OpenZeppelin | security

Omnichain Fungible Token (OFT) Adapter Audit

Fireblocks

July 14, 2025

Table of Contents

Summary

Type	Stablecoins
Timeline	From 2025-06-10 To 2025-06-13
Languages	Solidity

Total Issues	11 (10 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	1 (1 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	4 (3 resolved)
Notes & Additional Information	5 (5 resolved)

Scope

OpenZeppelin audited pull request #16 of the fireblocks/fireblocks-smart-contracts repository at commit 197f206.

In scope were the following files:

```
contracts
└── bridge-adapter
    ├── modules
    │   ├── PauseCapable.sol
    │   ├── RoleBasedOwnable.sol
    │   └── SalvageCapable.sol
    ├── interfaces
    │   └── IERC20MintableBurnable.sol
    └── FungibleLayerZeroAdapter.sol
```

System Overview

Pull request #16 adds the `FungibleLayerZeroAdapter` contract, which enables developers to integrate their existing ERC-20 tokens with the LayerZero protocol allowing them to bridge their tokens across different chains. The adapter is based on the official example adapter implementation provided by LayerZero. However, instead of using the locking mechanism, where the funds are locked in the source chain and unlocked on the destination chain, the `FungibleLayerZeroAdapter` contract implements a burn/mint mechanism.

This means that, upon each bridge transfer, the tokens are burned on the source chain and then freshly minted on the destination chain. This way, the cumulative total supply across all the supported chains represents the actual total supply of the token. In addition, the contract implements a fail-safe mechanism in case a bridge transfer fails on the destination chain. In such an event, the tokens are stored in the adapter and can be later retrieved by either the user or an admin.

The `PauseCapable` contract implements basic pause functionality while the `RoleBasedOwnable` contract implements a role-based access control system. Furthermore, the `SalvageCapable` contract allows the admin to retrieve any ETH, ERC-721, and ERC-20 tokens that are stuck in the adapter and are not part of the adapter's user flow.

Security Model and Trust Assumptions

Since the adapter is built upon the LayerZero messaging standard, it inherits its security model. Therefore, it is important to ensure that the adapter is configured correctly. This includes, but is not limited, to defining the adapter's peers which will be the individual adapters deployed on each supported chain. A misconfiguration can potentially lead to a loss of funds. For example, a peer could be set up with the wrong destination address such that the messages will not be delivered properly. In this case, users' tokens would be burned on the source chain without the corresponding minting operation being performed on the destination chain.

For access control, the protocol uses the role-based implementation of the OpenZeppelin `AccessControl` library. The various sensitive functionalities have been distributed among the following privileged roles:

- `DEFAULT_ADMIN_ROLE`: The default administrator of the adapter contract. This role is assigned at deployment and can manage all other roles.
- `PAUSER_ROLE`: The role can pause all functionality in the adapter contract including the management of roles.
- `SALVAGE_ROLE`: The role can retrieve any ETH, ERC-721, and ERC-20 tokens that are stuck in the adapter and are not part of the adapter's user flow.
- `EMBARGO_ROLE`: The role can retrieve any embargoed tokens and send them to a custom address.
- `CONTRACT_ADMIN_ROLE`: The role has all the permissions that had originally been given to the `onlyOwner` address in the OFT contract.

All of the above roles are assumed to be held by trusted entities acting in the best interests of the system.

High Severity

H-01 Existing Embargoed Amounts Are Overwritten if Another Bridge Transfer Fails

In `FungibleLayerZeroAdapter`, the `_credit` function stores the bridged amount for failed transfers in the `_embargoLedger` mapping. These funds can later be retrieved by the original recipient of the bridged tokens. However, if there is an existing amount in the ledger for a user, a subsequent failed transfer will overwrite it. Consequently, the user will not be able to retrieve those tokens anymore.

Consider adding the embargoed amount on top of the existing value in the `_embargoLedger` mapping.

