Zellic

LayerZero Core

Smart Contract Security Assessment

June 3, 2022

Prepared for:

Ryan Zarick and Isaac Zhang

LayerZero Labs

Prepared by:

Katerina Belotskaia

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 Core

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-upgradeable smart contracts. LayerZero Core refers to the core contracts behind 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, and so forth 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, and so forth.

Complex integration risks. Several high-profile exploits have not been the result of any bug within the contract itself, but rather they are 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, and so forth.

Code maturity. We review for possible 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, upgradeability weaknesses, centralization risks, and so forth.

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 being rated 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 numerous soft factors, such as our clients' threat models, their business needs, their project timelines, and so forth. We aim to provide useful and actionable advice to our partners that consider their long-term goals rather than simply provide a list of security issues at present.

1.3 Scope

The engagement involved a review of the following targets:

LayerZero Core Contracts

Repository	https://github.com/LayerZero-Labs/LayerZero

Versions	`Core: 27bf69645060c81b81396adc3afd965d18f9024d`
	`Core: 8116723c0876b9e65d77e196021d805de500dbad` (patch review)
	`Prooflib: 75b6094bc091faa898427cfc2db0f85f6a3eeec8`

Contracts	• 'UltraLightNodeV2.sol' • 'TreasuryV2.sol' • 'RelayerV2.sol' • 'NonceContract.sol' • 'EndpointLite.sol' • 'LayerZeroPacket.sol' • 'MPTValidator01.sol'
Type	Solidity
Platform	EVM-compatible

1.4 Project Overview

Zellic was contracted to perform a security assessment with one consultant for a total of one person-week. The assessment was conducted over the course of one 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

1.5 Project Timeline

The key dates of the engagement are detailed below.

May 30, 2022	Start of primary review period
June 3, 2022	End of primary review period
September 4, 2022	Patch review on LayerZero core

1.6 Disclaimer

This assessment does not provide any warranties on 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 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 30th to June 3rd, 2022, on the scoped contracts and discovered 2 findings. This was a re-review of LayerZero core, specifically focusing UltraLightNodeV2. 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 Core.

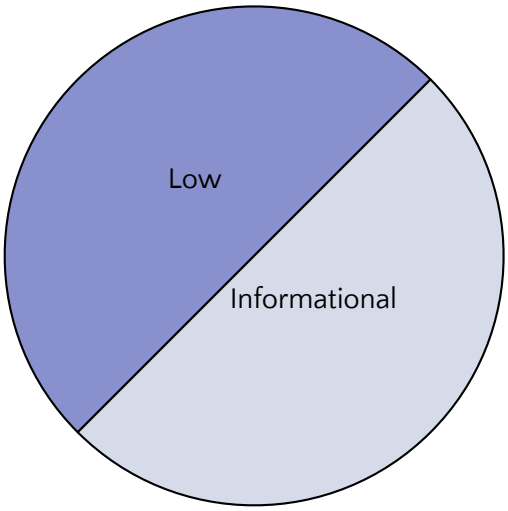
Of the 2 findings, 1 was of low severity and 1 was informational in nature.

Zellic thoroughly reviewed the LayerZero Core codebase to find protocol-breaking bugs as defined by the documentation and any technical issues outlined in the Methodology section of this document. Specifically, taking into account LayerZero's threat model, we focused heavily on issues that would break core invariants like executing payloads without the agreement of both Oracle and Relayer, or executing them out of order, leading to desynchronization between source and destination chains.

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 majority of the functions. The documentation was adequate, although it could be improved. The code was easy to comprehend, and in most cases, intuitive.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	1
Informational	1



3 Detailed Findings

3.1 Lack of check of the recipient to the zero address in withdrawNative function

- **Target**: UltraLightNodeV2
- **Category**: Coding Mistakes
- **Likelihood**: Low
- **Severity**: Low
- **Impact**: Low

Description

The commission in native tokens for Relayers, Oracles, and Treasury addresses is stored in the `nativeFees`mapping. If the caller of the`withdrawNative`function has any funds accrued for him, they will be sent to the address of the`_to`argument using the`call` function for transfer founds. This function does not reject the transaction in case of an error but returns the result as a boolean value. But since sending funds to the zero address is not an incorrect transfer, returned value will be true so the transaction will be completed successfully.

```
function withdrawNative(address payable _to, uint _amount) external
    override nonReentrant {
        nativeFees[msg.sender] = nativeFees[msg.sender].sub(_amount);

        (bool success, ) = _to.call{value: _amount}("");
        require(success, "LayerZero: withdraw failed");
        emit WithdrawNative(msg.sender, _to, _amount);
    }
```

Impact

In case of an incorrect transfer of funds to the zero address, all funds sent will be lost.

