// Smart Contract Security Assessment

01.07.2025 - 01.07.2025

Withdrawal Pause

Concrete

HALBORN

Withdrawal Pause - Concrete

Prepared by: HALBORN

Last Updated 03/11/2025

Date of Engagement by: January 7th, 2025 - January 7th, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
4	0	0	0	1	3

TABLE OF CONTENTS

1. Introduction

`Concrete` engaged `Halborn` to conduct a security assessment of `the new pausing mechanism for the ConcreteMultiStrategyVault` on January 7th 2025. The security assessment was scoped to the specific commit of the `sc_earn-v1` GitHub repository provided to the Halborn team. Commit hash and further details can be found in the Scope section of this report.

Key changes introduced in the commit in scope:

1. **New State Variable:**
    - `withdrawalsPaused`: Boolean to track whether withdrawals are paused.
2. **New Functionality:**
    - `toggleWithdrawalsPaused(bool withdrawalsPaused_)`: Allows the owner to pause/unpause withdrawals.
    - `_withdraw` function modified to check the `withdrawalsPaused` state and revert if withdrawals are paused.
3. **Tests:**
    - Tests added to validate that:
        - Only the admin can toggle the `withdrawalsPaused` state.
        - Withdrawals are blocked when paused and permitted when unpaused.

2. Assessment Summary

Halborn was provided 1 day for the engagement and assigned one full-time security engineer to review the security of the smart contract in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the new pausing mechanism for the `ConcreteMultiStrategyVault` contract.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were addressed by the `Concrete team`. The main one was:

- `Update the event emission by withdrawalsPaused() to ensure it uses the right values.`

3. Test Approach And Methodology

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (`solgraph`).
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Static Analysis of security for scoped contract, and imported functions (`slither`).
- Testnet deployment (`Foundry`).

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
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 (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:

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

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

FILES AND REPOSITORY

(a) Repository: sc_earn-v1

(b) Assessed Commit ID: 94eecfe

(c) Items in scope:

- src/interfaces/Errors.sol
- src/interfaces/IConcreteMultiStrategyVault.sol
- src/vault/ConcreteMultiStrategyVault.sol

Out-of-Scope: Third party dependencies and economic attacks.

REMEDIATION COMMIT ID:

- 04cc962
- 160244a

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	1

INFORMATIONAL
3

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
INCORRECT EVENT VALUES	LOW	SOLVED - 03/07/2025
POTENTIAL UPGRADEABILITY CONCERNS DUE TO NEW GLOBAL STATE VARIABLE	INFORMATIONAL	SOLVED - 03/07/2025
ERROR AND EVENT MISSING NATSPEC DOCUMENTATION	INFORMATIONAL	SOLVED - 03/07/2025
TOGGLEWITHDRAWALSPAUSED FUNCTION COULD BE DECLARED AS EXTERNAL	INFORMATIONAL	SOLVED - 03/07/2025

7. FINDINGS & TECH DETAILS

7.1 INCORRECT EVENT VALUES

// LOW

Description

The `toggleWithdrawalsPaused()` function emits the `WithdrawalPausedToggled` event, intended to log the state transition of the `withdrawalsPaused` variable. However, the emitted event incorrectly uses the same value, `withdrawalsPaused`, for both the old and new states. This results in inaccurate event logs, which can hinder effective monitoring, debugging, and external integration logic reliant on events.

```
/**
 * @notice Toggles the withdrawals paused state
 * @dev Can only be called by the owner. Emits a `WithdrawalPausedToggle
 * @param withdrawalsPaused_ The new state of the withdrawals paused stat
 */
function toggleWithdrawalsPaused(bool withdrawalsPaused_) public onlyOwne
  withdrawalsPaused = withdrawalsPaused_;
  emit WithdrawalPausedToggled(withdrawalsPaused, withdrawalsPaused_);
}
```

BVSS

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

Recommendation

Update the `WithdrawalPausedToggled` event emission to accurately reflect both the old and new states of the `withdrawalsPaused` variable.

```
/**
 * @notice Toggles the withdrawals paused state
 * @dev Can only be called by the owner. Emits a `WithdrawalPausedToggle
 * @param withdrawalsPaused_ The new state of the withdrawals paused stat
 */
function toggleWithdrawalsPaused(bool withdrawalsPaused_) public onlyOwne
  emit WithdrawalPausedToggled(withdrawalsPaused, withdrawalsPaused_);
  withdrawalsPaused = withdrawalsPaused_;
}
```

This modification ensures the `WithdrawalPausedToggled` event reliably logs the old and new states of the `withdrawalsPaused` variable, improving transparency and usability of the event.

