

Reward System Contract Audit Report



Updated: January 06, 2026

Table of Contents

- Introduction
 - Disclaimer
 - Scope of Audit
 - Methodology
 - Security Review Summary
 - Security Analysis
 - Contract Structure
-

Introduction

The purpose of this report is to document the security analysis and contract structure of the `RewardSystem.sol` contract. This audit report provides a deeper examination of signature mechanisms, reward invariants, and administrative controls for a thorough assessment.

Disclaimer

This report is based on the information provided at the time of the audit and does not guarantee the absence of future vulnerabilities. Subsequent security reviews and on-chain monitoring are strongly recommended.

Scope of Audit

The audit focused on the following aspects of `RewardSystem.sol`:

- Security mechanisms, including signature validation, pausing, and data integrity
- Code correctness, logical flow, and reward calculation precision
- Adherence to best practices, such as nonce usage and batch operations
- Gas efficiency, storage optimization, and upgrade paths

Methodology

The audit process involved:

- Manual code review
 - Automated analysis using Slither and Aderyn
 - Scenario-based testing using Foundry, simulating claims, pauses, and batch setups
-

Security Review Summary

Review commit hash:

- `9b507eccc17da2d0532a638d288d4b91a8adf1b0`

The `RewardSystem.sol` contract was analyzed deeply for security, highlighting EIP-712 integration, cycle limits, and role hierarchies. It demonstrates advanced protections against replay, over-claiming, and unauthorized minting, with efficient data structures for scalability.

Security Analysis

The `RewardSystem` utilizes NFT-based ROI claims using off-chain signatures for secure, gas-efficient distributions. EIP-712 (`EIP712Upgradeable`) standardizes claim hashing (`CLAIM_TYPEHASH`), with `ECDsa.recover` verifying signatures against a trusted `signer`, preventing forgery. Nonces (`nonces[msg.sender]`) and deadlines enforce one-time use, mitigating replay attacks.

Reward integrity is upheld in `claimROI`: staking timestamps are cross-verified with `ICOAuth`, quarters are capped by `maxCycles[at1Type][batchId]`, and amounts are recalculated against APYs (`veApyBp`, etc.) to match expected values, preventing inflation. Internal helpers like `_validateClaimEligibility` check cooldowns (`block.timestamp >= nftClaim.lastClaimed + quarterDuration`) and cycle limits, while `_verifyRewardAmounts` fetches NFT prices from `IGenesisKey` for precise computations (`veAmount = (price * veApyBp[...] * quartersToClaim) / (4 * BASIS_POINTS)`), ensuring economic fairness.

Admin functions (`setMaxCycles`, `setRewardDetailsForBatches`) are `OPERATOR_ROLE`-restricted, with array length checks for batch safety. Role management allows revocation, reducing insider risks. Storage uses nested mappings (`investors[userId].nftClaims`) for efficient per-NFT tracking, with keys hashed (`getKey`) for uniqueness. Events (`ROIClaimed`, `VEPayoutDue`) log distributions for traceability. Upgradeability via `__gap` (95 slots) and initializers supports evolution without data corruption. Risks like signer compromise are offset by updatable `setSigner`, and interface dependencies (`IMintableToken`) assume trusted implementations. The design excels in verifiable, capped rewards, resilient to timing attacks or data manipulation.

Contract Structure

The contract emphasizes hierarchical data and batch efficiency:

- **State Variables:** Roles (`PAUSER_ROLE`, `OPERATOR_ROLE`), nonces, `investors` (mapping `userId` to `InvestorInfo` with `nftKeys` array and `nftClaims` mapping), APY mappings (`veApyBp`, etc.), interfaces (`gPRLZ`, `sPRLZ`, `coaAuth`), `signer`, `quarterDuration`, and `genKeyAddresses`.
- **Initialization:** Configures EIP-712 domain, grants roles, sets tokens/signer, and defaults `quarterDuration` to 90 days.
- **Core Functions:**
 - **Claiming:** `claimROI` orchestrates validation (`_verifySignature`), eligibility (`_validateClaimEligibility`), updates (`_updateClaimInfo`), and distributions (`_distributeRewards` with mints and events).
 - **Admin:** Batch setters (`setMaxCycles`, `setRewardDetailsForBatches`, `setGenesisKeyAddresses`) for scalable configs; updaters (`setSigner`, `setGPRLZ`) with zero-address checks.
 - **Pausing:** `pause/unpause` for functional control.
 - **Getters:** `calculateExpectedRewards`, `getRewardDetails`, `getAllStakedKeysForUser` for transparency.