// Security Assessment

11.26.2025 - 11.27.2025

P2 Vault Integration with Aave- PR 19

Renzo Protocol

HALBORN

P2 Vault Integration with Aave- PR 19 - Renzo Protocol

Prepared by: HALBORN

Last Updated 12/09/2025

Date of Engagement: November 26th, 2025 - November 27th, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
4	0	0	1	1	2

TABLE OF CONTENTS

1. Introduction

`Renzo` engaged our security analysis team to conduct a focused security audit of their `SuperstateAaveV3LeverageStrategy` contract within the LiquidVaults ecosystem. This leverage strategy enables the protocol to use USCC (Superstate Short Duration Government Securities Fund) as collateral while borrowing USDC from Aave V3 Horizon pool to amplify yields when the USDC borrow rate remains below the USCC native yield. The primary objective was to meticulously assess the security architecture of this delegation strategy to identify vulnerabilities, evaluate the integration with Aave V3 protocol, and provide actionable recommendations to enhance the security and operational robustness of the leverage mechanism. Our assessment was strictly confined to the provided smart contract implementation, ensuring a focused and exhaustive analysis of its security features and potential risk vectors.

2. Assessment Summary

Our engagement with Renzo spanned a 1 day period, during which we dedicated one full-time senior security engineer equipped with extensive experience in DeFi protocol security, deep knowledge of Aave V3 mechanics, oracle manipulation vectors, and advanced smart contract auditing capabilities. The objectives of this assessment were to:

- Verify the correct implementation of Aave V3 integration patterns and leverage mechanics
- Identify potential security vulnerabilities related to oracle dependencies and arithmetic operations
- Evaluate parameter validation and access control mechanisms
- Assess the robustness of collateral and debt position calculations under edge cases
- Provide actionable recommendations to enhance security, gas efficiency, and operational resilience

3. Test Approach And Methodology

Our testing strategy employed a comprehensive blend of manual code review and systematic vulnerability analysis tailored to leverage strategies and lending protocol integrations. The assessment focused on identifying both common DeFi vulnerabilities and strategy-specific risks inherent to leveraged positions backed by oracle-priced collateral. The testing process included:

- Detailed examination of the Aave V3 integration architecture, including collateral supply, debt borrowing, and position management workflows
- Comprehensive manual code review with emphasis on arithmetic operations, oracle dependencies, and state-changing functions
- Analysis of potential depeg scenarios and their impact on position valuation and system stability
- Evaluation of parameter validation against documented constraints and Aave V3.2.0 protocol requirements
- Assessment of gas optimization opportunities in token address retrieval patterns
- Review of variable naming conventions and code clarity for maintainability and security

This executive summary encapsulates the key findings and recommendations from our security assessment of Renzo's SuperstateAaveV3LeverageStrategy contract. The identified issues range from parameter validation gaps and gas inefficiencies to a critical arithmetic underflow vulnerability that could manifest during oracle depeg events. By addressing these issues and implementing the recommended fixes, Renzo can significantly enhance the security, operational resilience, and cost-efficiency of their leverage strategy within the LiquidVaults ecosystem.

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:

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:

$$E = \prod m_e$$

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 (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:

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

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

SEVERITY	SCORE VALUE RANGE
Informational	0 - 1.9

5. SCOPE

REPOSITORY

(a) Repository: LiquidVaults

(b) Assessed Commit ID: https://github.com/Renzo-Protocol/LiquidVaults/tree/251721a1bb82873a6215d1c8d70fb62331a981c8

REMEDIATION COMMIT ID:

- https://github.com/Renzo-Protocol/LiquidVaults/pull/19/commits/2e237ede5fbc0b61e44a60c3cd1678f4041546f8

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	1	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
POTENTIAL UNDERFLOW FROM ORACLE DEPEG IN VALUE CALCULATION	MEDIUM	SOLVED - 12/01/2025
MISSING PARAMETER VALIDATION FOR AAVE PROTOCOL OPERATIONS	LOW	SOLVED - 12/01/2025
INEFFICIENT TOKEN ADDRESS RETRIEVAL FROM AAVE POOL	INFORMATIONAL	SOLVED - 12/01/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MISLEADING VARIABLE NAMING FOR UNDERLYING ASSET	INFORMATIONAL	SOLVED - 12/01/2025

7. FINDINGS & TECH DETAILS

7.1 POTENTIAL UNDERFLOW FROM ORACLE DEPEG IN VALUE CALCULATION

// MEDIUM

Description

The `SuperstateAaveV3LeverageStrategy` contract's `underlyingValue()` function is vulnerable to arithmetic underflow when the USCC/USDC oracle depegs or returns lower-than-expected values. The function calculates the total position value by converting USCC collateral to USDC using `getUsdcFromUsccAmount()`, then subtracting the USDC debt balance. If the oracle returns a depegged price (lower USDC value per USCC), the converted collateral value could be less than the debt balance, causing an underflow on line `usdcVaultValue -= usdcDebtBalance`. This scenario can occur even when the position had a healthy health factor during operations, as the depeg would only manifest during value calculation.
The current implementation will revert on underflow, causing a Denial of Service until oracle prices normalize. However, returning zero in this scenario could enable arbitrage opportunities depending on how consuming contracts use this value.