Remediation

SOLVED: The **Concrete team** fixed this finding in commit `04cc962` by correctly emitting the old and new values as recommended.

Remediation Hash

https://github.com/Blueprint-Finance/sc_earn-v1/commit/04cc962de489932fe53766ee145fbcd39210ed40

7.2 POTENTIAL UPGRADEABILITY CONCERNS DUE TO NEW GLOBAL STATE VARIABLE

// INFORMATIONAL

Description

The addition of the `withdrawalsPaused` state variable in the `ConcreteMultiStrategyVault` contract introduces a potential upgradeability issue if this contract is part of a proxy-based upgradeable system. In such systems, storage layout consistency between the implementation contracts is critical. Adding a new state variable without considering the existing storage layout risks creating storage collisions, leading to unintended behavior or state corruption.

Note: this finding was downgraded to informational because the `Concrete team` indicated that the deployment would be from scratch.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (1.7)

Recommendation

- **Verify Storage Layout Consistency:** Before deploying the upgradable contract, ensure that the new state variable does not collide with existing variables in the storage layout. Use tools such as `OpenZeppelin Upgrades` plugins to detect potential issues.

- **Explicit Storage Slot Management:** If required, use explicit storage slot definitions for new state variables in upgradeable contracts. For example:

```
/// @notice Indicates if the vault withdrawals are paused
/// @dev Mapped to a specific storage slot to prevent collisions
bool public withdrawalsPaused;
uint256[50] private __gap; // Reserve slots for future use
```

- **Proper Testing:** Simulate and test the upgrade process in a development environment to validate that storage remains consistent and no data corruption occurs.

Remediation

SOLVED: The **Concrete team** fixed this finding in commit `04cc962` by adding storage gaps as recommended.

Remediation Hash

https://github.com/Blueprint-Finance/sc_earn-v1/commit/04cc962de489932fe53766ee145fbcd392 10ed40

7.3 ERROR AND EVENT MISSING NATSPEC DOCUMENTATION

// INFORMATIONAL

Description

The new error and event introduced in the contract lack complete `NatSpec` documentation. Proper `NatSpec` annotations are crucial for ensuring that the purpose, parameters, and context of these errors and events are clear to developers, auditors, and external consumers of the smart contract.

BVSS

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

Recommendation

Add complete `NatSpec` annotations to all errors and events, including detailed descriptions of their purpose and parameters.

Remediation

SOLVED: The **Concrete team** fixed this finding in commit `160244a` by adding NatSpec comments as recommended.

Remediation Hash

https://github.com/Blueprint-Finance/sc_earn-v1/commit/160244a23e3761629af2e0f86fbd447fa9 6b2877

7.4 TOGGLEWITHDRAWALSPAUSED FUNCTION COULD BE DECLARED AS EXTERNAL

// INFORMATIONAL

Description

The `toggleWithdrawalsPaused()` function is declared as `public`, which allows it to be called both internally and externally. However, based on its purpose and usage (exclusively by the contract owner to toggle the withdrawals paused state), there is no indication that this function needs to be called internally.

BVSS

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

Recommendation

Change the visibility of `toggleWithdrawalsPaused()` from `public` to `external` to align with Solidity best practices and better reflect its intended purpose as an externally callable function.

Remediation

SOLVED: The **Concrete team** fixed this finding in commit `04cc962` by correctly emitting the old and new values as recommended.

Remediation Hash

https://github.com/Blueprint-Finance/sc_earn-v1/commit/04cc962de489932fe53766ee145fbcd39210ed40

STATIC ANALYSIS REPORT

Description

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their abis and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

All issues identified by Slither were proved to be false positives or have been added to the issue list in this report.

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.