Security Audit for RareBay Pre-sale

[RAREPresale_V3]

Prepared by**: RareLabs Security Team**

Date**: August 13, 2025**

Version 3.0

Contents

1 Executive Summary

This security audit, conducted by the RareLabs Security Team, evaluated the `Seed.sol` smart contract using Slither static analysis. The audit identified 18 issues across multiple severity levels, as summarized in Table 1. Key findings include medium-severity reentrancy vulnerabilities, low-severity issues like missing events and timestamp dependencies, and informational observations such as pragma variability and dead code.

Table 1: Summary of Audit Findings

Issue Type	Instances	Impact	Confidence
Reentrancy Vulnerabilities	3	Medium	High
Missing Events for State Changes	2	Low	High
Timestamp Dependency	6	Low	Medium
Pragma Variability	1	Informational	High
Dead Code	3	Informational	Medium
Solidity Version	1	Informational	High
Low-Level Calls	1	Informational	High
Naming Convention	1	Informational	High

2 Introduction

The `Seed.sol` contract, implementing the $\texttt{RAREPresale}_V 3_A uditedpresaleprotocol, managestokendeposits, wit$

3 Audit Scope and Methodology

The audit focused on the Seed.sol contract and its OpenZeppelin dependencies, analyzed as of August 25, 2025.

3.1 Methodology

- **Static Analysis:** Employed Slither to detect reentrancy, timestamp dependencies, and naming issues. - **Manual Review:** Validated findings and assessed their contextual impact.

3.2 Scope

The scope includes Seed.sol and related OpenZeppelin contracts (e.g., Ownable.sol, IERC20.sol).

4 Findings and Recommendations

The following sections detail the Slither findings, categorized by severity. Additionally, a manual fix for the medium reentrancy issues is proposed below, as the contract may be pre-deployed. If deployed, consider deploying a new version with fixes; otherwise, use the suggested code modifications.

4.1 Medium Severity Findings

4.1.1 M1: Reentrancy Vulnerabilities

Description: Functions perform state changes after external calls, risking reentrancy attacks.

Instances: - Seed.sol, Lines 356-359:

```
_withdrawToken(usdtToken, to, amount, collectedUSDT);  // Line 357
collectedUSDT -= amount;  // Line 358
```

- Seed.sol, Lines 351-354:

```
_withdrawToken(wcoreToken, to, amount, collectedWCORE);  // Line 352
collectedWCORE -= amount;  // Line 353
```

- Seed.sol, Lines 211-218:

```
rareToken.transferFrom(msg.sender, address(this), amount);  // Line 216
emit TokensDeposited(amount);  // Line 217
```

Impact: Potential reentrancy could alter state variables like collectedUSDT or collectedWCORE.

Recommendation: Use the Checks-Effects-Interactions pattern; update state before external calls. To solve the issues for collectedWCORE and collectedUSDT, modify the withdraw functions as follows (manual fix for pre-deployed contract):

```
function withdrawUSDT(address to, uint256 amount) external onlyOwner {
    collectedUSDT -= amount;
    _withdrawToken(usdtToken, to, amount, collectedUSDT + amount);  //
        Adjust if the fourth param is used for checks
}

function withdrawWCORE(address to, uint256 amount) external onlyOwner {
    collectedWCORE -= amount;
    _withdrawToken(wcoreToken, to, amount, collectedWCORE + amount);
        // Adjust if needed
}
```

If the contract is pre-deployed and cannot be modified, recommend not allowing contributions (buys) with these tokens by pausing the presale or adding a modifier to restrict contributions during withdrawals. For example, add a paused state:

```
bool public paused = false;

modifier whenNotPaused() {
    require(!paused, "Presale paused");
    _;
}

function contribute(...) external whenNotPaused { ... }

function withdrawUSDT(...) external onlyOwner {
    paused = true;
    // perform withdrawal
    paused = false;
}
```

This prevents buys during withdrawals, mitigating reentrancy risks.

4.2 Low Severity Findings

4.2.1 L1: Missing Events for State Changes

Description: State updates lack events for transparency.
Instances: - Seed.sol, Lines 351-354:

```
collectedWCORE -= amount;  // Line 353
```

- Seed.sol, Lines 356-359:

```
collectedUSDT -= amount;  // Line 358
```

Impact: Reduced off-chain tracking capability.
Recommendation: Emit events for state changes:

```
emit TokensWithdrawn(amount);
```

4.2.2 L2: Timestamp Dependency

Description: Use of block.timestamp introduces potential miner manipulation risks.
Instances: - Seed.sol, Lines 167-173:

```
require(bool,string)(startTime > block.timestamp,Error: Start time
    must be in the future)  // Line 168
```

- Seed.sol, Lines 175-181:

```
require(bool,string)(startTime > block.timestamp,Error: Start time
    must be in the future)  // Line 176
```

- Seed.sol, Lines 211-218:

```
require(bool,string)(block.timestamp < presaleStartTime,Error: Cannot
    deposit after presale starts)  // Line 212
```

- Seed.sol, Lines 324-337:

```
require(bool,string)(block.timestamp > presaleEndTime,Claim: Presale
    has not ended)  // Line 325
```

- Seed.sol, Lines 407-409:

```
presaleActive && block.timestamp >= presaleStartTime &&
    block.timestamp <= presaleEndTime  // Line 408
```

- Seed.sol, Lines 411-413:

```
block.timestamp >= preseedStartTime && block.timestamp <=
    preseedEndTime  // Line 412
```

Impact: Minor timing inaccuracies possible.
Recommendation: Use block numbers for critical timing logic.

4.3 Informational Findings

4.3.1 I1: Pragma Variability

Description: Multiple Solidity versions complicate maintenance.
Instances: - $^{0}.8.0 in OpenZeppelincontracts.-^{0}.8.28 in$Seed.sol.**Impact:** Inconsistent comp