EzRVaults PR15 - Contracts PR170 PR171

Renzo Protocol

HALBORN

EzRVaults PR15 - Contracts PR170 PR171 - Renzo Protocol

Prepared by: HALBORN

Last Updated 06/05/2025

Date of Engagement: June 2nd, 2025 - June 11th, 2025

Summary

0% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
2	0	0	0	0	2

TABLE OF CONTENTS

1. Summary

2. Introduction

Renzo engaged our security analysis team to conduct a comprehensive security assessment of their smart contract ecosystem. The primary objective was to thoroughly evaluate the security architecture of the smart contracts to identify vulnerabilities, assess existing security measures, and provide actionable recommendations to enhance both the security and operational effectiveness of their smart contract framework. Our assessment was strictly limited to the provided smart contracts, ensuring a focused and exhaustive analysis of their security features.

3. Assessment Summary

Our engagement with Renzo spanned a four-day period, during which we assigned a full-time security engineer with extensive experience in blockchain security, advanced penetration testing skills, and deep knowledge of various blockchain protocols. The objectives of this assessment were to:

- Verify the correct functionality of the smart contract operations.

- Identify potential security vulnerabilities within the smart contracts.

- Provide recommendations to improve the security and efficiency of the smart contracts.

4. Test Approach And Methodology

Our testing strategy combined manual and automated techniques to ensure a comprehensive evaluation. Manual testing was essential for detecting logical and implementation flaws, while automated testing provided broad code coverage and rapid identification of common vulnerabilities. The testing process included:

- A detailed review of the smart contracts' architecture and intended functionality.

- Comprehensive manual code reviews and walkthroughs.

- Functional and connectivity analysis using tools such as Solgraph.

- Customized script-based manual testing and testnet deployment using Foundry.

This executive summary highlights the key findings and recommendations from our security assessment of the Renzo smart contract ecosystem. By addressing the identified issues and implementing the recommended improvements, Renzo can significantly enhance the security, reliability, and trustworthiness of its smart contract platform.

5. 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.

5.1 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
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:

$$E = \prod m_e$$

5.2 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 (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
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:

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

5.3 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

6. SCOPE

FILES AND REPOSITORY

(a) Repository: Contracts
(b) Assessed Commit ID: https://github.com/Renzo-Protocol/Contracts/pull/170
(c) Items in scope:

Out-of-Scope:

FILES AND REPOSITORY

(a) Repository: Contracts
(b) Assessed Commit ID: https://github.com/Renzo-Protocol/Contracts/pull/171
(c) Items in scope:

Out-of-Scope:

FILES AND REPOSITORY

(a) Repository: EzRVaults
(b) Assessed Commit ID: https://github.com/Renzo-Protocol/EzRVaults/pull/15
(c) Items in scope:

Out-of-Scope:

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

7. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - INCONSISTENT ADDRESS USAGE BETWEEN GAS RECORDING AND REFUND	INFORMATIONAL	PENDING

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-02 - INCONSISTENT STATE UPDATE AFTER EXTERNAL CALL MAY CAUSE LOGIC MISMATCH	INFORMATIONAL	PENDING

8. FINDINGS & TECH DETAILS

8.1 (HAL-01) INCONSISTENT ADDRESS USAGE BETWEEN GAS RECORDING AND REFUND

// INFORMATIONAL

Description

In the `OperatorDelegator` contract, the `_recordGas` function records gas usage under `msg.sender` (or `gasRefundAddress` if set), whereas `_refundGas` issues the refund to `tx.origin`. This discrepancy causes a mismatch when operations are executed via an intermediary, such as a contract wallet or multisig: the recorded gas amount is attributed to the intermediary's address, but the refund is sent to the original externally owned account (EOA). Consequently, `adminGasSpentInWei[tx.origin]` may be zero, causing the refund to fail to reflect the actual recorded usage.
While the current design assumes refunds will only be claimed by EOAs interacting through EigenLayer contracts, this approach introduces fragility if the admin is a contract or if future upgrades alter calling patterns. Furthermore, `_refundGas` lacks a fallback branch for cases where `gasRefundAddress != address(0)`, resulting in gas refunds being misdirected or inaccessible.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:M/D:M/Y:N (1.3)

Recommendation

To ensure consistent and reliable refund behavior, the same address key should be used in both `_recordGas` and `_refundGas`. Aligning both functions to use `tx.origin`, as assumed by the refund logic, would prevent mismatches and ensure EOAs receive their expected refunds. Alternatively, if `gasRefundAddress` is intended to take precedence, `_refundGas` should use it instead of `tx.origin` when set.

8.2 (HAL-02) INCONSISTENT STATE UPDATE AFTER EXTERNAL CALL MAY CAUSE LOGIC MISMATCH

// INFORMATIONAL

Description

In the `WithdrawQueue` contract, the `completeStETHRebalance` function makes an external call to `stETHWithdrawalQueue.claimWithdrawals` before updating the `stETHPendingWithdrawAmount` and `stETHPendingWithdrawAmountById` state variables. This claim call transfers ETH back to the contract, triggering the `receive` function, which subsequently forwards ETH to the deposit queue.
This sequence introduces a subtle but significant issue: the contract processes ETH before finalizing its internal accounting for the claim. Although this behavior is not directly controllable by users, the contract responds to the side effect of an external call (`receive`) before completing its intended logic. If additional logic is later added to the `receive` function or the deposit queue path that relies on `stETHPendingWithdrawAmount`, it may operate on outdated or inconsistent state data.

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 `stETHPendingWithdrawAmount` and `stETHPendingWithdrawAmountById[_requestId]` **before** invoking `claimWithdrawals`. This ensures that the internal state remains consistent prior to any external interaction, even if the external call indirectly reenters the contract or triggers related execution paths.

---

introduced by code modifications.