Zellic

LayerZero Solidity Examples

Smart Contract Security Assessment

May 21, 2022

Prepared for:

Ryan Zarick and Isaac Zhang

LayerZero Labs

Prepared by:

Katerina Belotskaia and Aaron Esau

Zellic Inc.

Contents

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 email us at hello@zellic.io or contact us on Telegram at https://t.me/zellic_io.

1 Introduction

1.1 About LayerZero Solidity Examples

LayerZero is an omnichain interoperability protocol designed for lightweight message passing across chains. LayerZero provides authentic and guaranteed message delivery with configurable trustlessness. The protocol is implemented as a set of gas-efficient, non-upgradable smart contracts. LayerZero Solidity Examples refers to the demo contracts that may be used as templates for projects built on the LayerZero omnichain network.

1.2 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 open-source 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. We analyze the scoped smart contract code using automated tools to quickly sieve out and catch these "shallow" bugs. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, etc. 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 manually review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents. We also thoroughly examine the specifications and designs themselves for inconsistencies, flaws, and vulnerabilities. This involves use-cases that open the opportunity for abuse, such as flawed tokenomics or share pricing, arbitrage opportunities, etc.

Complex integration risks. Several high-profile exploits have been the result of not any bug within the contract itself, but rather an unintended consequence of its interaction with the broader DeFi ecosystem. We perform a meticulous review of all of the contract's possible external interactions, and summarize the associated risks; for example: flash loan attacks, oracle price manipulation, MEV/sandwich attacks, etc.

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

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; we assign it on a case-by-case basis based on our professional judgment and experience. As one would expect, both the severity and likelihood of an issue affect its impact; for instance, a highly severe issue's impact may be attenuated by a very low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Similarly, Zellic organizes its reports such that the most important findings come first in the document, rather than 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 numerous soft factors, such as our clients' threat models, their business needs, project timelines, etc. We aim to provide useful and actionable advice to our partners that consider their long-term goals, rather than simply a list of security issues at present.

1.3 Scope

The engagement involved a review of the following targets:

LayerZero Solidity Examples Contracts

Repository	https://github.com/LayerZero-Labs/solidity-examples
Versions	`d58564f70100d2192f6fd5db6803ac453d023232`
Programs	• BasedOFT • GlobalCappedOFT • LzApp • NonblockingLzApp • OFT • OFTCore • ONFT1155 • ONFT1155Core • ONFT721 • ONFT721Core • PausableOFT • ProxyOFT • ProxyONFT1155 • ProxyONFT721 • UniversalONFT721
Type	Solidity
Platform	EVM-compatible

1.4 Project Overview

Zellic was contracted to perform a security assessment with two consultants, for a total of 2 person-week. The assessment was conducted over the course of 1 calendar week.

Contact Information

The following project managers were associated with the engagement:

Jasraj Bedi, Co-Founder
jazzy@zellic.io

Stephen Tong, Co-Founder
stephen@zellic.io

The following consultants were engaged to conduct the assessment:

Katerina Belotskaia, Engineer
kate@zellic.io

Aaron Esau, Engineer
aaron@zellic.io

1.5 Project Timeline

The key dates of the engagement are detailed below.

May 16, 2022	Start of primary review period
May 20, 2022	End of primary review period

1.6 Disclaimer

This assessment does not provide any warranties on finding all possible issues within its scope; i.e., the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees on any additional code added to the assessed project after our assessment has concluded. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program. Finally, this assessment report should not be considered financial or investment advice.

2 Executive Summary

Zellic conducted an audit for LayerZero Labs from May 16th to May 20th, 2022 on the scoped contracts and discovered 2 finding. Fortunately, no critical issues were found. We applaud LayerZero Labs for their attention to detail and diligence in maintaining incredibly high code quality standards in the development of LayerZero Solidity Examples.

