LayerZero OVault

Security Assessment

July 16th, 2025 — 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-OVT-SUG-00 | Code Maturity 4

OS-OVT-SUG-01 | Code Duplication 5

Appendices

Vulnerability Rating Scale **6**

Procedure **7**

01 — Executive Summary

Overview

LayerZero engaged OtterSec to assess the `ovault` program. This assessment was conducted between July 14th and July 15th, 2025. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 2 findings throughout this audit engagement.

We made a recommendation to remove duplicate code instances (OS-OVT-SUG-01), and suggested addressing certain inconsistencies in the codebase to ensure adherence to coding best practices. (OS-OVT-SUG-00).

Scope

The source code was delivered to us in a Git repository at https://github.com/LayerZero-Labs/devtools. This audit was performed against commit 13bee6b.

A brief description of the program is as follows:

Name	Description
ovault	It defines a cross-chain ERC4626-compatible vault system utilizing LayerZero, allowing users to deposit or redeem assets on a hub chain and receive tokens on a target chain

02 — General Findings

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

ID	Description
OS-OVT-SUG-00	Suggestions regarding inconsistencies in the codebase and ensuring adherence to coding best practices.
OS-OVT-SUG-01	Suggestions for removing duplicate code instances.

Code Maturity OS-OVT-SUG-00

Description

1. Add slippage checks to `depositSend` and `redeemSend` to ensure users receive at least a minimum expected output, protecting them from rounding errors or unfavorable vault pricing.
2. The default ERC4626 implementation does not account for inflation attacks when the vault is initially empty. It is possible for an attacker to front-run deposits by donating assets to inflate share prices. To mitigate this, deployers should pre-fill the vault.

Remediation

Implement the above-mentioned suggestions.

Patch

1. Issue #1 was addressed in ea38deb.
2. Issue #2 was addressed in 373eedb by adding documentation highlighting the attack vector.

Code Duplication OS-OVT-SUG-01

Description

1. `OVaultComposer::retry` redundantly checks the same condition twice in a row, as shown below, which is unnecessary and should be simplified to improve clarity.

>_ *packages/ovault-evm/contracts/OVaultComposer.sol* SOLIDITY

```
function retry(bytes32 _guid, bool _removeExtraOptions) external payable nonReentrant {
    FailedMessage memory failedMessage = failedMessages[_guid];
    if (_failedGuidState(failedMessage) != FailedState.CanOnlyRetry) revert
        ↪  CanNotRetry(_guid);
    if (_failedGuidState(failedMessage) != FailedState.CanOnlyRetry) revert
        ↪  CanNotRetry(_guid);
  [...]
}
```

2. The assignment of `ASSET_DECIMAL_CONVERSION_RATE` and `SHARE_DECIMAL_CONVERSION_RATE` in constructor is duplicated in `OVaultComposer::constructor`.
3. The assignment of `failedMessages[_guid]` inside `OVaultComposer::lzCompose` is duplicated.

Remediation

Remove the duplicate code instances to eliminate unnecessary code.

Patch

1. The duplicate code in issue #1 was removed in 41e6b52.
2. The duplicate code in issue #2 was removed in c1718c1.

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.

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.