// Private Smart Contract Security Assessment 09.12.2024 - 09.16.2024

ezRVaults

Renzo Protocol

HALBORN

ezRVaults - Renzo Protocol

Prepared by: HALBORN

Last Updated 09/19/2024

Date of Engagement by: September 12th, 2024 - September 16th, 2024

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
13	2	0	0	0	11

TABLE OF CONTENTS

1. Introduction

Renzo engaged our security analysis team to conduct a comprehensive security audit of their smart contract ecosystem. The primary aim was to meticulously assess the security architecture of the smart contracts to pinpoint vulnerabilities, evaluate existing security protocols, and offer actionable insights to bolster security and operational efficacy of their smart contract framework. Our assessment was strictly confined to the smart contracts provided, ensuring a focused and exhaustive analysis of their security features.

2. Assessment Summary

Our engagement with Renzo spanned a 4 days period, during which we dedicated one full-time security engineer equipped with extensive experience in blockchain security, advanced penetration testing capabilities, and profound knowledge of various blockchain protocols. The objectives of this assessment were to:

- Verify the correct functionality of smart contract operations.

- Identify potential security vulnerabilities within the smart contracts.

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

In summary, Halborn identified some security issues that were mosly addressed by the `Renzo team`.

3. Test Approach And Methodology

Our testing strategy employed a blend of manual and automated techniques to ensure a thorough evaluation. While manual testing was pivotal for uncovering logical and implementation flaws, automated testing offered broad code coverage and rapid identification of common vulnerabilities. The testing process included:

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

- Comprehensive manual code reviews and walkthroughs.

- Functional and connectivity analysis utilizing tools like Solgraph.

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

This executive summary encapsulates the pivotal findings and recommendations from our security assessment of Renzo smart contract ecosystem. By addressing the identified issues and implementing the recommended fixes, Renzo can significantly boost the security, reliability, and trustworthiness of its smart contract platform.

4. 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.

4.1 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:

EXPLOITABILIY 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

EXPLOITABILIY 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:

$$E = \prod m_e$$

4.2 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

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:

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

4.3 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

5. SCOPE

FILES AND REPOSITORY

(a) Repository: EzRVaults

(b) Assessed Commit ID: 58ae612

(c) Items in scope:

- contracts/EzRVaultsFactory.sol
- contracts/EzRVaultsFactoryStorage.sol
- contracts/EzRVault/EzRVault.sol
- contracts/EzRVault/EzRvaultStorage.sol
- contracts/Errors/Errors.sol

Out-of-Scope:

REMEDIATION COMMIT ID:

- https://github.com/Renzo-Protocol/EzRVaults/pull/3

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
2	0	0	0	11

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
VAULT DOS VULNERABILITY DUE TO PERMANENT DELEGATION LOCK	CRITICAL	SOLVED - 09/14/2024
INCORRECT USE OF UNDERLYING DECIMALS	CRITICAL	SOLVED - 09/14/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
PAUSING FUNCTIONALITY LACKS RESTRICTION ON UNPAUSING	INFORMATIONAL	SOLVED - 09/14/2024
INCORRECT HANDLING LEADING TO TRANSFER TO ZERO ADDRESS	INFORMATIONAL	ACKNOWLEDGED
MISSING VALIDATION FOR ZERO-LENGTH NAME AND SYMBOL	INFORMATIONAL	SOLVED - 09/14/2024
INCORRECT BALANCE HANDLING	INFORMATIONAL	ACKNOWLEDGED
VAULT ID DUPLICATION RISK IN CREATEVAULT	INFORMATIONAL	ACKNOWLEDGED
MISSING CALL IN INITIALIZER	INFORMATIONAL	SOLVED - 09/14/2024
FEE VALIDATION LOGIC LEADING TO REVERSION IN INITIALIZE	INFORMATIONAL	ACKNOWLEDGED
REDUNDANT UNDERLYING PARAMETER IN INITIALIZE	INFORMATIONAL	ACKNOWLEDGED
INCORRECT ASSUMPTION ON TOKEN BALANCE DELEGATION IN DEPOSIT	INFORMATIONAL	ACKNOWLEDGED
MISSING PARAMETER IN COMPLETE EMERGENCY TRACKED WITHDRAWAL`	INFORMATIONAL	ACKNOWLEDGED

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
UNNECESSARY COMPLEXITY IN WITHDRAWSTARTED EVENT	INFORMATIONAL	ACKNOWLEDGED