Recommendations

Add checks for the `to` argument.

```
require(_to ≠ address(0x0), "LayerZero: _to cannot be zero address");
```

Remediation

LayerZero Labs acknowledged this finding and implemented a fix in commit `efc2c0cf ffbc10abfe87163e0c3ab29ce4f0d1a4`.

3.2 Lack of check that chainAddressSizeMap is setted for dstChainId

- **Target**: UltraLightNodeV2
- **Category**: Coding Mistakes
- **Likelihood**: N/A
- **Severity**: Informational
- **Impact**: Informational

Description

The `send` function accepts the recipient's `dstChainId` and `_dstAddress`. Since the `_dstAddress` is intended for another network, it is transmitted as bytes and its length is checked to make sure that the address is correct. But if the length value for `dstChainId` is not set by the owner, `dstAddress` length will be compared to 0.

```
function send(address _ua, uint64 /*_nonce*/, uint16 _dstChainId, bytes
    calldata _dstAddress, bytes calldata
    _payload, address payable _refundAddress, address _zroPaymentAddress,
     bytes calldata
    _adapterParams) external payable override onlyEndpoint {
        ...
        require(ulnLookup[dstChainId] != bytes32(0), "LayerZero:
    dstChainId does not exist");
        require(dstAddress.length == chainAddressSizeMap[dstChainId],
    "LayerZero: incorrect remote address
    size");
```

Impact

In case of an incorrect message transfer to the empty value of `_dstAddress` and an unspecified value `chainAddressSizeMap` for `dstChainId`, the transaction will not be rejected and the incorrect message will be successfully sent.

Recommendations

Add check that `chainAddressSizeMap` value for `dstChainId` was set.

Remediation

LayerZero Labs acknowledged this finding and implemented a fix in commit `efc2c0cf ffbc10abfe87163e0c3ab29ce4f0d1a4`.

4 Discussion

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

4.1 Notes on in-scope contracts

Patch Review: Changed the nonce index path to include dstAddress.

We were asked to review a minor patch to the LayerZero smart contracts. The patch fixed a potential denial-of-service vulnerability. We did not find any indication of in-the-wild exploitation of the vulnerability. We did not find any issues with the patch.

The patch itself was an update to the UltraLightNode contract. The previous version is ULNV1, and the new version is ULNV2. The changes are summarized below:

Changed the nonce index path to include dstAddress. If nonces are not indexed by dstAddress in addition to srcChainId and srcAddress, a denial-of-service attack is possible. In this scenario, a malicious user application configures a custom Relayer and Oracle. The custom Relayer and Oracle then collude to desynchronize the nonces associated with a certain srcChainId and srcAddress. This change blocks this attack so that only the attacking contract itself is affected.

4.2 Incorrect parsing of data inside `getPacketV2` function when using a zero `_remoteAddressSize` value

Passing a zero value in the `_remoteAddressSize` argument of `validateProof` function leads to incorrect parsing of data inside the `getPacketV2` function.

The proof library should check the `_remoteAddressSize` value to confirm it is nonzero.

Remediation

LayerZero Labs acknowledged this finding and implemented a fix in commit `f1a10e2e bca2b009928a525ff6fe8ca4349bd35f`.

4.3 `_getVerifiedLog` may allow out-of-bounds read via `getItemByIndex`

The `_getVerifiedLog` function in all MPT validation library versions use the unsafe `getItemByIndex` function at least once:

```
log.topicZeroSig = bytes32(it.next().getItemByIndex(0).toUint());
```

We understand that the LayerZero Labs team actively looks for small gas optimizations in safe places. Assuming all other code is safe, this use of the unsafe function presents no risk to the proof validation library. However, we generally recommend developers prioritize safety over optimization as seemingly small, unexploitable potential bugs may be chained with other bugs in ways that lead to a larger impact.

In this case the simple solution is to replace the `getItemByIndex` call with `safeGetItemByIndex`:

```
log.contractAddress = bytes32(it.next().toUint());
log.topicZeroSig = bytes32(it.next().getItemByIndex(0).toUint());
log.topicZeroSig = bytes32(it.next().safeGetItemByIndex(0).toUint());
log.data = it.next().toBytes();
```

Remediation

LayerZero Labs acknowledged this finding and implemented a fix in commit `32e548402066d9c7d37529b23de721b9a0601083`.