PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For LayerZero (LZMultiCall)

14 January 2026

paladinsec.co   info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for LayerZero's contracts on the Ethereum network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	LayerZero
URL	https://layerzero.network/
Platform	Ethereum
Language	Solidity
Preliminary Contracts	https://github.com/LayerZero-Labs/lz-multicall/commit/14034a165fe0984797635f799f4f18dcb78550f7
Resolution #1	https://github.com/LayerZero-Labs/lz-multicall/commit/de6428f2c3d6e090c70de3664e066596fa0c50e6

1.2 Contracts Assessed

- `LZMultiCall`
- `TransferDelegate`

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
🔴 High	0	-	-	-
🟠 Medium	0	-	-	-
🟡 Low	2	2	-	-
🟣 Informational	5	-	-	5
Total	**7**	**2**	-	**5**

Classification of Issues

Severity	Description
🔴 **High**	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
🟠 **Medium**	Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
🟡 **Low**	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
🟣 **Informational**	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 LZMultiCall

ID	Severity	Summary	Status
01	LOW	Insufficient `msg.value` validation	✓ RESOLVED
02	LOW	Missing reentrancy guard	✓ RESOLVED
03	INFO	Executor could be problematic in edge cases	ACKNOWLEDGED
04	INFO	Executions targeting delegate contract can leave native currency in `LZMultiCall`	ACKNOWLEDGED
05	INFO	Accidental approvals to `LZMultiCall` could be harmful	ACKNOWLEDGED
06	INFO	`QuoteId` can be used for multiple executions	ACKNOWLEDGED
07	INFO	No way for signers to revoke issued signatures	ACKNOWLEDGED

1.3.2 TransferDelegate

No issues found.

2 Findings

2.1 LZMultiCall

`LZMultiCall` executes batched calls on behalf of users via EIP-712 signatures, enabling gasless transactions where anyone can submit signed operations including validated ERC20 transfers through a linked delegate contract.

2.1.1 Issues & Recommendations

Issue #01	**Insufficient `msg.value` validation**
Severity	● LOW SEVERITY
Description	There is no check that the sum of all `call.value == the msg.value` of the execute call. This can result in leftover native gas tokens in the contract. In `_executeCalls()`, all the values of `call.value` should be summed up, and after the `for` loop, it should enforce that `msg.value ==` sum of all `call.value`. Alternatively, at the end of each execute call, if the LZMulticall has a non-zero native balance, the balance can be transferred out to the caller. Without such validation, anybody can steal the leftover native currency in the contract.
Recommendation	We recommend implementing the fix described above.
Resolution	✓ RESOLVED

Issue #02	**Missing reentrancy guard**
Severity	● LOW SEVERITY
Description	If there is no proper case where the `call.target` should be the LZMultiCall contract itself, it is recommended to add a check that `call.target != address(this)`. Otherwise, consider adding a nonreentrant modifier for executing functions.
Recommendation	We recommend implementing one of the fixes described above.
Resolution	✓ RESOLVED

Issue #03	Executor could be problematic in edge cases
Severity	● INFORMATIONAL
Description	Anyone can call `execute` with the user's signature. If the `execute` function is expected to be called by a contract but a user frontruns it and calls it first using the signer's nonce, it will cause the contract function to revert if try-catch is not used.
Recommendation	Consider allowing the signer to specify who the `msg.sender` of the execute call can be and if such is not provided (`address(0)`) skip the `msg.sender` validation.
Resolution	● ACKNOWLEDGED This is by design. The team added a NatSpec comment to discourage non-deterministic execution based on `tx.origin`.

Issue #04	Executions targeting delegate contract can leave native currency in `LZMultiCall`
Severity	● INFORMATIONAL
Description	Unlike `_handleCall` which passes the `_value` as the `msg.value` to the external call, `_handleTransfer()` does not. However, it does not verify that the `call.value` is 0 which can result in an unnecessary native amount being left in the contract.
Recommendation	We recommend adding a check if the `call.target` is `TRANSFER_DELEGATE` that `call.value` is 0.
Resolution	● ACKNOWLEDGED This is by design. The team implemented the sweep function which mitigates the effect.

Issue #05	Accidental approvals to `LZMultiCall` could be harmful
Severity	● INFORMATIONAL
Description	As token allowances are not intended to be granted to `LZMultiCall`, but TransferDelegate, it is recommended to prevent ERC20's `transferFrom` selector from being a call selector in `_handleCall`. This prevents anyone from misusing any token allowance granted by the user to the multicall contract by mistake.
Recommendation	We recommend implementing the fix described above.
Resolution	● ACKNOWLEDGED The team added a NatSpec comment indicating `LZMultiCall` should never receive any allowance of any kind.

Issue #06	`_quoteId` can be used for multiple executions
Severity	● INFORMATIONAL
Description	There is no validation that the `_quoteId` has not been used before. it can also be frontrun. Off-chain logic could be confused by this if it relies on `_quoteId` for anything.
Recommendation	Consider storing `_quoteId` per signer in storage and checking if it was used before or remove it altogether.
Resolution	● ACKNOWLEDGED This is by design. The team has modified the NatSpec to indicate `_quoteId` is not a unique identifier.

Issue #07	No way for signers to revoke issued signatures
Severity	● INFORMATIONAL
Description	At some point a signer might wish to prevent issued signatures from being executed. They could set the allowance of a certain token to 0 but they will not be able to stop any other executions from happening.
Recommendation	We recommend introducing a function that allows `msg.sender` to increase their nonce by one, which would prevent recently created signatures by them to revert on execution.
Resolution	● ACKNOWLEDGED Calling `execute` can be used to increment the current nonce and invalidate signatures, similarly to how EOA nonces are incremented in Ethereum.

2.2 TransferDelegate

`TransferDelegate` forwards ERC20 `transferFrom` calls exclusively from the `LZMultiCall` contract, acting as the token allowance holder that users approve to enable delegated token transfers.

2.2.1 Issues & Recommendations

No issues found.

PALADIN
BLOCKCHAIN SECURITY