Comment

RISK ACCEPTED: The **Renzo Protocol team** accepted the risk of this finding.

9.10 ETH TRANSFER RESTRICTIONS CAN BE BYPASSED

// INFORMATIONAL

Description

In the `LEZyVault` contract, the `receive` function restricts ETH acceptance to whitelisted senders by checking the `whitelistedEthSender` mapping. However, this restriction can be bypassed if ETH is sent via the `selfdestruct` opcode from another contract or through certain L1 reward distribution mechanisms, which do not trigger the `receive` function and directly credit the contract's balance. As a result, ETH may be received from non-whitelisted sources, potentially leading to unexpected contract balances or operational inconsistencies.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:M/D:N/Y:N (1.0)

Recommendation

Acknowledge that ETH can be forcibly sent to the contract outside of the `receive` function, and ensure that all contract logic and accounting are robust against unexpected ETH deposits. Consider documenting this behavior for users and avoid relying on the contract's ETH balance for critical logic or security assumptions.

Remediation Comment

ACKNOWLEDGED: The **Renzo Protocol team** acknowledged this finding and made sure that the scenario was well documented. In case the receive is bypassed through selfDestruct the ETH received will be considered as the donation to the vault depending on the delegateStrategy being used for the vault.

9.11 INCONSISTENT ASSET DECIMALS IN STRATEGY REPORTING

// INFORMATIONAL

Description

In the `LEZyVault` contract, the `_trackUnderlying` function aggregates the vault's asset balance with the values returned by the `underlyingValue` function from each strategy. However, strategies do not enforce or report the decimals of the assets they manage, which can lead to integration issues if a strategy returns a value with a different decimal factor than the vault's main asset. Adding values with mismatched decimals can result in incorrect total asset calculations, affecting accounting, fee accrual, and user balances.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:M/D:N/Y:N (1.0)

Recommendation

Ensure that all strategies used with the vault return values from `underlyingValue` that are denominated in the same decimal format as the vault's main asset. Carefully review and test each strategy before deployment to confirm decimal consistency, and document this requirement for future integrations. Consider adding validation or interface requirements to enforce decimal alignment between the vault and its strategies.

Remediation Comment

SOLVED: The **Renzo Protocol team** solved this finding by adding the underlying asset check in DelegateStrategy::underlyingValue(address asset) to ensure that the strategy reports the underlyingValue in correct underlying decimals.

Remediation Hash

https://github.com/Renzo-Protocol/LiquidVaults/pull/10/commits/0833124c878749b69d2c837e2ca9ad4a2ef3b16d

9.12 REDUNDANT ASSIGNMENT IN INITIALIZATION

// INFORMATIONAL

Description

In the `LEZyVault` contract, the `initialize` function assigns values to `performanceFeeBps` and `feeRecipient` twice consecutively. This redundancy does not impact the contract's logic or security but introduces unnecessary code, which can reduce code clarity and maintainability.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

Remove the duplicate assignments of `performanceFeeBps = _feeBps;` and `feeRecipient = _feeRecipient;` in the `initialize` function of the `LEZyVault` contract to improve code readability and prevent confusion.

Remediation Comment

SOLVED: The **Renzo Protocol team** solved this finding by removing the redundant assignment in initialize.

Remediation Hash

https://github.com/Renzo-Protocol/LiquidVaults/pull/9/commits/8fd5e0d91989df25c27c8275458e9cce2062ef9d

9.13 EMITTING EVENT FOR UNCHANGED DELEGATE STRATEGIES

// INFORMATIONAL

Description

In the `LEZyVault` contract, the `addDelegateStrategies` function emits the `DelegateStrategyAdded` event for all input addresses, regardless of whether each address was actually added to the set. Since `EnumerableSet.add` returns false if the address is already present, emitting the event for all input addresses can be misleading and may not accurately reflect changes to the allowed delegate strategies.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

Update the `addDelegateStrategies` function in the `LEZyVault` contract to emit the `DelegateStrategyAdded` event only for addresses that were actually added to the set. Collect the addresses for which `add` returns true and emit the event with this filtered list to ensure accurate event logs.

Remediation Comment

ACKNOWLEDGED: The **Renzo Protocol team** acknowledged this finding.

9.14 DIRECT REDEMPTION COULD INTEGRATE WITH WITHDRAW QUEUE

// INFORMATIONAL

Description

In the `LEZyVault` contract, the `redeem` function is currently disabled and always reverts, requiring users to interact with the withdraw queue for redemptions. However, it is possible to design the `redeem` function to internally call the withdraw queue contract, allowing users to redeem shares directly through the vault. This approach would streamline the user experience by enabling a standard ERC4626 interface while still enforcing the intended withdrawal flow.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

Consider implementing the `redeem` function in the `LEZyVault` contract to interact with the withdraw queue contract, so that direct redemptions are routed through the proper withdrawal process. This would maintain compatibility with ERC4626 expectations and improve usability, while still enforcing the protocol's withdrawal logic.

Remediation Comment

ACKNOWLEDGED: The **Renzo Protocol team** acknowledged this finding.