BVSS

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

Recommendation

Refactor the calculation to use signed integers for intermediate values, allowing proper handling of negative positions.
Consider whether returning zero or reverting better aligns with the protocol's risk model. Reverting prevents operations during depeg events but causes DOS, while returning zero allows continued operation but may enable exploitation if external contracts don't validate the returned value appropriately.

Remediation Comment

SOLVED: The **Renzo Protocol team** solved this finding by adding a check for debt being higher than collateral. If a negative value is found, it will revert until the issue is resolved. Revert was chosen as we do not want deposits/withdrawals to be allowed or any other actions if this scenario happens.

Remediation Hash

https://github.com/Renzo-Protocol/LiquidVaults/pull/19/commits/2e237ede5fbc0b61e44a60c3cd1678f4041546f8

7.2 MISSING PARAMETER VALIDATION FOR AAVE PROTOCOL OPERATIONS

// LOW

Description

The `SuperstateAaveV3LeverageStrategy` contract lacks enforcement of documented parameter constraints in multiple functions. In the `supplyCollateral()` function, the `referralCode` parameter is documented to "always be 0" but no validation ensures this. Similarly, the `borrowDebt()` and `repayDebt()` functions specify that `interestRateMode` "there always should be 2 for variable rate," yet no check enforces this value. This is particularly concerning given that Aave V3.2.0 has deprecated stable/fixed rate mode (value `1`), meaning any caller could inadvertently pass invalid parameters that may cause unexpected behavior or revert from the Aave pool.

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 explicit validation for these parameters to match the documented requirements.

Remediation Comment

SOLVED: The **Renzo Protocol team** solved this finding by adding sanity checks for expected values to be passed in. Transactions will revert if invalid values are passed.

7.3 INEFFICIENT TOKEN ADDRESS RETRIEVAL FROM AAVE POOL

// INFORMATIONAL

Description

The `SuperstateAaveV3LeverageStrategy` contract retrieves aToken and variable debt token addresses inefficiently. The `getAToken()` function calls `getReserveData(_asset)` and then accesses the `aTokenAddress` field from the returned struct. Similarly, `getVariableDebtToken()` retrieves the full reserve data struct just to access `variableDebtTokenAddress`. The Aave V3 `IPool` interface provides dedicated functions `getReserveAToken()` and `getReserveVariableDebtToken()` that directly return these addresses without fetching the entire reserve data structure, making the current implementation unnecessarily gas-intensive.

BVSS

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

Recommendation

Utilize the dedicated Aave pool functions for direct token address retrieval:

```
function getAToken(address _asset) public view returns (address) {
    IPool collateralPool = getAaveV3Pool();
    return collateralPool.getReserveAToken(_asset);
}

function getVariableDebtToken(address _asset) public view returns (address) {
    IPool pool = getAaveV3Pool();
    return pool.getReserveVariableDebtToken(_asset);
}
```

Remediation Comment

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

Remediation Hash

https://github.com/Renzo-Protocol/LiquidVaults/pull/19/commits/2e237ede5fbc0b61e44a60c3cd1678f4041546f8

7.4 MISLEADING VARIABLE NAMING FOR UNDERLYING ASSET

// INFORMATIONAL

Description

The `SuperstateAaveV3LeverageStrategy` contract uses the variable name `usdcDebtToken` to store the USDC token address, which is misleading as it suggests this is the debt token (variable debt token) address. In reality, this variable stores the underlying USDC ERC20 token address, not the Aave debt token that represents borrowed USDC. The actual debt token address is retrieved via `getVariableDebtToken(address(usdcDebtToken))`. This naming convention creates confusion throughout the codebase, as `usdcDebtToken` is used to reference the underlying asset in multiple locations, including the `borrowDebt()`, `repayDebt()`, and `underlyingValue()` functions.

BVSS

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

Recommendation

Rename the variable to accurately reflect that it represents the underlying USDC token address, not the debt token. Consider using `usdcToken` or `underlyingUsdcToken` to avoid confusion. Update all references throughout the contract accordingly to maintain clarity about which token is being referenced.

Remediation Comment

SOLVED: The **Renzo Protocol team** solved this issue by renaming some variables to prevent confusion.

Remediation Hash

https://github.com/Renzo-Protocol/LiquidVaults/pull/19/commits/2e237ede5fbc0b61e44a60c3cd1678f4041546f8

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.