OpenZeppelin | security

ZEROBASE Token Audit

ZEROBASE

August 7, 2025

Table of Contents

Summary

Type	Cross-Chain
Timeline	From 2025-08-06 To 2025-08-06
Languages	Solidity

Total Issues	13 (0 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	3 (0 resolved)
Notes & Additional Information	10 (0 resolved)

Scope

OpenZeppelin audited the ZeroBase-Pro/Economics repository at commit 3c55c3f.

In scope were the following files:

```
Economics
├── src
│   └── ZEROBASE.sol
└── script
    ├── CrossChainTransfer.s.sol
    ├── SetTrustedRemote.s.sol
    └── ZeroBase.s.sol
```

System Overview

The ZEROBASE token (ZBT) is an ERC-20-compliant utility token of the ZEROBASE network, a decentralized, real-time ZK-proof system. The token has an initial total supply of 1 billion tokens on each chain, with a standard 18-decimal precision. Built on LayerZero's Omnichain Fungible Token (OFT) standard, ZBT enables seamless cross-chain functionality while preserving native ERC-20 behavior on each supported blockchain. The `ZEROBASE` contract inherits from OpenZeppelin's `Ownable` contract, serving as the primary administrator for LayerZero management functions and initially receiving the complete token allocation.

The audited deployment scripts provide a comprehensive framework for establishing cross-chain token operations between the Ethereum Sepolia testnet and the BSC testnet environments. The `ZeroBase.s.sol` deployment script facilitates contract creation using LayerZero's endpoint (`0x6EDCE65403992e310A62460808c4b910D972f10f`), ensuring proper integration with the omnichain protocol infrastructure.

The `SetTrustedRemote.s.sol` configuration script establishes the peer relationships required for cross-chain functionality between deployed contract instances. The `CrossChainTransfer.s.sol` transfer execution script demonstrates the operational capability by initiating a token transfer from Ethereum Sepolia to BSC testnet. It configures a transfer of 100 ZBT tokens with a 5% tolerance (minimum 95 ZBT delivery) to a specified recipient address. The script implements fee estimation by querying the contract's `quoteSend` function before execution, ensuring adequate native token payment for LayerZero messaging fees. The transfer includes execution options specifying 200000 gas units for the destination chain processing, with the deployer's address designated as the refund recipient for any unused fees.

Current deployments include the Ethereum Sepolia instance at `0xb733e5ff6361770771dcfe58491713661ac11bb3` and the BSC testnet deployment at `0x9c8749892de34dce62c06eb2e0fb1d62b8601a49`. Both contracts maintain consistent ownership under the `0x2b2E23ceC9921288f63F60A839E2B28235bc22ad` address, ensuring unified governance across the multi-chain ecosystem.

During the audit, all configuration parameters, including LayerZero Endpoint IDs, chain identifiers, and network-specific settings, were validated against the official LayerZero Documentation.

Privileged Roles

The ZEROBASE token inherits ownership from OpenZeppelin's Ownable contract. The `_owner` address:

- will be set up as a multisig
- is able to execute all `onlyOwner` functions on the ERC-20 contract
- is assigned as the `_delegate` in LayerZero's `OAppCore` contract

In addition, a separate `_receiver` address is designated to receive the initial total supply of ZBT tokens. Similarly to the owner, the receiver will also be set up as a multisig wallet.

Low Severity

L-01 Hardcoded Etherscan API Keys

The `ZeroBase.s.sol` file contains commented-out Forge CLI commands that also contain hardcoded Etherscan API keys. While the commands are not actively executed, keeping sensitive information such as API keys in the open-source code poses a security and maintainability risk.

Consider removing any hardcoded API keys from the codebase entirely, including the commented-out sections. If these CLI commands are useful for reference, consider documenting them in a secure, non-source-controlled location and using environment variables for any sensitive values. This helps ensure that sensitive credentials are not inadvertently exposed or committed. Given that the current set of keys are already exposed, consider revoking them and regenerating new keys for future usage.

L-02 Missing Test Suite

The repository lacks a test suite to verify the correct behavior of the deployed contracts and essential functionality such as deployment correctness, admin role assignment, and token minting. This poses a risk to contract reliability and security, as regressions or bugs may go unnoticed. By convention, Foundry-based projects place test scripts in the `test/` directory, with filenames ending in `.t.sol`.

Consider implementing a comprehensive test suite using Foundry, covering core features such as deployment validation, admin role behavior, minting logic, and token transfers. A robust test suite is essential for ensuring contract correctness and supporting safe ongoing development.

L-03 Limited Testing in `CrossChainTransferScript`

The `CrossChainTransferScript` currently tests cross-chain token transfers only from Sepolia to BSC testnet. This makes the script incomplete, as it does not test token transfers in the reverse direction (i.e., BSC testnet to Sepolia), which is essential for validating bidirectional functionality.

