

# LayerZero Security Audit

---

*LZMulticall*

Released January 13, 2026

## Performed By

Carter Snay  
James Kahn  
Oliver Willis

csnay@iol.unh.edu  
jkahn@iol.unh.edu  
owillis@iol.unh.edu



**University of  
New Hampshire**  
Interoperability Labs

+1-603-862-0090 | [www.iol.unh.edu](http://www.iol.unh.edu)

# Table of Contents

<b>1</b>	<b>Legal Notice .....</b>	<b>2</b>
<b>2</b>	<b>Executive Summary .....</b>	<b>3</b>
2.1	About LZMulticall	
2.1.1	Review Timeline	
2.1.2	Scope	
<b>3</b>	<b>Findings .....</b>	<b>5</b>
3.1	Findings Summary	
3.2	Detailed Findings	
3.2.1	Info	
3.2.2	Gas	
	<b>Appendix A: Our Methodology .....</b>	<b>10</b>
	Risk Classification	
	Review Phases	
	Phase I: Initial Scoping	
	Phase II: Codebase Review	
	Phase III: Local Testing	
	Proof of Vulnerability	
	<b>Appendix B: The Interoperability Labs .....</b>	<b>13</b>

# 1 Legal Notice

---

The Interoperability Labs Blockchain team makes every effort to identify as many vulnerabilities in the code as possible within the given time period but assumes no responsibility for the findings presented in this document. A security audit by the team does not constitute an endorsement of the underlying business or product. The audit was time-boxed, and the review focused solely on the security aspects of the Solidity implementation of the contracts.

## 2 Executive Summary

---

The UNH Interoperability Labs conducted a security assessment for LayerZero Labs from December 15, 2025 to January 13, 2026. During this assessment, the UNH Interoperability Labs reviewed the LayerZero Labs LZMulticall code for security vulnerabilities, design issues and general weaknesses.

During the assessment, 2 informational findings and a gas optimization finding were identified by the team.

### 2.1 About LZMulticall

LZMulticall acts as a generalized execution router designed to batch multiple arbitrary calls, applying a uniform fee layer to all transactions. It supports both direct execution and signature-authorized execution, allowing off-chain systems and partners to assemble transaction bundles that users can approve via signatures.

TransferDelegate is a helper contract used by LZMulticall specifically for executing ERC20 token transfers that users have pre-approved.

#### 2.1.1 Review Timeline

- **December 15, 2025:** Initial Audit Scope Review
- **December 16, 2025:** Draft report delivered
- **January 7, 2026:** Newest commit merged to codebase and shared with audit team
- **January 9, 2026:** Second draft report delivered
- **January 12, 2026:** Newest commit shared with audit team
- **January 13, 2026:** Third draft and final report delivered

## 2.1.2 Scope

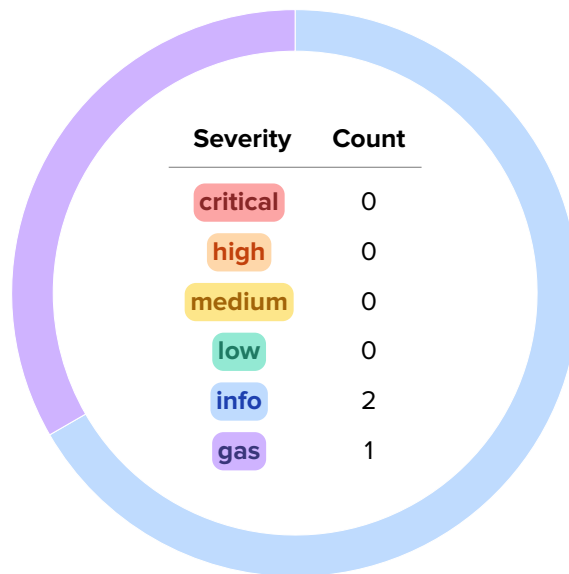
<b>Project Name</b>	LZMulticall		
<b>URL</b>	<a href="https://layerzero.network/">https://layerzero.network/</a>		
<b>Language</b>	Solidity		
<b>Scope</b>	Repo	<a href="https://github.com/LayerZero-Labs/lz-multicall">https://github.com/LayerZero-Labs/lz-multicall</a>	
	Hash	14034a165fe0984797635f799f4f18deb78550f7	December 10, 2025
		1e97a449fc88a916cfaa81c73ae2a440f85ccc31	January 6, 2026
		de6428f2c3d6e090c70de3664e066596fa0c50e6	January 9, 2026

### Files in Scope

```
src/
├── interfaces/
│   ├── ILZMultiCall.sol
│   └── ITransferDelegate.sol
├── LZMultiCall.sol
└── TransferDelegate.sol
```

## 3 Findings

Two informational NatSpec findings and one gas optimization were found.



