Airdrop Contract Audit Report

Updated: January 08, 2026

Table of Contents

Introduction

The purpose of this report is to document the security analysis and contract structure of the `AirdropGenKey.sol` contract. This audit examines airdrop mechanics, signature-based claims, reward distributions, and integration with external interfaces for secure and controlled asset airdrops.

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 `AirdropGenKey.sol`:

- Security mechanisms, including signature validation, nonce management, and access controls for claims and withdrawals
- Code correctness, logical flow, and compliance with EIP-712, ERC721, and ERC20 standards
- Adherence to best practices for upgradeability, reward handling, and integration with ICOAAuth
- Gas efficiency, error handling, and prevention of replay attacks or unauthorized claims

Methodology

The audit process involved:

- Manual code review of inheritance, overrides, signature hashing, and claim flows
- Automated analysis using Slither and Aderyn to detect common vulnerabilities like replay attacks or access control issues
- Scenario-based testing using Foundry for simulating claims, reward distributions, signature verifications, and edge cases (e.g., expired signatures or insufficient balances)

Security Review Summary

Review commit hash:

- `0c1fcf72ede03bf697c60f435e7a866f0312c9d3`

The `AirdropGenKey.sol` contract provides a secure system for signature-based airdrops and rewards, with nonce protections and role-based administration to mitigate unauthorized access and replay risks.

Security Analysis

As an upgradeable contract, `AirdropGenKey` inherits from OpenZeppelin's `Initializable`, `AccessControlUpgradeable`, and `EIP712Upgradeable`, ensuring safe initialization, role management, and typed data hashing for signatures. The `OPERATOR_ROLE` restricts sensitive operations like setting the signer, reward tokens, or withdrawing assets, while `DEFAULT_ADMIN_ROLE` handles initial setup.

Claim functions (`claim`, `claimReward`) use EIP-712 signatures verified against a backend `signer` to authorize distributions, incorporating nonces (`userNonces`, `userRewardNonces`) and deadlines to prevent replay and expiration issues. Digests are marked as used post-verification, and user authenticity is enforced (e.g., via `coaAuth.walletToUserId`). NFT claims track token IDs incrementally (`currentAtlTokenID`), ensuring sequential distribution, while token claims (SPRLZ) use `SafeERC20` for secure transfers.

The contract integrates with `COAAuth` for user ID validation, adding an external trust layer. Withdrawal functions (`withdrawGenKey`, `withdrawGenKeys`) are operator-only, preventing arbitrary asset drains. The `onERC721Received` callback safely handles incoming NFTs with event logging.

Events (e.g., `GenesisKeyClaimed`, `RewardClaimed`) enable on-chain monitoring. Upgradeability is supported with a 40-slot `__gap` for future expansions without storage collisions. Limits like `MAX_NFT_PER_TXN` mitigate gas exhaustion in batch claims.

Automated tools (Slither/Aderyn) flagged no high-severity issues, though minor optimizations (e.g., in loops) were noted. Testing confirmed resilience against common attacks like signature malleability, replay, or front-running.

The contract prioritizes security through explicit checks, audited dependencies, and minimal attack surfaces.

Contract Structure

The contract extends upgradeable standards with custom airdrop and reward logic:

- **State Variables**: Signer address; mappings for claims, nonces, token IDs (available, claimed), user rewards; enum for reward types; ICOAAuth interface; max NFTs per transaction.
- **Initialization**: Sets roles, signer, and EIP-712 domain.
- **Core Functions**:
  - `claim`: Signature-based claim for initial YARD NFTs with nonce and deadline checks.
  - `claimReward`: Handles NFT or token rewards based on type, with user ID validation and sequential ID assignment.
  - `setSigner` / `setCOAAuthAddress` / `setMaxClaimPerTxn`: Operator updates for configuration.
  - `withdrawGenKey` / `withdrawGenKeys`: Operator withdrawals of held NFTs.
  - `setRewardTokens` / `setRewardCurrentTokenID`: Batch setup for reward addresses and starting IDs.
  - `onERC721Received`: ERC721 callback for receiving NFTs.
- **Getters**: Functions to retrieve claimed IDs, reward amounts, token addresses, and user-specific data.

- **Helpers/Constants**: EIP-712 typehashes; events for logging; enum for reward categorization.