Migrator Contract Audit Report

Updated: January 07, 2026

Table of Contents

Introduction

The purpose of this report is to document the security analysis and contract structure of the `Migrator.sol` contract. This audit examines token and NFT migration mechanics, access controls, reentrancy protections, and compliance with upgradeable standards for a secure transition from V1 to V2 assets.

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 `Migrator.sol`:

- Security mechanisms, including role-based access, reentrancy guards, and NFT/ERC20 transfer controls
- Code correctness, logical flow, and compliance with ERC721/ERC20 standards
- Adherence to best practices for upgradeability and migration processes
- Gas efficiency, error handling, and prevention of unauthorized actions

Methodology

The audit process involved:

- Manual code review of inheritance, overrides, and migration flows
- Automated analysis using Slither and Aderyn to detect common vulnerabilities like reentrancy or access control issues
- Scenario-based testing using Foundry for simulating NFT/ERC20 migrations, role assignments, and edge cases (e.g., insufficient balances or approvals)

Security Review Summary

Review commit hash:

- `faaf061e7ff5604909844bd0a35ebc4d83c4ce21`

The `Migrator.sol` contract provides a secure framework for asset migrations, with role-restricted configurations and reentrancy protections to prevent exploits during transfers.

Security Analysis

As an upgradeable contract, `Migrator` inherits from OpenZeppelin's `AccessControlUpgradeable` and `Initializable`, ensuring safe initialization and role management. The `SIGNER_ROLE` restricts critical setup functions (e.g., `setERC721Requirements`, `setTokenInfo`), while `DEFAULT_ADMIN_ROLE` handles initial grants.

Migration functions (`migrateERC20Token`, `migrateAllAsset`, `migrateAllByTransfer`) are protected by a custom `nonReentrant` modifier to mitigate reentrancy attacks, particularly during token transfers. ERC20 migrations enforce allowance checks, balance validations, and price-based calculations to prevent over-transfers or unauthorized withdrawals. NFT migrations validate ownership and approvals before withdrawing old assets and either minting new ones (via `mintAsFreeMinter`) or transferring pre-received ones.

The `onERC721Received` implementation safely handles incoming NFTs, storing them in `receivedTokenIds` for later distribution. Withdrawal functions like `withdrawGenKeys` are role-restricted to prevent arbitrary asset extraction.

Events (e.g., `TokenMigrationCompleted`, `NFTMigrationCompleted`) provide transparency for on-chain monitoring. The design avoids direct burns or mints outside controlled paths, reducing supply manipulation risks. Upgradeability is supported with a 99-slot `__gap` for future expansions without storage collisions.

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

The contract prioritizes security through minimalism, explicit checks, and audited OpenZeppelin dependencies.

Contract Structure

The contract extends upgradeable standards with custom migration logic:

- **State Variables**: `Requirements` struct for V1/V2 addresses and price; mappings for minted IDs, migrated tokens, and received NFTs; counters for total migrations; reentrancy lock.
- **Initialization**: Grants admin role and sets default price.
- **Core Functions**:
  - `migrateERC20Token`: Handles ERC20 swaps with allowance/balance checks and price multiplication.
  - `migrateAllAsset`: Batch NFT migration via withdrawal and new minting.
  - `migrateAllByTransfer`: Batch NFT migration using pre-received tokens.
  - `setERC721Requirements` / `setTokenInfo`: Role-restricted setup for migration parameters.
  - `withdrawGenKeys`: Admin withdrawal of held NFTs.
  - `onERC721Received`: ERC721 callback for receiving NFTs.
- **Helpers**: Internal functions for NFT withdrawal, minting/transferring, and array management (e.g., `_removeReceivedTokenId`).

- **Modifiers/Errors**: `nonReentrant` guard; custom `TransactionMessage` errors for clear failure reasons.