### 3.1 Findings Summary

ID	Severity	Title	Status
I-01	info	Missing <code>_expiration</code> NatSpec Parameter Description	RESOLVED
I-02	info	<code>LZMultiCall::_handleCall</code> Has Misleading NatSpec Comment	ACKNOWLEDGED
G-01	gas	Upgrade SafeERC20 to use OpenZeppelin Release v5.5	ACKNOWLEDGED

## 3.2 Detailed Findings

### 3.2.1 Info

#### [I-01] Missing `_expiration` NatSpec Parameter Description

Category	Target
NatSpec Cleanup	LZMultiCall.sol

#### Description

Three functions inside of the LZMultiCall.sol contract do not include the `_expiration` parameter in the NatSpec.

#### Recommended Mitigation

The simplest resolution is to make use of the NatSpec function descriptions in the ILZMultiCall.sol interface and use the NatSpec `@inheritdoc` tag. This provides one source of truth for public/external function descriptions, preventing the need to update comments in multiple places.

```

LZMultiCall.sol
diff
50  /**
   -  * @notice Executes multiple calls with a user's signature.
   -  * @dev This enables account abstraction: anyone can submit the transaction (pay gas) on behalf
   -  *    of the user who signed.
   -  * @dev The nonce must match the signer's current nonce to prevent replay attacks.
   -  * @dev If target is `TRANSFER_DELEGATE` for any call, it validates that the `from` address in
   -  *    the transfer call matches the `_signer` address.
   -  * @dev Any ETH, token, or authorization left after this call can be permissionlessly claimed by
   -  *    anyone.
   -  * @param _calls Array of calls to execute
   -  * @param _quoteId Unique identifier for this execution
   -  * @param _signer Address that authorized the calls
   -  * @param _signature EIP-712 signature from the user
51 +  * @inheritdoc ILZMultiCall
52  */
53  function execute(
54      Call[] calldata _calls,
55      bytes32 _quoteId,
56      uint256 _expiration,
57      address _signer,
58      bytes calldata _signature
59  ) public payable virtual {

```

```
LZMultiCall.sol diff
109  /**
    -   * @notice Gets the digest to sign for a given set of calls.
    -   * @dev Useful for off-chain signature generation.
    -   * @param _calls Array of calls to execute
    -   * @param _quoteId Unique identifier for this execution
    -   * @param _signer Address that will sign the calls
    -   * @return digest Digest that should be signed
110 +   * @inheritdoc ILZMultiCall
111  */
112  function getDigestToSign(Call[] calldata _calls, bytes32 _quoteId, uint256 _expiration, address
    ↪ _signer)
    ...
197  /**
    -   * @notice Internal function to get the digest to sign for a given set of calls.
    -   * @param _calls Array of calls to execute
    -   * @param _quoteId Unique identifier for this execution
    -   * @param _nonce Nonce for replay protection
    -   * @return digest Digest that should be signed
198 +   * @inheritdoc ILZMultiCall
199  */
200  function _getDigestToSign(Call[] calldata _calls, bytes32 _quoteId, uint256 _expiration, uint256
    ↪ _nonce)
```

## Resolution

LayerZero has acknowledged this, and has updated the NatSpec in the following commit:

1c97a449fc88a916cfaa81c73ae2a440f85ccc31

## [I-02] LZMultiCall::\_handleCall Has Misleading NatSpec Comment

Category	Target
NatSpec Cleanup	LZMultiCall.sol

### Description

NatSpec param comment in LZMultiCall::\_handleCall function indicates that the parameter's data location is in calldata, whereas the implementation indicates the location is in memory.

### Recommended mitigation

```

LZMultiCall.sol
diff
164  /**
165   * @notice Internal function to forward arbitrary calls.
166   * @param _target Address of the target contract to call
167 -   * @param _data Calldata
167 +   * @param _data Bytes in memory of low level call
168   * @param _value Value to send with the call
169   */
170  function _handleCall(address _target, bytes memory _data, uint256 _value) internal virtual {

```

### Resolution

LayerZero has acknowledged this, but prefers to keep NatSpec unchanged.