7. FINDINGS & TECH DETAILS

7.1 VAULT DOS VULNERABILITY DUE TO PERMANENT DELEGATION LOCK

// CRITICAL

Description

After undelegation via the `DelegationManager`, the vault becomes completely non-functional, as all user actions like `deposit`, `withdraw`, or `claim` will revert with a `VaultNotDelegated` error. Additionally, the vault owner cannot reset the delegation address because the `setDelegateAddress` function checks whether `delegateAddress` is already set, leading to a `DelegateAddressAlreadySet` error. This creates a permanent denial-of-service (DoS) for the vault, effectively locking users out from interacting with it.
The root of the issue is the logic preventing delegation reset and over-restrictive delegation checks, which make recovery impossible once undelegation occurs. The statement that "this can only be set once" is also incorrect and should be removed, as delegation needs to be restorable after undelegation.

BVSS

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

Recommendation

To resolve the issue: 1. Modify the `setDelegateAddress` function to allow the delegation address to be reset if `delegationManager.delegatedTo(address(this))` returns `address(0)` (indicating undelegation). 2. Simplify the `onlyWhenDelegated` modifier to revert only if `delegationManager.delegatedTo(address(this)) == address(0)`, removing the reliance on `delegateAddress`. 3. Remove the `delegateAddress` state variable from the contract, as it is redundant, and adjust related checks accordingly. 4. Update the contract comment to remove the statement "THIS CAN ONLY BE SET ONCE", allowing redelegation after undelegation.
This ensures the vault remains functional even after an undelegation event, preventing permanent DoS.

Remediation

SOLVED: The `delegateAddress` variable was removed and the `delegatedTo` mapping from the `delegationManager` directly used for all the checks as recommended.

Remediation Hash

https://github.com/Renzo-Protocol/EzRVaults/pull/3

7.2 INCORRECT USE OF UNDERLYING DECIMALS

// CRITICAL

Description

The `getRate`, `deposit`, and `withdraw` functions in the `EzRVault` contract incorrectly use `underlyingDecimals` by multiplying values directly by `underlyingDecimals` instead of `10 ** underlyingDecimals`. This miscalculation leads to an incorrect rate factor, resulting in an initial rate of `18` instead of `1e18` (`10 ** 18`), which is the expected precision for tokens with 18 decimal places. This can lead to significant confusion, truncation errors, and precision loss when calculating token rates, minting LP tokens, or handling withdrawals.

BVSS

AO:A/AC:L/AX:L/C:N/I:H/A:N/D:N/Y:N/R:N/S:C (9.4)

Recommendation

Update the `getRate`, `deposit`, and `withdraw` functions to use `10 ** underlyingDecimals` instead of multiplying by `underlyingDecimals`. This ensures proper scaling of values and avoids precision loss when interacting with tokens of various decimal places, maintaining the correct rate and consistent token handling across all operations.

Remediation

SOLVED: An scale factor function was introduced that calculates the intended conversion rate.

Remediation Hash

https://github.com/Renzo-Protocol/EzRVaults/pull/3

7.3 PAUSING FUNCTIONALITY LACKS RESTRICTION ON UNPAUSING

// INFORMATIONAL

Description

The current `setPause` function in `EzRVault` allows both the pauser and the owner to toggle the pause state of the protocol, meaning either can pause or unpause the vault. This could create a situation where an unauthorized or less trusted party can unpause the protocol, increasing the risk of misuse. Ideally, only the owner should have the ability to unpause the protocol to maintain better control over the system's security state.

BVSS

AO:S/AC:L/AX:L/C:N/I:H/A:N/D:N/Y:N/R:N/S:C (1.9)

Recommendation

Modify the `setPause` function so that while the pauser can pause the protocol, only the owner can unpause it. This ensures stricter control over resuming operations after a pause, reducing the risk of unauthorized protocol reactivation.

Remediation

SOLVED: The contract is now using the owner for the unpausing functionality.

Remediation Hash

https://github.com/Renzo-Protocol/EzRVaults/pull/3

7.4 INCORRECT HANDLING LEADING TO TRANSFER TO ZERO ADDRESS

// INFORMATIONAL

Description

The `vaultRewardsDestination` address is set during initialization and cannot be reconfigured afterward. If `vaultRewardsDestination` is mistakenly set to zero, this causes multiple issues: - The `processRewards` function will not function properly, as it requires a valid `vaultRewardsDestination` to forward non-underlying rewards. - The `sweepERC20` function will transfer any non-underlying tokens to an invalid or zero address via the `_processRewards` function, potentially causing token loss.
Although the constructor checks that `_rewardsDestination` is not zero, this check becomes irrelevant since the `vaultRewardsDestination` cannot be updated after initialization. If zero is desired or mistakenly set, the contract becomes vulnerable to these issues.

