Zellic

NativeOFTV2

Smart Contract Security Assessment

June 16, 2023

Prepared for:

Layer Zero

Prepared by:

Syed Faraz Abrar and Nipun Gupta

Zellic Inc.

Contents

About Zellic 2

1 Executive Summary 3

1.1 Goals of the Assessment . . . 3

1.2 Non-goals and Limitations . . . 3

1.3 Results . . . 3

2 Introduction 4

2.1 About NativeOFTV2 . . . 4

2.2 Methodology . . . 4

2.3 Project Overview . . . 5

2.4 Project Timeline . . . 5

3 Threat Model 6

3.1 Module: NativeOFTV2.sol . . . 6

4 Audit Results 9

4.1 Disclaimer . . . 9

About Zellic

Zellic was founded in 2020 by a team of blockchain specialists with more than a decade of combined industry experience. We are leading experts in smart contracts and Web3 development, cryptography, web security, and reverse engineering. Before Zellic, we founded perfect blue, the top competitive hacking team in the world. Since then, our team has won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io or follow @zellic_io on Twitter. If you are interested in partnering with Zellic, please contact us at hello@zellic.io.

1 Executive Summary

Zellic conducted a security assessment for Layer Zero from `June 13th to June 16th, 2023`. During this engagement, Zellic reviewed NativeOFTV2's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.1 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:

- Could an attacker send another user's `NativeOFTV2` tokens without prior approval?
- Does the chain-native token to `NativeOFTV2` token conversion work as intended?
- Could an attacker abuse a logical bug to mint `NativeOFTV2` tokens to themselves for free?

1.2 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

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

1.3 Results

During our assessment on the scoped NativeOFTV2 contracts, we discovered no findings.

2 Introduction

2.1 About NativeOFTV2

NativeOFTV2 is Layer Zero's example implementation of a cross-chain wrapped native token. On the source chain, where NativeOFTV2 is deployed, users can: deposit the native tokens and receive NativeOFTV2 at a 1:1 ratio, withdraw the native tokens back for NativeOFTV2 at a 1:1 ratio, and send NativeOFTV2 tokens to a different Layer Zero supported chain.

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 the contracts' 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 code base in general. We look for violations of industry best practices and guidelines and code quality stan-

dards. We also provide suggestions for possible optimizations, such as gas optimization, upgradeability 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 Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of eight person-days. The assessment was conducted over the course of four calendar days.

Contact Information

The following project managers were associated with the engagement:

Chad McDonald, Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Syed Faraz Abrar, Engineer
faith@zellic.io

Nipun Gupta, Engineer
nipun@zellic.io

2.4 Project Timeline

The key dates of the engagement are detailed below.

June 13, 2023	Start of primary review period
June 16, 2023	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 smart 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.

3.1 Module: NativeOFTV2.sol

Function: `deposit()`

This is used to convert chain-native tokens to NativeOFTV2 tokens at a 1:1 rate.

Branches and code coverage (including function calls)

Intended branches

- Should swap the chain-native tokens to NativeOFTV2 tokens at a 1:1 rate.
  ☑ Test coverage

Function: `sendFrom(address _from, uint16 _dstChainId, byte[32] _toAddress, uint256 _amount, LzCallParams _callParams)`

This function is used to send NativeOFTV2 tokens to a remote chain

Inputs

- `_from`
  - **Control**: Fully controlled.
  - **Constraints**: Must be either the caller's address, or an address that has approved the caller to spend the required `_amount` of NativeOFTV2 tokens.
  - **Impact**: This is the address the `NativeOFTV2` tokens are sent from if possible.
- `_dstChainId`
  - **Control**: Fully controlled.
  - **Constraints**: Must be valid, otherwise function execution will fail later on.
  - **Impact**: This is the destination chain's ID.
- `_toAddress`

  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: This is the address on the destination chain that the tokens will be sent to.
- `_amount`
  - **Control**: Fully controlled.
  - **Constraints**: The `_from` address must own at least `_amount` NativeOFTV2 tokens, or `msg.value` must contain the required amount of native tokens that will be swapped for NativeOFTV2 tokens.
  - **Impact**: This is the amount of NativeOFTV2 tokens to be sent to the destination chain.
- `_callParams`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: Optional relay specific parameters. Can be left empty.

Branches and code coverage (including function calls)

Intended branches

- Should work correctly with `_from == msg.sender` with the required amount of tokens already being owned by `_from`.
  ☑ Test coverage
- Should work correctly with `_from == msg.sender` with the required amount of tokens not owned by `_from`, but supplied through `msg.value`.
  ☑ Test coverage
- Should work correctly with `_from ≠ msg.sender` with the required amount of tokens already being owned by `_from`.
  ☑ Test coverage
- Should work correctly with `_from ≠ msg.sender` with the required amount of tokens not owned by `_from`, but supplied through `msg.value`.
  ☑ Test coverage
- Should work correctly when using custom `_callParams` (with correct parameters).
  ☑ Test coverage

Negative behaviour

- Should revert of `_from ≠ msg.sender` and `_from` has not approved `msg.sender`.
  ☑ Negative test
- Should revert with insufficient `msg.value`.
  ☑ Negative test

- Should revert if custom `_callParams` are used and the minimum gas limit is set too low.
  ☑ Negative test
- Should revert if custom `_callParams` are used and a minimum gas limit has not been set.
  ☑ Negative test

Function: `withdraw(uint256 _amount)`

This is used to convert NativeOFTV2 tokens to chain-native tokens at a 1:1 rate.

Inputs

- `_amount`
  - **Control**: Fully controlled.
  - **Constraints**: Caller must own at least this amount of NativeOFTV2 tokens.
  - **Impact**: This is the amount of NativeOFTV2 tokens that are burned and converted to chain-native tokens.

Branches and code coverage (including function calls)

Intended branches

- Should swap the NativeOFTV2 tokens to chain-native tokens at a 1:1 rate.
  ☑ Test coverage

Negative behaviour

- Should revert if the caller does not own at least `_amount` NativeOFTV2 tokens.
  ☑ Negative test

4 Audit Results

At the time of our audit, the audited code was not deployed to any chains.

During our assessment on the scoped NativeOFTV2 contracts, we discovered no findings.

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 audit version of 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.

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.