Smart Contract Audit Report

1. Overview

- **Project Name:** AURK
- **Contracts Audited:**
    - File: Staking.sol
- **Code base:** Provided by file
- **Date:** 18/06/2025

2. Executive Summary

This audit aims to assess the security of the smart contracts for the ICO protocol. We identified issues:

- High Risk
- Medium Risk
- Low Risk
- Informational / Gas Optimization Suggestions

All high and medium severity issues have been addressed / partially addressed / not addressed by the development team at the time of publishing this report.

3. Methodology

Our audit involved the following steps:

1. **Manual Code Review** – Analyzed logic for correctness, authorization, error handling, etc.
2. **Automated Testing** – Used static analysis tools such as Slither, MythX, or Hardhat plugins.
3. **Gas Usage Review** – Identified areas for optimization.
4. **Dependency Analysis** – Assessed third-party libraries (e.g., OpenZeppelin).

4. Risk Classification

Severity	Description
High	Critical vulnerabilities that can lead to loss.
Medium	Significant issues but not critical.
Low	Minor issues, often best practices.
Informational	No direct risk but useful suggestions.

6. Ownership Privileges

Staking.sol

- The owner can start the staking program, which initializes the reward period.
- The owner can stop the staking program prematurely and disable all future staking.
- The owner can prevent users from staking any new tokens.
- The owner can re-enable staking after it has been paused.
- The owner can change the percentage fees for staking and unstaking not more than 10%.
- The owner can update the minimum stake amount.
- The owner can change the total reward budget.
- The owner can add reward tokens to the contract.
- The owner can withdraw the fees from the contract.
- The owner can emergency withdraw the fees and the unused rewards from the contract.

7. Findings

7.1 High Severity

7.2 Medium Severity

[M-01] Missing limits in minimum stake and reward budget.

- **Description:** The setMinimumStake and setTotalRewardBudget functions lack crucial input validation, creating a medium-severity centralization risk. Without enforced limits, an owner could set an impossibly high minimum stake, effectively blocking new users, or set an unrealistically large reward budget to advertise a deceptive APY. This unchecked administrative power could mislead users and disrupt the contract's core functionality.
- **Mitigation:** To mitigate this, both functions require the implementation of require statements to enforce sensible upper and lower bounds on these critical parameters. This would protect the contract's integrity and shield users from potential manipulation or operational errors.
- File: Staking.sol
- Line: L215-217, L224-227
- **Status:** Open

[M-02] The owner can drain staked tokens.

- **Description:** The emergencyWithdrawAll function allows the contract owner to unilaterally withdraw the entire token balance of the contract. This includes all user-staked funds, not just fees or rewards. This function serves as an unprotected backdoor, enabling the owner to steal every asset held within the contract at any time. It represents a direct and severe threat to all users, as their deposited funds can be drained instantly by a malicious owner, completely undermining the contract's security and trustworthiness.
- **Mitigation:** The only secure mitigation for this critical vulnerability is to remove the emergencyWithdrawAll function from the contract entirely. Its existence is an unacceptable centralization risk.
- File: Staking.sol
- Line: L296-300
- **Status:** Open

7.3 Low Severity

[L-01] Missing events arithmetic.

- **Description:** It is recommended to emit all the critical parameter changes.
- File: Staking.sol
- Line: 205-228
- **Status:** Open

[L-02] Floating pragma solidity version.

- **Description:** Adding the constant version of solidity is recommended, as this prevents the unintentional deployment of a contract with an outdated compiler that contains unresolved bugs.
- File: Staking.sol
- Line: 3
- **Status:** Open

7.4 Informational/Optimization Severity

8. Conclusion

The smart contracts are generally well-written and follow modern best practices. After addressing the findings noted, the contracts should be considered safe for deployment, assuming no changes are made without further auditing.

9. Disclaimer

This audit report is not the final deployed version of the contract. So, any changes made after the audit will be considered out of the scope of the audit, and we will not be responsible for any changes related to that.