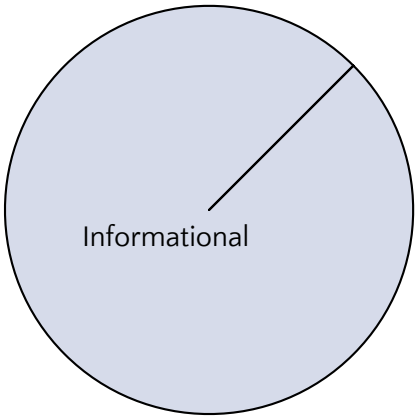
Our 2 findings were both informational in nature. Additionally, Zellic recorded its notes and observations from the audit for LayerZero Labs's benefit at the end of the document.

Zellic thoroughly reviewed the LayerZero Solidity Examples codebase to find any technical issues outlined in the Methodology section of this document. Specifically, taking into account the project's threat model, we focused heavily on issues that would break the integrity or availability of the token examples.

Our general overview of the code is that it was very well-organized and structured. The code coverage is high and tests are included for the vast majority of the functions. The documentation was extensive. The code was easy to comprehend and intuitive.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	0
Informational	2



3 Detailed Findings

3.1 Out-of-bounds read from `toAddressBytes` allows undefined behavior

- **Target**: OFTCore, ONFT721Core, ONFT1155Core
- **Category**: Coding Mistakes
- **Likelihood**: N/A
- **Severity**: Informational
- **Impact**: Informational

Description

The following assembly code may read up to 32 bytes out of bounds of `toAddressBytes` because the size of `toAddressBytes` is not checked:

```
address toAddress;
assembly {
    toAddress := mload(add(toAddressBytes, 20))
}
```

Impact

There is no direct security impact of this instance of out-of-bounds read. However, this code pattern allows undefined behavior and is potentially dangerous. In the past, even low-level vulnerabilities have been chained with other bugs to achieve critical security compromises.

Recommendations

The size of a `uint` is 32 bytes. So, the branch that uses the `MLOAD` instruction should require that the size of `toAddressBytes` is greater than or equal to the read size of 32 bytes.

Remediation

TBD

3.2 Messaging library provides a function to renounce ownership

- **Target**: LzApp
- **Category**: Business Logic
- **Likelihood**: N/A
- **Severity**: Informational
- **Impact**: Informational

Description

The abstract LzApp contract implements Ownable which provides a method named `renounceOwnership` that removes the current owner (reference). This is likely not a desired feature.

Impact

If `renounceOwnership` were called for any reason, the contract would be left without an owner.

In the case of the PausableOFT contract, if an owner were to pause the token and renounce ownership, transactions would be blocked forever. Of course, pausable tokens have centralization risk as an owner can choose to make the token worthless at any time.

Also, the impact of the `renounceOwnership` method call is for a LzApp.sol contract that in case the receiving of messages from the another chain is blocked due to failed receiving, the call of `forceResumeReceive` function will not be possible.

Recommendation

Override the `renounceOwnership` function:

```
function renounceOwnership() public {
    revert("This feature is not available.");
}
```

Remediation

TBD

4 Discussion

The purpose of this section is to document miscellaneous observations the we made during the assessment.

4.1 Inherent centralization risk

In LzApp.sol the contract's admin has full control over the `trustedRemoteLookup` list via `setTrustedRemote` function and central authority over functions such as `setConfig`.

```
function setTrustedRemote(uint16 _srcChainId, bytes calldata
    _srcAddress) external onlyOwner {
    trustedRemoteLookup[_srcChainId] = _srcAddress;
    emit SetTrustedRemote(_srcChainId, _srcAddress);
}
```

```
function setConfig(uint16 _version, uint16 _chainId, uint _configType,
    bytes calldata
    _config) external override onlyOwner {
    lzEndpoint.setConfig(_version, _chainId, _configType, _config);
}
```

In case of a private key compromise, an attacker could:

- change UltraLightNode configuration for the current user application. For example, set the addresses of the fully controlled oracle and relayer contracts.
- add a trusted remote that causes `_creditTo` to be called fraudulently
- remove all trusted remotes, causing the token not to be sendable cross-chains (which may render some tokens worthless).

We strongly recommend using a multi-signature address wallet. This would prevent an attacker if a private key were compromised. Also, users should be aware of the possibility of suspending or completely stopping the cross-chain messages transfer via this user application contract from the owner of the contract. So there is a possibility that the sent tokens will not be delivered.