LeaderboardV2 Audit Report

Audited By: zuhaibmohd

12th June 2025

Leaderboard V2 Security Review Report

Introduction

A time-boxed security review of the **Leaderboard V2** contracts was conducted by **zuhaibmohd**, with a focus on the security aspects of the smart contracts.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where I attempt to find as many vulnerabilities as possible.

> ⚠ **Note:**
> We cannot guarantee 100% security after the review or even that the review will uncover any vulnerabilities.
> It is recommended to conduct **subsequent reviews**, run **bug bounty programs**, and enable **on-chain monitoring** to ensure long-term security.

About zuhaibmohd

zuhaibmohd is an independent smart contract security researcher.
Check out his previous work or connect on X @zuhaib44.

About LeaderboardV2

The LeaderboardV2 contract is a leaderboard system for KAITO token deposits where users can deposit tokens along with a Twitter handle and message, subject to minimum deposit amounts and cooldown periods between deposits. The contract maintains a history of all deposits and allows the owner to withdraw tokens to any address and view all transactions.

Severity Classification

Severity	Likelihood →	High	Medium	Low
Impact ↓				
High	**Critical**	High	Medium	
Medium	High	Medium	Low	
Low	Medium	Low	Low	

- **Impact:** The potential technical, economic, and reputational damage from a successful exploit.
- **Likelihood:** The probability that a given vulnerability will be discovered and exploited.
- **Severity:** The overall criticality based on the above two factors.

> **Informational**
> Findings in this category are recommended improvements to enhance code structure, usability, and overall system effectiveness.

Scope

The following smart contracts were included in the scope of the audit:

- https://basescan.org/address/0x6fc660f114716b49b07501D67395F30fc8cA4018#code

Findings Summary

ID	Title	Severity	Status
M-01	Unrestricted Arbitrary Code Execution via Owner Escape Hatch	Medium	Acknowledged
L-01	Potential Out-of-Gas Vulnerability in getAllTransactions() Due to Unbounded Array Return	Low	Acknowledged
L-02	Missing Balance Tracking and Visibility for KAITO Token Balance	Low	Acknowledged
L-03	Ownable: Does not implement 2-Step-Process for transferring ownership	Low	Acknowledged

Detailed Findings

[M-01] - Unrestricted Arbitrary Code Execution via Owner Escape Hatch

Description: The `escape` function in the LeaderboardV2 contract allows the contract owner to execute arbitrary code on any address with any calldata and value. While protected by the `onlyOwner` modifier, this represents a significant centralization risk and potential security vulnerability.

```
function escape(
    address _to,
    bytes calldata _data,
    uint256 _value
) external onlyOwner {
    (bool success, ) = _to.call{value: _value}(_data);
    require(success, "Escape hatch execution failed");
}
```

Attack Scenarios:

1. A malicious or compromised owner could drain all KAITO tokens by calling malicious contracts.
2. The owner could execute functions that manipulate token balances or approvals etc.,

Recommendations:

1. Remove or Restrict the `escape()` function.
2. Implement Time Lock for transparency.

[L-01] - Potential Out-of-Gas Vulnerability in getAllTransactions() Due to Unbounded Array Return

Description:
The `getAllTransactions()` function in the LeaderboardV2 contract currently returns the entire `entries` array containing all deposit records without any pagination mechanism. This implementation poses a significant gas limit risk as the array grows larger over time.

The current implementation:

```
function getAllTransactions()
    external
    view
    onlyOwner
    returns (DepositEntry[] memory)
{
    return entries;
}
```

The gas cost increases linearly with the size of the `entries` array. Once the array grows beyond a certain size, the function call could exceed the block gas limit, rendering the function unusable and potentially blocking access to historical transaction data.

Recommendation:
Implement pagination by modifying the function to accept start index and count parameters:

```
function getAllTransactions(uint256 startIndex, uint256 count)
    external
    view
    onlyOwner
    returns (DepositEntry[] memory)
{
    // Validate startIndex
    require(startIndex < entries.length, "Start index out of bounds");

    // Calculate the end index while preventing overflow
    uint256 endIndex = startIndex + count;

    // Adjust endIndex if it exceeds array length
    if (endIndex > entries.length) {
        endIndex = entries.length;
    }

    // Calculate actual number of entries to return
    uint256 resultCount = endIndex - startIndex;

    // Create result array
    DepositEntry[] memory result = new DepositEntry[](resultCount);

    // Copy entries to result array
    for (uint256 i = 0; i < resultCount; i++) {
        result[i] = entries[startIndex + i];
    }

    return result;
}
```

[L-02] - Missing Balance Tracking and Visibility for KAITO Token Balance

Description: The `LeaderboardV2` contract lacks a public function to view the current KAITO token balance held by the contract. While the contract can receive KAITO tokens through deposits and send them through withdrawals, there is no easy way for users or the owner to check the current balance.

Recommendation: Add a public view function to check the current KAITO token balance. Here's the recommended implementation:

```
// ... existing code ...

/// @notice Returns the current KAITO token balance of the contract
/// @return The amount of KAITO tokens held by the contract
function getKAITOBalance() public view returns (uint256) {
    return KAITO.balanceOf(address(this));
}

// ... existing code ...
```

[L-03] - Ownable: Does not implement 2-Step-Process for transferring ownership

Description: The contracts `LeaderboardV2.sol` does not implement a 2-Step-Process for transferring ownership. So ownership of the contract can easily be lost when making a mistake when transferring ownership.

Since the privileged roles have critical function roles assigned to them. Assigning the ownership to a wrong user can be disastrous. So Consider using the Ownable2Step contract from OZ https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol.

The way it works is there is a `transferOwnership` to transfer the ownership and `acceptOwnership` to accept the ownership.

Recommendations: Implement 2-Step-Process for transferring ownership via `Ownable2Step` .
