OtterSec

LayerZero Endpoint Dollar

Security Assessment

January 28th, 2026 — Prepared by OtterSec

Nicholas R. Putra nicholas@osec.io

Zhenghang Xiao kiprey@osec.io

Table of Contents

Executive Summary **2**

Overview 2

Key Findings 2

Scope 2

General Findings **3**

OS-LED-SUG-00 | Missing Reentrancy Guard on Unwrap 4

Appendices

Vulnerability Rating Scale **5**

Procedure **6**

© 2026 Otter Audits LLC. All Rights Reserved.

01 — Executive Summary

Overview

LayerZero engaged OtterSec to assess the `LZEndpointDollar` program. This assessment was conducted between January 23rd and January 27th, 2026. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 1 finding throughout this audit engagement.

In particular, we suggested adding a reentrancy guard to the unwrap function (OS-LED-SUG-00).

Scope

The source code was delivered to us in a Git repository at https://github.com/LayerZero-Labs/Endpoint-Token. This audit was performed against commit e08e5d9.

A brief description of the program is as follows:

Name	Description
LZEndpointDollar	An upgradeable ERC-20 wrapper that mints and burns a standardized *dollar* token 1:1 against owner-whitelisted 6-decimal ERC-20 underlyings. It provides permit support, strict admin controls, and safe wrapping/unwrapping to unify multiple dollar assets under a single canonical token.

© 2026 Otter Audits LLC. All Rights Reserved.

02 — General Findings

Here, we present a discussion of general findings identified during our audit. While these findings do not pose an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-LED-SUG-00	`unwrap` performs an external token transfer without a reentrancy guard.

© 2026 Otter Audits LLC. All Rights Reserved.

Missing Reentrancy Guard on Unwrap

OS-LED-SUG-00

Description

Although `LZEndpointDollar::wrap` first burns the amount before performing the transfer and is not currently vulnerable to reentrancy, it still performs an external token transfer. Thus, adding `nonReentrant` provides defence-in-depth and maintains symmetry with `wrap`, which currently utilizes the `nonReentrant` modifier.

>_ *LZEndpointDollar.sol* SOLIDITY

```solidity
/**
* @inheritdoc ILZEndpointDollar
* @dev Only whitelisted tokens can be wrapped.
*/
function wrap(address _token, address _to, uint256 _amount)
    public
    virtual
    nonReentrant
    onlyWhitelistedToken(_token)
{
    IERC20(_token).safeTransferFrom(msg.sender, address(this), _amount);
    _mint(_to, _amount);
    emit TokenWrapped(_token, msg.sender, _to, _amount);
}

/**
* @inheritdoc ILZEndpointDollar
*/
function unwrap(address _token, address _to, uint256 _amount) public virtual
    ↪ onlyWhitelistedToken(_token) {
    _burn(msg.sender, _amount);
    IERC20(_token).safeTransfer(_to, _amount);
    emit TokenUnwrapped(_token, msg.sender, _to, _amount);
}
```

Remediation

Add the `nonReentrant` modifier to `unwrap`.

Patch

Resolved in b701948.

© 2026 Otter Audits LLC. All Rights Reserved.

A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

CRITICAL	Vulnerabilities that immediately result in a loss of user funds with minimal preconditions. Examples: • Misconfigured authority or access control validation. • Improperly designed economic incentives leading to loss of funds.
HIGH	Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions. • Exploitation involving high capital requirement with respect to payout.
MEDIUM	Vulnerabilities that may result in denial of service scenarios or degraded usability. Examples: • Computational limit exhaustion through malicious input. • Forced exceptions in the normal user flow.
LOW	Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions.
INFO	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants. • Improved input validation.

© 2026 Otter Audits LLC. All Rights Reserved.

B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.

© 2026 Otter Audits LLC. All Rights Reserved.