Zellic

Prepared for
Moksh Mohindra
LayerZero

Prepared by
Katerina Belotskaia
Vlad Toie
Zellic

June 6, 2025

OFTAdapter Burn

Smart Contract Security Assessment

Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for LayerZero from June 3rd to June 4th, 2025. During this engagement, Zellic reviewed OFTAdapter Burn's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- How are the tokens burned/minted on each side of the bridge?
- Are there any potential edge cases in terms of maintaining the supply balance of the tokens on each side of the bridge?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody
- Other contract components that are inherited but not included in the scope of this assessment

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped OFTAdapter Burn contracts, there were no security vulnerabilities discovered.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	0
Informational	0

2. Introduction

2.1. About OFTAdapter Burn

LayerZero contributed the following description of OFTAdapter Burn:

> Introduce a custom OFT adapter contract that integrates a new IMintSelfBurnToken interface to burn ERC20 tokens when debiting from the source chain and mint the same amount when crediting on the destination chain. This shifts the bridging mechanism from “lock-and-unlock” to burn-and-mint, ensuring a single global supply of ERC20 tokens across chains while preserving OFT semantics.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no

hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3. Scope

The engagement involved a review of the following targets:

OFTAdapter Burn Contracts

Type	Solidity
Platform	EVM-compatible
Target	devtools
Repository	https://github.com/LayerZero-Labs/devtools ↗
Version	`5b677f9db097bc26c2240aa73e4778e0c71acfb8`
Programs	`MintSelfBurnToken` `MintSelfBurnOFTAdapter`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of one person-day. The assessment was conducted by two consultants over the course of two calendar days.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io ↗

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Katerina Belotskaia
Engineer
kate@zellic.io ↗

Vlad Toie
Engineer
vlad@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

June 3, 2025	Start of primary review period
June 4, 2025	End of primary review period

3. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

Function: `_credit(address _to, uint256 _amountLD, uint32 /* _srcEid */)`

This is an internal function that handles the token credit logic during cross-chain token receipt.

Inputs

- `_to`
  - **Control**: Decoded from cross-chain–message data.
  - **Constraints**: N/A.
  - **Impact**: The address of the token receiver for tokens transferred across chain.
- `_amountLD`
  - **Control**: Decoded from cross-chain–message data.
  - **Constraints**: N/A.
  - **Impact**: The amount of tokens to be minted to the receiver address.

Branches and code coverage

Intended branches

- The expected amount of tokens are minted.
  - ☐ Test coverage
- The `innerToken` balance of the contract is not changed.
  - ☐ Test coverage
- The `_to` balance is increased by the `_amountLD`.
  - ☐ Test coverage

Function call analysis

- `IMintSelfBurnToken(address(innerToken)).mint(_to, _amountLD);`
  - **What is controllable?** N/A.

- **If the return value is controllable, how is it used and how can it go wrong?** There is no return value here.
- **What happens if it reverts, reenters or does other unusual control flow?** Mints the specified amount of tokens to the `_to` account balance. The function reverts if `_to` is the zero address; however, in this implementation, `_to` is replaced with the `0xdead` address in such cases, ensuring the zero address is never passed to the mint function.

Function: `_debit(address _from, uint256 _amountLD, uint256 _minAmountLD, uint32 _dstEid)`

This is an internal function that handles token debit logic during cross-chain token transfers.

Inputs

- `_from`
  - **Control**: Full control.
  - **Constraints**: `msg.sender`.
  - **Impact**: The address of the token sender.
- `_amountLD`
  - **Control**: Full control.
  - **Constraints**: The caller must own enough tokens to transfer.
  - **Impact**: The amount of OFT tokens specified by the caller to send to the `_dstEid` chain.
- `_minAmountLD`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Determine the minimum amount of tokens that should be transferred after dust removal.
- `_dstEid`
  - **Control**: Full control.
  - **Constraints**: `peers` should contain the `_dstEid`. Only the owner can set up eIDs that can be used.
  - **Impact**: The ID of the destination chain where tokens will be sent.

Branches and code coverage

Intended branches

- The expected amount of tokens are burned.

- ☐ Test coverage
- The `innerToken` balance of the contract is not changed.
  - ☐ Test coverage

Negative behavior

- The caller does not own enough tokens to transfer.
  - ☐ Negative test
- The caller does not provide an approval.
  - ☐ Negative test
- The `amountReceivedLD` is less than `_minAmountLD`.
  - ☐ Negative test
- The `_dstEid` is not supported.
  - ☐ Negative test

Function call analysis

- `super._debitView(_amountLD, _minAmountLD, _dstEid);`
  - **What is controllable?** `_amountLD`, `_minAmountLD`, and `_dstEid`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns `amountSentLD` and `amountReceivedLD`, which are equal since the default `_debitView` implementation from the OFTCore contract is used. These values represent `_amountLD` after dust removal based on the `decimalConversionRate`.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if `amountReceivedLD` is less than `_minAmountLD`.
- `IERC20(address(innerToken)).transferFrom(_from, address(this), amountSentLD);`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value here.
  - **What happens if it reverts, reenters or does other unusual control flow?** Transfers the `amountSentLD` (without dust) to this contract. This function reverts if `msg.sender` has not granted sufficient allowance.
- `IMintSelfBurnToken(address(innerToken)).burn(amountSentLD);`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value here.
  - **What happens if it reverts, reenters or does other unusual control flow?**

Burns the specified amount of tokens from the contract's balance. There should be no issues here, as the specified tokens are transferred to the contract's balance in advance.

4. Assessment Results

During our assessment on the scoped OFTAdapter Burn contracts, there were no security vulnerabilities discovered.

4.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.