// Security Assessment

10.28.2025 - 10.28.2025

Superstate USCC- Vault Strategy

Renzo Protocol

HALBORN

Superstate USCC- Vault Strategy - Renzo Protocol

Prepared by: HALBORN

Last Updated 11/19/2025

Date of Engagement: October 28th, 2025 - October 28th, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

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

TABLE OF CONTENTS

1. Summary

2. Introduction

Renzo Protocol engaged our security analysis team to conduct a comprehensive security audit of their USCC (US Short-term Cash Coin) integration strategy within the LiquidVaults ecosystem. The primary aim was to meticulously assess the security architecture of the `UsccDepositStrategy` and `UsccDepositWithdrawalHelper` smart contracts to pinpoint vulnerabilities, evaluate existing security protocols, and offer actionable insights to bolster security and operational efficacy of their Superstate integration framework. Our assessment was strictly confined to the smart contracts provided, ensuring a focused and exhaustive analysis of their USCC deposit and withdrawal mechanisms.

3. Assessment Summary

Our engagement with Renzo Protocol spanned a 1 day period, during which we dedicated one full-time security engineer equipped with extensive experience in blockchain security, DeFi protocol analysis, and profound knowledge of Chainlink oracle implementations. The assessment focused on Renzo's novel USCC strategy implementation, which facilitates bidirectional conversion between USDC and USCC tokens through Superstate's managed infrastructure. The objectives of this assessment were to:

- Verify the correct functionality of USCC deposit and withdrawal operations within the vault ecosystem.
- Identify potential security vulnerabilities within the oracle integration and state management mechanisms.
- Evaluate the robustness of the two-phase deposit/withdrawal process and helper contract architecture.
- Provide recommendations to enhance the security and reliability of the Superstate integration.

4. Test Approach And Methodology

Our testing strategy employed a blend of manual code analysis and targeted vulnerability assessment techniques specifically tailored to Renzo's USCC integration architecture. The evaluation encompassed both the main strategy contract and the auxiliary helper contract, with particular attention to oracle safety, state management, and cross-contract communication patterns. The testing process included:

- Detailed examination of the Chainlink oracle integration and price feed validation mechanisms.
- Comprehensive analysis of the two-phase deposit/withdrawal workflow and state transitions.
- Security assessment of the helper contract's role in tracking in-flight transactions and TVL calculations.
- Event emission logic validation and potential information disclosure through incorrect state management.
- Functional testing of tolerance mechanisms and edge case handling in amount validations.

This executive summary encapsulates the pivotal findings and recommendations from our security assessment of Renzo Protocol's USCC strategy implementation. The identified issues primarily centered around oracle safety implementations and event emission logic, which could impact both security and operational transparency. By addressing the identified Chainlink oracle validation gaps and state management inconsistencies, Renzo can significantly enhance the security, reliability, and trustworthiness of their Superstate USCC integration 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

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
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 (C:N) Low (C:L) Medium (C:M) High (C:H) Critical (C: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

SEVERITY	SCORE VALUE RANGE
Informational	0 - 1.9

6. SCOPE

REMEDIATION COMMIT ID:

- https://github.com/Renzo-Protocol/LiquidVaults/pull/15/commits/100491d64bf01749ed2ab924761527aaa57c1fd8

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

7. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	2	0

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
EVENT EMISSION WITH INCORRECT EXPECTED AMOUNTS	LOW	SOLVED - 10/29/2025
UNSAFE CHAINLINK ORACLE IMPLEMENTATION WITH DIVISION BY ZERO RISK	LOW	SOLVED - 10/29/2025

8. FINDINGS & TECH DETAILS

8.1 EVENT EMISSION WITH INCORRECT EXPECTED AMOUNTS

// LOW

Description

The `UsccDepositWithdrawalHelper` contract contains a logical error in the `completeUsccWithdrawal` and `completeUsdcDeposit` functions where the expected amount variables are reset to zero before emitting completion events. In `completeUsccWithdrawal`, the `expectedWithdrawalAmountUsdc` is set to 0 before emitting `UsccWithdrawalCompleted(expectedWithdrawalAmountUsdc)`, causing the event to always emit 0 instead of the actual expected withdrawal amount. The same issue occurs in `completeUsdcDeposit` with `expectedDepositAmountUscc`.
This problem cascades to the `UsccDepositStrategy` contract, where the `completeUsdcDeposit` and `completeWithdrawUscc` functions emit `USCCDepositCompleted` and `USCCWithdrawalCompleted` events respectively. These events read the expected amounts from the helper contract after the helper has already reset them to zero, resulting in all completion events emitting 0 values instead of the meaningful expected amounts.

BVSS

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

Recommendation

Store the expected amounts in local variables before resetting the state variables to preserve the values for event emission. In `completeUsccWithdrawal`, cache `expectedWithdrawalAmountUsdc` before setting it to 0, and in `completeUsdcDeposit`, cache `expectedDepositAmountUscc` before resetting it. Use these cached values in the event emissions to ensure the events contain the actual expected amounts rather than zero values.

Remediation Comment

Client Note:
Updated event logic to cache/emit events to not get 0 values.

Remediation Hash

https://github.com/Renzo-Protocol/LiquidVaults/pull/15/commits/100491d64bf01749ed2ab924761527aaa57c1fd8

8.2 UNSAFE CHAINLINK ORACLE IMPLEMENTATION WITH DIVISION BY ZERO RISK

// LOW

Description

The `UsccDepositStrategy` contract implements unsafe Chainlink oracle usage in the `getUsdcFromUsccAmount` and `getUsccFromUsdcAmount` functions. The contract directly casts the `int256 price` returned from `usccPriceFeed.latestRoundData()` to `uint256` without validating that the price is positive. Additionally, in `getUsccFromUsdcAmount`, the contract performs division by the price value without checking if it equals zero, which would cause a revert and denial of service.
The implementation also ignores critical oracle staleness and validity checks by not validating the `roundId`, `answeredInRound`, and `updatedAt` parameters returned by `latestRoundData()`. This could lead to the contract using stale or invalid price data, potentially resulting in incorrect conversion calculations that could be exploited for arbitrage or cause significant financial losses.

BVSS

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

Recommendation

Implement comprehensive oracle safety checks including price validation, staleness detection, and division-by-zero protection. Validate that the returned price is greater than zero before casting to `uint256`. Add staleness checks by comparing `updatedAt` against a maximum acceptable age threshold. Consider implementing a fallback mechanism or circuit breaker when oracle data is stale or invalid to prevent the contract from operating with unreliable price information.

Remediation Comment

Client Note:
Added a check for negative value and staleness check of over 2 days. Expected heartbeat is 24 hours. The price is low fluctuation so we are allowing up to 48 hours for the price to get refreshed. I did NOT use the recommendation for checking answeredInRound - chainlink shows this field is deprecated "answeredInRound: Deprecated - Previously used when answers could take multiple rounds to be computed" https://docs.chain.link/data-feeds/api-reference

Remediation Hash

https://github.com/Renzo-Protocol/LiquidVaults/pull/15/commits/100491d64bf01749ed2ab924761527aaa57c1fd8

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.