SILVER PRLZ (sPRLZ) 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 `PearlzToken.sol` contract (referred to as `sPRLZ` in the system). This audit provides a deeper examination of token mechanics, access controls, and integration points for a thorough assessment of its robustness, adherence to best practices, and overall design integrity.

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 `PearlzToken.sol`:

- Security mechanisms, including minting authorization, blacklisting, and fund recovery
- Code correctness, logical flow, and ERC20 compliance with upgradeability
- Adherence to best practices, such as ownership management and arithmetic safety
- Gas efficiency, storage optimization, and upgrade paths

Methodology

The audit process involved:

- Manual code review with line-by-line analysis of imports, overrides, and custom logic
- Automated analysis using Slither and Aderyn for static vulnerability detection
- Scenario-based testing using Foundry, simulating minting, blacklisting, transfers, and upgrades

---

Security Review Summary

Review commit hash:

- `0c1fcf72ede03bf697c60f435e7a866f0312c9d3`

The `PearlzToken.sol` contract (sPRLZ) was analyzed deeply for security, focusing on multi-minter delegation, blacklisting enforcement, and native asset recovery. As an upgradeable ERC20 with ownership and

minting hierarchies, it maintains strong controls over supply and transfers, ensuring integration safety within the broader system (e.g., with the `RewardSystem`).

Security Analysis

The sPRLZ contract extends `ERC20Upgradeable` and `OwnableUpgradeable`, implementing a delegated minting model suitable for ecosystem rewards (e.g., via `RewardSystem`). Security centers on `minters` mapping

- `mintMaster`: initially the owner controlling addresses in the minter mapping using the `listMinter` function . This hierarchy allows flexible delegation
- `minters`: minting is restricted to authorized addresses in the minters mapping

The contract's minimalism reduces attack surface: Limited external calls (e.g., no dependencies beyond OpenZeppelin), no reentrancy-prone hooks, and owner-restricted functions (`onlyOwner` for blacklisting and BNB transfer) uphold access invariants. Risks like mintMaster centralization are offset by transferable control, and Solidity ^0.8.9's built-in overflow checks complement SafeMath. Overall, the design ensures secure, delegable supply management, resilient to unauthorized minting or fund drains, with emphasis on ecosystem interoperability.

Contract Structure

The contract follows an upgradeable ERC20 pattern with custom extensions for minting and recovery:

- **State Variables**: `deadWallet` (constant for potential burns), `_isBlacklisted` (mapping for address restrictions), `mintMaster` (address for minter control), `minters` (mapping for authorized minters).
- **Initialization**: `initialize` sets ownership, token name/symbol ("Pearlz", "PRLZ"), assigns `mintMaster` to owner, and performs initial mint.
- **Core Functions**:
  - **Minting**: `mint` (restricted to minters) for supply increase; `listMinter` and `setMintMasterAddress` (mintMaster-restricted) for delegation.
  - **Blacklisting**: `blacklistAddress` (owner-only) to toggle restrictions.
  - **Getters**: `returnNameAndVersion` (pure, returns "pRLZ, 1.1" for versioning).