BVSS

AO:S/AC:L/AX:L/C:N/I:H/A:N/D:N/Y:N/R:N/S:C (1.9)

Recommendation

Allow the `vaultRewardsDestination` to be updated after initialization if necessary. Additionally, prevent the execution of the `sweepERC20` function for non-underlying tokens when `vaultRewardsDestination` is not set to a valid address. This will ensure safe handling of rewards and prevent accidental transfers to the zero address.

Remediation

ACKNOWLEDGED: The **Renzo team** acknowledged this issue.

7.5 MISSING VALIDATION FOR ZERO-LENGTH NAME AND SYMBOL

// INFORMATIONAL

Description

In the `initialize` function of the `EzRVault` contract, the `name` and `symbol` parameters are not validated for zero-length strings. This could lead to vaults being created with empty names or symbols, resulting in user confusion and potential difficulties in identifying or interacting with the vaults.

BVSS

AO:S/AC:L/AX:L/C:N/I:H/A:N/D:N/Y:N/R:N/S:U (1.5)

Recommendation

Add a check to ensure that both `name` and `symbol` parameters are not zero-length strings during the vault initialization process. This will prevent the creation of vaults with empty or invalid identifiers.

Remediation

SOLVED: The length is now validated.

Remediation Hash

https://github.com/Renzo-Protocol/EzRVaults/pull/3

7.6 INCORRECT BALANCE HANDLING

// INFORMATIONAL

Description

In the `completeEmergencyTrackedWithdrawal` function, the current implementation re-deposits the entire balance of the underlying token held by the vault, using `balanceOf(address(this))` for the re-deposit operation. This approach could unintentionally include any externally sent funds to the vault, which were not part of the emergency withdrawal. Such deposits would either be misallocated or erroneously considered part of the emergency re-deposit, leading to confusion or accounting issues.

BVSS

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

Recommendation

Instead of relying on the full balance of the vault, track the difference in the vault's balance before and after the `completeQueuedWithdrawal` call and re-deposit only this difference. If the intention is to redeposit the full balance, explicitly note in the documentation that any additional underlying funds sent to the contract will be credited on the next emergency withdrawal tracking and treated as a donation. This will ensure transparency and prevent unintended fund allocations.

Remediation

ACKNOWLEDGED: The `address(this).balance` is used as any excess funds will just be considered as donated to vault token holders. Otherwise, the owner has to call `sweepERC20` in an additional call, as the vault should not be holding any underlying token.

7.7 VAULT ID DUPLICATION RISK IN CREATEVAULT

// INFORMATIONAL

Description

The `createVault` function in `EzRVaultsFactory` allows anyone to create a vault, which is identified by a hash combining the vault's name, symbol, underlying token, and strategy. This design may lead to duplicate vaults for the same strategy but with different names or symbols, potentially creating confusion and introducing inconsistencies. The issue arises because, during vault creation, the underlying token is validated against the strategy, making the vault's ID unnecessarily complex. This could result in multiple vaults with different identifiers tied to the same strategy, creating confusion for users.

BVSS

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

Recommendation

To prevent duplicate vaults and maintain clarity, the vault ID should be simplified to use only the strategy address. This ensures that only one vault is created per strategy, avoiding potential token name/symbol confusion and ensuring the integrity of the vault system.

Remediation

ACKNOWLEDGED: It is intentional to have multiple vault with the same underlying and strategy, which enables `ezVaults` to have different use cases. The **Renzo team** would not want to limit creating only a single vault with underlying strategy.

7.8 MISSING CALL IN INITIALIZER

// INFORMATIONAL

Description

In the `EzRVault` contract, the initializer does not explicitly call the `__ReentrancyGuard_init` function, which could be considered necessary for initializing the reentrancy protection in some cases. However, since OpenZeppelin's `ReentrancyGuardUpgradeable` sets `NOT_ENTERED` as the default state (`1`) in its storage, this omission does not currently introduce any immediate functional risk. Still, best practices dictate explicitly initializing inherited components to ensure future compatibility and avoid potential issues in other versions or contexts.

BVSS

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

Recommendation

While the contract functions as expected without calling `__ReentrancyGuard_init`, it is recommended to include the call within the `initialize` function for clarity and to adhere to best practices in initializing upgradeable contracts. This ensures consistent behavior, especially in future upgrades or when integrating with other contracts.