### 3.2.2 Gas

#### [G-01] Upgrade SafeERC20 to use OpenZeppelin Release v5.5

Category	Target
Gas-Optimization	LZMultiCall.sol

#### Description

The inclusion of OpenZeppelin's LowLevelCall library from the v5.5 release in the LZMultiCall contract allows for the opportunity to upgrade other potential imports to the same version. With consideration of LayerZero being a multi-chain protocol, special care must be taken when considering the opcodes used in OpenZeppelin's v5.5 release. The SafeERC20 library is one of those imports which do not use unsupported opcodes, and provides lower level gas optimizations through assembly.

#### Recommended mitigation

```

LZMultiCall.sol
diff
4 import {SignatureChecker} from "@openzeppelin/contracts/utils/cryptography/SignatureChecker.sol";
5 import {EIP712} from "@openzeppelin/contracts/utils/cryptography/EIP712.sol";
- import {SafeERC20} from "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
- import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
6 + import {SafeERC20} from "@openzeppelin-v5.5.0/contracts/token/ERC20/utils/SafeERC20.sol";
7 + import {IERC20} from "@openzeppelin-v5.5.0/contracts/token/ERC20/IERC20.sol";
8 import {LowLevelCall} from "@openzeppelin-v5.5.0/contracts/utils/LowLevelCall.sol";

```

#### Resolution

LayerZero has acknowledged this gas optimization.

## Appendix A: Our Methodology

---

The Interoperability Labs audit team follows a comprehensive methodology in ensuring the security and reliability of smart contracts and Web3 protocols. While the specific testing procedures performed vary between the project and protocol, the tooling and manual review process remains the same to ensure thorough analysis has been completed on all items within the defined scope of the audit. Throughout the security review, the audit team maintains communication with the development team, providing feedback on identified vulnerabilities and optimizations. The following sections provide an overview of our systematic audit process and methodology.

## Risk Classification

			Impact →			
Likelihood ↓		Informational	Low	Medium	High	Critical
	Very Unlikely	Info	Low	Low	Medium	Critical
	Unlikely	Info	Low	Low	Medium	Critical
	Possible	Info	Low	Medium	High	Critical
	Likely	Info	Low	Medium	High	Critical
	Very Likely	Low	Medium	High	Critical	Critical

The PricewaterhouseCoopers-style matrix is used to provide a comprehensive risk assessment.

## Review Phases

### Phase I: Initial Scoping

- Independent review of project documentation to understand the business logic of the project.
- Identification of critical components and key areas of focus and possible areas of exploitation.
- Ensure that the project's documentation is accurate, complete, understandable, and aligned with the code.
- Discussion with the development team to clarify objectives, expectations, and known issues.

### Phase II: Codebase Review

- **Static Analysis:** Automated tools to scan for common vulnerabilities with Slither and Aderyn.
- **Manual Review:** In-depth inspection of the code by the audit team to identify issues, including unsafe coding practices from known previous exploits.
- **Function State Machine Diagramming:** Generate a flow diagram illustrating the intended transaction paths, as well as unintended and potentially exploitable paths.

## Phase III: Local Testing

### Methods:

- **Unit Testing:** Validate individual functions for correctness.
- **Integration Testing:** Ensure that different components interact as expected.

### Techniques Used in This Audit:

- **Unit Testing**
- **Manual Review**
- **Function State Machine Diagramming**

## Proof of Vulnerability

**Objective:** Prove how a found exploit can be executed.

### Activities:

- Perform controlled attacks on a local fork of the protocol to show how an exploit can be executed.
- Test edge cases and unexpected scenarios discovered in the diagramming phase.

By following a structured and comprehensive methodology, we aim to provide actionable insights to strengthen the security and reliability of the protocol. This ensures security, resilience, and long-term success for the protocol.

## Appendix B: The Interoperability Labs

The University of New Hampshire Interoperability Labs (UNH-IOL) is the foremost independent testing facility for data networking companies worldwide.

We accelerate the launch of innovative products by providing standards and compliance testing to ensure that devices meet industry standards. Whether it's traditional Ethernet or advanced technologies like 5G, blockchain, and autonomous vehicles, our services provide comprehensive testing for device interoperability, conformance, and certification.

Our state-of-the-art laboratory and 36 years of extensive experience make it a strategic resource for industry startups and Fortune 500 companies needing collaboration, innovation, and standards development to help them shape the future of networking.

Join us and become a part of a community driving the next generation of networking technology. Together, we can shape the future of networking.

<https://www.iol.unh.edu/membership>

University of New Hampshire Interoperability Laboratory

21 Madbury Rd., Ste 100 Durham, NH 03824-4716

+1-603-862-0090 | [www.iol.unh.edu](http://www.iol.unh.edu)

Contact our Blockchain Team: [blockchain@iol.unh.edu](mailto:blockchain@iol.unh.edu)