Additionally, the script uses a hardcoded recipient address, `0x2b2E23ceC9921288f63F60A839E2B28235bc22ad`, which corresponds to the owner of the ZEROBASE token on the BSC test network. Using a privileged address for testing may not accurately reflect typical usage scenarios, as such an address could bypass certain permission or validation checks that would apply to ordinary users.

Consider extending the script to test cross-chain transfers from BSC to Sepolia, and replacing the hardcoded address with dynamically generated or randomly selected test addresses.

Notes & Additional Information

N-01 `README.md` Lacks ZEROBASE Token Documentation, Setup Instructions, and Contribution Guidelines

The current `README.md` in the project repository provides minimal information and lacks ZEROBASE token documentation, project setup, testing procedures, and contribution guidelines.

Since the project is open-source, consider expanding the README to include a clear token overview, setup and testing instructions, contribution guidelines (if contributions are accepted), and a secure vulnerability disclosure process to support community involvement and responsible security reporting.

N-02 Improper Development Environment Configuration

The repository's development environment appears to be misconfigured. The `foundry.toml` file suggests the use of Git submodules for managing external dependencies, but the required `.gitmodules` file is missing, which breaks proper submodule integration. Additionally, a `node_modules/` folder is present without an accompanying `package.json` file, making it unclear which Node.js packages are intended to be used or managed. These inconsistencies hinder reproducibility and can cause confusion for collaborators or CI setups.

Consider ensuring that the development environment is correctly and fully configured. If Git submodules are being used for Foundry dependencies, include the `.gitmodules` file and ensure that submodules are properly initialized and updated. If Node.js dependencies are needed, add a `package.json` file to define and manage them.

N-03 Unnecessary Files and Folders Committed to Repository

The repository includes several files and directories that do not need to be version-controlled, such as `broadcast/`, `cache/`, `lib/forge-std/`, `node_modules/`, `out/`, and `.DS_Store`. These are typically build artifacts, dependencies, or system-generated files that can clutter the repository, unnecessarily increase its size, and cause merge conflicts or platform-specific issues.

Consider removing these files and directories from the repository and adding them to the `.gitignore` file to prevent future commits. Keeping the repository clean helps improve clarity, reduce noise in diffs, and streamline collaboration.

N-04 Unused Import

The import `OFTReceipt` from `"../node_modules/@layerzerolabs/lz-evm-oapp-v2/contracts/oft/interfaces/IOFT.sol"` is not used in `CrossChainTransfer.s.sol` and could be removed.

Consider removing unused imports to improve the overall clarity and readability of the codebase.

N-05 Literal Number With Many Digits

The `1000000000` literal number used to define the initial supply of the ZEROBASE token uses many digits.

For literal numbers with a lot of digits, consider using scientific notation and underscores. This will help improve readability and prevent misleading code, which could have unintended consequences.

N-06 Missing Security Contact

Providing a specific security contact (such as an email address or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Consider adding a NatSpec comment containing a security contact above the `ZEROBASE` contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

N-07 Missing Docstrings

The `ZEROBASE` contract is missing docstrings.

Consider thoroughly documenting the contract using the Ethereum Natural Specification Format (NatSpec). This includes adding a contract-level docstring to describe its overall purpose, a docstring for the constant to explain its role and significance, and a docstring for the constructor to clarify its behavior and any initialization it performs. Clear documentation improves readability, maintainability, and integration with development tools.

N-08 Incomplete Automation in `SetTrustedRemoteScript` for Peer Configuration

`SetTrustedRemoteScript` only contains `setPeer` configuration for the Sepolia network. Specifically, it registers the ZEROBASE token deployed on the BSC testnet as a peer for the ZEROBASE token on Sepolia. However, the counterpart configuration, registering the Sepolia-deployed token as a peer on the BSC testnet, is present in the code but has been commented out. Although the peers have been correctly configured on both testnets, Sepolia and BSC, the current script requires manual intervention (e.g., uncommenting lines) to complete the full peer registration.

Consider refactoring the script to automate peer registration for both directions by loading configuration parameters from a dedicated environment or config files.

N-09 Commented-Out Line of Code

Commented-out code can negatively impact code clarity and readability.

Line 10 of the `ZeroBase` script has redundant, commented-out code.

Consider removing all instances of commented-out and redundant code to improve the clarity and maintainability of the codebase.

N-10 SPDX License Set to Unlicensed

The SPDX License identifier for all contracts has been set to `UNLICENSED`.

To avoid legal issues regarding copyright and follow best practices, consider adding SPDX license identifiers to files as suggested by the Solidity documentation.

Conclusion

The `ZEROBASE` token contract was audited along with the deployment and configuration scripts that integrate it with the LayerZero Omnichain protocol. No high- or medium-severity issues were identified. Some fixes were suggested to improve the readability of the codebase and facilitate future integrations and development.

The audit identified that the project can benefit from the addition of a comprehensive automated test suite. Testing should cover both contract-level functionality and cross-chain interactions to ensure reliability, especially given the complexity of omnichain architecture and the critical nature of cross-chain token transfers.