Remediation

SOLVED: The contract now calls the reentrancy guard initializer.

Remediation Hash

https://github.com/Renzo-Protocol/EzRVaults/pull/3

7.9 FEE VALIDATION LOGIC LEADING TO REVERSION IN INITIALIZE

// INFORMATIONAL

Description

In the `EzRVault` contract's `initialize` function, if `protocolFee` is set to `BASIS_POINTS` (100%), the `_vaultFee` must be zero to avoid exceeding the basis points limit. Otherwise, the function will revert due to the combined fee exceeding `BASIS_POINTS`. This restricts the vault's configuration and may inadvertently prevent valid fee structures. Furthermore, it could lead to unnecessary reverts when attempting to set a legitimate fee structure where both `protocolFee` and `_vaultFee` are non-zero but still within the allowed range.

BVSS

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

Recommendation

Update the fee validation logic to allow `_vaultFee` to be non-zero as long as the combined value of `protocolFee + _vaultFee` does not exceed `BASIS_POINTS`. This will provide more flexibility in configuring the vault fees while ensuring they stay within the allowed percentage range.

Remediation

ACKNOWLEDGED: The **Renzo team** acknowledged this issue.

7.10 REDUNDANT UNDERLYING PARAMETER IN INITIALIZE

// INFORMATIONAL

Description

In the `initialize` function of the `EzRVault` contract, the `_underlying` parameter is passed but is redundant since the underlying token can be directly retrieved from the strategy via `underlyingToken()`. The contract already verifies that the strategy's `underlyingToken` matches the `_underlying` parameter, making the explicit input of `_underlying` unnecessary.

BVSS

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

Recommendation

Remove the `_underlying` parameter from the `initialize` function and instead rely on the `underlyingToken()` function of the strategy to fetch the underlying asset. This simplifies the contract interface and reduces the possibility of input errors during vault initialization.

Remediation

ACKNOWLEDGED: The **Renzo team** acknowledged this issue.

7.11 INCORRECT ASSUMPTION ON TOKEN BALANCE DELEGATION IN DEPOSIT

// INFORMATIONAL

Description

In the `deposit` function of the `EzRVault` contract, the comment stating that "This call assumes any balance of tokens in this contract will be delegated" is misleading. The function only delegates the amount specified in the `tokenAmount` parameter, which is transferred from the caller during the function call. Any tokens directly sent to the contract outside of this process will not be delegated and will effectively be lost or locked without further interaction.

BVSS

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

Recommendation

Update the comment to reflect the correct behavior: "Only the `tokenAmount` specified in the deposit parameter will be delegated. Any tokens sent directly to the contract outside of this call will not be delegated and may be lost." This ensures clarity for developers and users interacting with the contract, preventing potential misunderstandings or mismanagement of funds.

Remediation

ACKNOWLEDGED: The **Renzo team** acknowledged this issue.

7.12 MISSING PARAMETER IN COMPLETE EMERGENCY TRACKED WITHDRAWAL`

// INFORMATIONAL

Description

In the `completeEmergencyTrackedWithdrawal` function of the `EzRVault` contract, the `middlewareTimesIndex` parameter is hardcoded to `0` in the call to `completeQueuedWithdrawal`. This assumes that the middleware index will always be `0`, which may not be true in all cases, especially if future modifications to the strategy introduce multiple middleware indices. This hardcoding can result in incomplete or incorrect withdrawal processing if the expected index changes.

BVSS

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

Recommendation

Introduce a `middlewareTimesIndex` parameter to the `completeEmergencyTrackedWithdrawal` function, allowing flexibility for future scenarios where the middleware index might differ. This will ensure compatibility and correctness of the withdrawal process, even if multiple middleware indices are used in the strategy.

Remediation

ACKNOWLEDGED: The **Renzo team** acknowledged this issue.

7.13 UNNECESSARY COMPLEXITY IN WITHDRAWSTARTED EVENT

// INFORMATIONAL

Description

The `WithdrawStarted` event in the `EzRVault` contract emits an array for both `strategies` and `shares`, even though the vault will only ever hold a single strategy and its associated shares. This adds unnecessary complexity and redundancy to the event data, making it less efficient and harder to interpret, especially since the vault only operates with a single strategy.

BVSS

AO:S/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (0.5)

Recommendation

Simplify the `WithdrawStarted` event by emitting a single `strategy` and a single `shares` value, instead of arrays. This will streamline the event and reduce gas costs, making it easier to work with while still conveying the necessary information.

Remediation

ACKNOWLEDGED: The **Renzo team** acknowledged this issue.