***Update**: Resolved in pull request #19 at commit 0bf36ee.*

Medium Severity

M-01 DoS From Unbounded Loop During Salvage of Tokens

In the `_authorizeSalvageERC20` function, there is an unbounded `for` loop that iterates over all the embargoed token balances in the `_embargoLedger` mapping and adds them up, storing the total in the `totalEmbargoedBalance` variable. However, if the mapping becomes sufficiently large, the loop could potentially exceed the block gas limit, causing the function to revert. Since an attacker can create additional entries in the `_embargoLedger` mapping at a marginal gas cost on both the source and destination chains, they can execute the attack at will.

Consider tracking the total amount of embargoed funds in a separate state variable to avoid iterating over the unbounded `_embargoLedger` mapping in a `for` loop.

***Update**: Resolved in pull request #19 at commit 5924a48.*

Low Severity

L-01 Allow Users to Retrieve Their Embargoed Funds

When a bridge transfer fails on the destination chain, the tokens are added to the embargo ledger. A user can either retrieve these tokens to the original recipient's address or the admin can send them to an address of their choosing. This design introduces a trusted role that can move embargoed funds at will, thereby opening up a potential attack vector.

Instead of allowing the admin to send the tokens to an arbitrary address, consider allowing the recipient of the funds to retrieve the tokens to a custom address themselves. This will help prevent a single account from having control over all the embargoed tokens.

***Update**: Acknowledged, not resolved. The Fireblocks team stated:*

> *No changes required. We understand that this introduces a privileged role, and as with all such roles, proper safeguards must be in place. This is a common pattern across our contracts, addressed through secure MPC wallets and strict Transaction Authorization Policies. Our current implementation already allows recipients to withdraw their embargoed funds, directly or via a delegate. Supporting custom withdrawal addresses is a potential enhancement for future iterations, but we do not see it as a replacement.*

L-02 Compromised Pauser Can Indefinitely Pause the Adapter

When the `PAUSER_ROLE` pauses the adapter, the assignment and revocation of roles are blocked. In addition, only the `PAUSER_ROLE` is able to unpause the adapter. Therefore, if the `PAUSER_ROLE` is compromised and the adapter is paused, there is no way for the admin to unpause the contract. Any tokens that are bridged or embargoed at the time will be lost.

Consider allowing the `DEFAULT_ADMIN_ROLE` to manage roles when the `FungibleLayerZeroAdapter` contract is paused. This would enable the adapter contract to be unpaused and the `PAUSER_ROLE` to be removed from the compromised account within a single transaction.

***Update**: Resolved in pull request #20 at commit 5559527.*

L-03 Incomplete Docstrings

Throughout the codebase, multiple instances of incomplete docstrings were identified:

- In the `embargoedAccounts` function of `FungibleLayerZeroAdapter.sol`, not all return values are documented.
- In the `renounceRole` function of `RoleBasedOwnable.sol`, the `role` and `account` parameters are not documented.

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of a contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

***Update**: Resolved in pull request #21 at commit 10a669f.*

L-04 Unsafe ABI Encoding

It is not uncommon to use `abi.encodeWithSignature` or `abi.encodeWithSelector` to generate calldata for a low-level call. However, the first option is not typo-safe while the second option is not type-safe. As a result, both of these methods are error-prone and should be considered unsafe.

The use of `abi.encodeWithSelector` within `FungibleLayerZeroAdapter.sol` is unsafe.

Consider replacing all the occurrences of unsafe ABI encodings with `abi.encodeCall`, which checks whether the supplied values actually match the types expected by the called function and also avoids errors caused by typos.

***Update**: Resolved in pull request #22 at commit 32402c5.*

Notes & Additional Information

N-01 Variables Initialized With Their Default Values

Within `FungibleLayerZeroAdapter.sol`, multiple instances of variables being initialized with their default values were identified:

- The `i` variable
- The `i` variable
- The `totalEmbargoedBalance` variable
- The `i` variable

To avoid unnecessary gas consumption, consider refraining from initializing variables with their default values.

