// Security Assessment

04.30.2025 - 05.02.2025

# NAWA Finance (Diff Review)

*Coredao*

HALBORN

# NAWA Finance (Diff Review) - Coredao

Prepared by: HALBORN

Last Updated 05/05/2025

Date of Engagement: April 30th, 2025 - May 2nd, 2025

## Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

| ALL FINDINGS | CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL |
|---|---|---|---|---|---|
| 5 | 0 | 0 | 0 | 3 | 2 |

## TABLE OF CONTENTS

## 1. Summary

## 2. Introduction

`CoreDAO` engaged our security analysis team to conduct a comprehensive security assessment 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.

## 3. Assessment Summary

Our engagement with `CoreDAO` spanned an 2 day 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.

## 4. 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 `CoreDAO` smart contract ecosystem. By addressing the identified issues and implementing the recommended fixes, `CoreDAO` can significantly boost 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)<br>Specific (AO:S) | 1<br>0.2 |

| EXPLOITABILITY METRIC ($M_E$) | METRIC VALUE | NUMERICAL VALUE |
|---|---|---|
| Attack Cost (AC) | Low (AC:L)<br>Medium (AC:M)<br>High (AC:H) | 1<br>0.67<br>0.33 |
| Attack Complexity (AX) | Low (AX:L)<br>Medium (AX:M)<br>High (AX:H) | 1<br>0.67<br>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)<br>Low (I:L)<br>Medium (I:M)<br>High (I:H)<br>Critical (I:C) | 0<br>0.25<br>0.5<br>0.75<br>1 |
| Integrity (I) | None (I:N)<br>Low (I:L)<br>Medium (I:M)<br>High (I:H)<br>Critical (I:C) | 0<br>0.25<br>0.5<br>0.75<br>1 |
| Availability (A) | None (A:N)<br>Low (A:L)<br>Medium (A:M)<br>High (A:H)<br>Critical (A:C) | 0<br>0.25<br>0.5<br>0.75<br>1 |
| Deposit (D) | None (D:N)<br>Low (D:L)<br>Medium (D:M)<br>High (D:H)<br>Critical (D:C) | 0<br>0.25<br>0.5<br>0.75<br>1 |
| Yield (Y) | None (Y:N)<br>Low (Y:L)<br>Medium (Y:M)<br>High (Y:H)<br>Critical (Y:C) | 0<br>0.25<br>0.5<br>0.75<br>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)<br>Partial (R:P)<br>Full (R:F) | 1<br>0.5<br>0.25 |
| Scope ($s$) | Changed (S:C)<br>Unchanged (S:U) | 1.25<br>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:

- 083ccbe

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

## 7. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

| CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL |
|---|---|---|---|---|
| 0 | 0 | 0 | 3 | 2 |

| SECURITY ANALYSIS | RISK LEVEL | REMEDIATION DATE |
|---|---|---|
| STRATEGY ASSIGNMENT BYPASSES UPGRADE CHECKS | LOW | NOT APPLICABLE - 05/05/2025 |
| FEE CALCULATION MAY LEAK VAULT FUNDS | LOW | RISK ACCEPTED - 05/05/2025 |
| INCONSISTENT BALANCE ACCOUNTING AND REWARD VALUE DRIFT | LOW | RISK ACCEPTED - 05/05/2025 |
| FEE PERCENTAGE SETTERS LACK UPPER BOUND CHECK | INFORMATIONAL | ACKNOWLEDGED - 05/05/2025 |
| MISLEADING CANWITHDRAW FLAG USAGE | INFORMATIONAL | SOLVED - 05/05/2025 |

## 8. FINDINGS & TECH DETAILS

### 8.1 STRATEGY ASSIGNMENT BYPASSES UPGRADE CHECKS // LOW

#### Description

In the `CoreNFTVault` contract, the `setStrategy` function allows the owner to directly set a strategy address without enforcing the upgrade delay, token compatibility check, or asset migration logic defined in `proposeStrategyUpgrade` and `upgradeStrategy`. This circumvents key upgrade safety guarantees, enabling immediate strategy replacement even if an upgrade is already pending. While gated by `onlyOwner`, this weakens governance safety by introducing inconsistent upgrade pathways.

#### BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:M/A:N/I:C/D:N/Y:N (2.3)

#### Recommendation

Remove or restrict the `setStrategy` function to only be callable when no pending upgrade exists and the current strategy is unset. Encourage using the upgrade flow for all changes.

#### Remediation Comment

**NOT APPLICABLE**: The `setStrategy` has `require(address(strategy) == address(0), "Strategy already set");`

This function is only used during initial setup and any later calls will revert because this require function requires the strategy address to be zero.

## 8.2 FEE CALCULATION MAY LEAK VAULT FUNDS

// LOW

### Description

In `CoreNFTVault.withdraw`, if `strategy.withdraw` returns 0, the fallback logic grants the user their original `coreAmount`. This creates a scenario where user withdrawals can be fulfilled using vault-held ETH that may not belong to them, especially if `strategy.withdraw` fails silently. Additionally, `_collectFee` computes fees based on `expectedCoreValue` rather than `received`, potentially misaligning fee amounts with actual yield.

### BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:M/A:N/I:C/D:N/Y:N (2.3)

### Recommendation

Revert if `strategy.withdraw` returns 0 unless business-justified. Always compute the fee based on the actual `received` amount returned from the strategy call to avoid mispricing.

### Remediation Comment

**RISK ACCEPTED**: The risk of the finding was accepted, stating that: "This is intended design because when we submit a withdraw request to B14g, it sends all matured unbonding requests amount together. So if 10 different users place unbond request and one of them withdraws, the other 10 position’s CORE tokens are also sent."

## 8.3 INCONSISTENT BALANCE ACCOUNTING AND REWARD VALUE DRIFT

// LOW

### Description

In the `DualCoreStrategy` contract, the `withdraw` function does not use a balance differential to determine the actual CORE received, unlike `withdrawDirect` in `DualCoreNFTStrategy`. Instead, it assumes correctness from the vault's transfer behavior, which may lead to discrepancies if unexpected ETH is present in the contract. Moreover, the reward value calculated during `unstake` (via `exchangeCore`) is stored and later used during withdrawal without re-evaluation. This static calculation opens a window where users could strategically delay withdrawal, waiting for favorable exchange rates, thereby gaming the reward fee model in `CoreNFTVault`.

### BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:C/D:N/Y:N (2.0)

### Recommendation

Use differential balance accounting (`address(this).balance - balanceBefore`) in the `withdraw` function to accurately reflect real token inflow. Recalculate the `coreValue` at withdrawal time to avoid reliance on stale values. If gaming the fee is a concern, consider storing both the original value and the rate used during `unstake`, and then apply a fee on the minimum of the two or introduce time-bound rate validity.

### Remediation Comment

**RISK ACCEPTED**: The risk of the finding was accepted stating that: "We cannot use differential amounts again because of the way B14g works similar to **HAL-02**"

## 8.4 FEE PERCENTAGE SETTERS LACK UPPER BOUND CHECK

// INFORMATIONAL

### Description

In `CoreNFTVault`, both `setFeePercentage` and `setDirectWithdrawFeePercentage` allow the owner to set values without validating that the input is below a reasonable maximum (e.g. 10000 = 100%). This could result in 100% or greater fee configurations, effectively locking user funds or capturing the entire withdrawal amount.

### BVSS

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

### Recommendation

Enforce a maximum cap (e.g., `< 10000`) on `rewardFeePercentage` and `directWithdrawFeePercentage` to prevent accidental or malicious misuse.

### Remediation Comment

**ACKNOWLEDGED**: The finding was acknowledged stating that: "We’ve removed upper bond limits because in future there can be a strategy that will sit on top of our vault and will require 100% of fees."

## 8.5 MISLEADING CANWITHDRAW FLAG USAGE

// INFORMATIONAL

### Description

In `CoreNFTVault`, the `UnbondRequest.canWithdraw` variable is marked as `true` after a withdrawal completes, which semantically suggests the opposite of its intended meaning. This may confuse maintainers or integrators, as `canWithdraw` typically implies that a withdrawal is available, not already executed.

### BVSS

AO:S/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 `withdrawn` or `hasWithdrawn`, or invert the logic so that `canWithdraw = true` represents actual availability for withdrawal rather than completion.

### Remediation Comment

**SOLVED:** The issue was solved.

### Remediation Hash

https://github.com/Nawa-Finance/nawa-core/commit/083ccbe87b56ce3b5097ab00c860cb337be133e5

---