***Update**: Resolved in pull request #23 at commit efb1e78.*

N-02 Multiple Optimizable State Reads

In `FungibleLayerZeroAdapter.sol`, multiple instances of optimizable storage reads were identified. One such example is the `_peerEids.length` storage read.

Consider reducing SLOAD operations that consume unnecessary amounts of gas by caching the values in a memory variable.

***Update**: Resolved in pull request #24 at commit 611fd65.*

N-03 Missing Security Contact

Providing a specific security contact (such as an email 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.

Throughout the codebase, multiple instances of contracts not having a security contact were identified:

- The `FungibleLayerZeroAdapter` contract
- The `IERC20MintableBurnable` interface
- The `PauseCapable` abstract contract
- The `RoleBasedOwnable` abstract contract
- The `SalvageCapable` abstract contract

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

***Update**: Resolved in pull request #26 at commit 7cd2120.*

N-04 Outdated Solidity Versions

Using an outdated version of Solidity can lead to vulnerabilities and unexpected behavior in contracts. Hence, it is important to keep the Solidity compiler version up to date.

Throughout the codebase, multiple instances of files using outdated Solidity versions were identified:

- The `FungibleLayerZeroAdapter.sol` is using an outdated Solidity version (`0.8.22`).
- The `IERC20MintableBurnable.sol` is using an outdated Solidity version (`0.8.22`).
- The `PauseCapable.sol` is using an outdated Solidity version (`0.8.22`).
- The `RoleBasedOwnable.sol` is using an outdated Solidity version (`0.8.22`).
- The `SalvageCapable.sol` is using an outdated Solidity version (`0.8.22`).

Consider taking advantage of the latest Solidity version to improve the overall readability and security of the codebase. Regardless of the Solidity version that is used, consider pinning the version consistently throughout the codebase to prevent bugs due to incompatible future releases. Additionally, consider compiling with `via-ir` which will apply better gas optimizations without requiring changes to the code.

***Update**: Resolved in pull request #27 at commit 84779cb.*

N-05 Unnecessary `unchecked` Blocks

Starting with Solidity version 0.8.22, the compiler optimizes the loop counter increments in certain `for` loops by not performing overflow checks and instead automatically generating an unchecked arithmetic increment for the counter. However, for this optimization to apply, the following conditions must be met:

- Loop condition must be `i` < ... (strict < comparison).
- Loop counter `i` must be a local integer variable, unmodified in the body or the condition.
- Right-hand side must match `i`'s type.
- Increment must be `i++` or `++i` (no compound operations - e.g., `i += 1`).

Within `FungibleLayerZeroAdapter.sol`, multiple instances of such `for` loops with redundant `unchecked` blocks were identified:

- The `unchecked{ ++i; }` block in the `listPeers` function
- The `unchecked{ ++i; }` block in the `_setPeer` function
- The `unchecked{ ++i; }` block in the `_authorizeSalvageERC20` function

To improve the overall clarity, intent, and readability of the codebase, consider removing the unnecessary `unchecked` blocks.

***Update**: Resolved in pull request #28 at commit f4ccf54.*

Conclusion

The audited codebase implements a LayerZero Omnichain Fungible Token (OFT) adapter for the cross-chain bridging of ERC-20 tokens. The adapter is based on the official example adapter provided by LayerZero and implements additional logic to burn/mint tokens instead of locking them up in the adapter.

During the audit, a high-severity issue was identified that could cause a user's embargoed funds to be lost if a subsequent bridge transfer failed on the destination chain. A medium-severity issue was also identified whereby an unbounded `for` loop could lead to a DoS attack. Overall, the codebase was found to be well-written and thoroughly documented.

The Fireblocks team is appreciated for being highly responsive and providing valuable support throughout the audit process.