



PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For LayerZero
(Omnichain Governance
Executor)

30 March 2023



paladinsec.co



info@paladinsec.co

Table of Contents

Table of Contents	2
Disclaimer	3
1 Overview	4
1.1 Summary	4
1.2 Contracts Assessed	4
1.3 Findings Summary	5
1.3.1 OmnichainProposalSender	6
1.3.2 OmnichainGovernanceExecutor	6
2 Findings	7
2.1 OmnichainProposalSender	7
2.1.1 Privileged Functions	8
2.1.2 Issues & Recommendations	9
2.2 OmnichainGovernanceExecutor	15
2.2.1 Privileged Functions	16
2.2.2 Issues & Recommendations	17



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 advocacy 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 Omnichain Governance Executor 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/omnichain-governance-executor/tree/7f7f213c149506dfcb4ce4666f261d8a1d6991d6
Final Contracts	

1.2 Contracts Assessed

Name	Contract	Live Code Match
OmnichainPropos		
alSender		
OmnichainGovern		
anceExecutor		

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● High	0	-	-	-
● Medium	1	1	-	-
● Low	3	3	-	-
● Informational	8	7	-	1
Total	12	11	-	1

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 OmnichainProposalSender

ID	Severity	Summary	Status
01	MEDIUM	Griefing: Anyone can potentially steal ETH from the contract owner (timelock) by instating a failed execution	✓ RESOLVED
02	LOW	Allowing anyone to retry might be considered an excessive privilege	✓ RESOLVED
03	LOW	Contract lacks a way to cancel stored transactions	✓ RESOLVED
04	INFO	Upgradeability might make sense given the contract design	✓ RESOLVED
05	INFO	Typographical errors	✓ RESOLVED
06	INFO	Gas optimization	✓ RESOLVED
07	INFO	Lack of indexing for event parameters	✓ RESOLVED

1.3.2 OmnichainGovernanceExecutor

ID	Severity	Summary	Status
08	LOW	Allowing anyone to retry might be considered an excessive privilege	✓ RESOLVED
09	INFO	Setting the owner to itself will eventually brick the executor	✓ RESOLVED
10	INFO	Contract can call its own nonblockingLzReceive function through _executeTransaction	✓ RESOLVED
11	INFO	Lack of validation	ACKNOWLEDGED
12	INFO	The contract might not be a 100% drop-in replacement of a timelock as it does not contain a fallback function	✓ RESOLVED

2 Findings

2.1 OmnichainProposalSender

OmnichainProposalSender is a bridge contract which can be used by governance DAOs to execute proposals on other chains.

For example, let's say the Uniswap DAO has their governance voting and executing logic deployed on Ethereum. OmnichainProposalSender can then be used as a utility by Uniswap to bridge messages from Ethereum to a destination chain of their choice.

The way this works is that the DAO must first deploy and configure an OmnichainProposalSender on the destination chain. The sender can then use the Layer Zero bridge to bridge transactions to the destination chain. The executor will finally execute these messages as soon as it receives them.


The intended setup for this contract with a governor Bravo DAO (eg. Uniswap and Compound) is to have the owner of this sender be the governance timelock. The owner of the timelock is, as always, the governor Bravo.

The sender allows for configuring multiple remote chains, while every message targets exactly one remote chain.

2.1.1 Privileged Functions

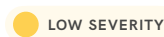
- `execute`
- `setTrustedRemoteAddress`
- `setConfig`
- `setSendVersion`
- `transferOwnership`
- `renounceOwnership`

2.1.2 Issues & Recommendations

Issue #01	Griefing: Anyone can potentially steal ETH from the contract owner (timelock) by instating a failed execution
Severity	 MEDIUM SEVERITY
Location	<u>Line 65</u> <code>payable(tx.origin).transfer(msg.value);</code>
Description	<p>The expected ownership structure is as follows:</p> <p>Governor Bravo -> Timelock -> Sender</p> <p>This means that to call the execute function on the sender, which can only be called by the owner, a timelock transaction that does this needs to be queued by the governor.</p> <p>However, once the timelock has expired and the transaction can be executed, it can be executed by anyone.</p> <p>Due to the way a Compound timelock works, there is no strict requirement for the executor to provide the Ethereum value for the transaction. In fact, the timelock assumes that the ETH will come straight from the balance of the timelock.</p> <p>This means that once a transaction becomes executable, a malicious party might attempt to gas-grief the try-catch to cause it to go to the catch branch and to receive all the ETH which was supposedly going to be paid.</p> <p>Even if gas-griefing is not possible, transactions might still be badly configured, allowing malicious parties to front run the catch branch and get a refund straight from the timelock's ETH balance.</p> <p>Even worse, Layer Zero transactions are always configured with excessive gas and almost always send some gas tokens back to the refund address. This contract has been configured with <code>tx.origin</code> as the refund address as well, which means that whoever can execute the transaction (likely an MEV team), gets whatever refund is provided at all times in the documented ownership setup.</p>
Recommendation	Consider doing the gas refund to the <code>msg.sender</code> .

Resolution

The client has indicated that the refund should occur as incentive for execution. This means that no changes were made and that protocols that implement this should take their time to understand this. They might lose ETH as a result of it, depending on the gas allowance they initially set.

Issue #02**Allowing anyone to retry might be considered an excessive privilege****Severity****Description**

Whenever a transaction fails to be sent to Layer Zero, it is stored in a queue to allow anyone to retry this transaction. This is done to avoid the annoyance of having to go through the timelock again simply because insufficient gas was configured by the timelock transaction.


When cached, anyone can retry that transaction themselves. We do not see any reason why the attack surface should be widened here as the governance could simply keep track of an allowlist of EOAs which are permitted to retry transactions.

Recommendation

Consider reducing the attack surface by only allowing retries by a set of addresses defined in a configurable allowlist.

Resolution

The client has indicated that this is desired. No changes were made.

Issue #03**Contract lacks a way to cancel stored transactions****Severity** LOW SEVERITY**Description**

Whenever a transaction fails to be sent to LayerZero, it is stored in a queue to allow anyone to retry this transaction. This is done to avoid the annoyance of having to go through the timelock again simply because insufficient gas was configured by the timelock transaction.

However, once the transaction has been cached, the governance might no longer be interested in executing it. Leaving it open to be executed is therefore a governance risk and might prove an annoyance in certain unforeseeable edge cases.

Recommendation

Consider adding a remove function. Consider adding an allowlist to who can retry executions.

It might make sense to use an enumerable map for the storedExecutionHashes to allow easy iteration over the presently stored hashes. This is up to the client of course as events suffice for this practice as well.

Resolution RESOLVED

The client has indicated this is not a business requirement they are looking to pursue. No changes were made.

Severity

 INFORMATIONAL

Description

OmnichainProposalSender is not upgradeable. This is typically applauded as upgradeability typically means straying away from decentralization.

However, in this case, the contract owner, e.g. the timelock, can already pretty much execute anything on the executor. We are under the impression that making the contract upgradeable with the owner (the timelock) as the proxy admin would not be a significant privilege escalation and might make it easier to incorporate new features into the sender over time.

Recommendation

Consider whether contract upgradeability is a privilege escalation if the admin is the contract owner. If not, consider whether it is beneficial to have this contract as an upgradeable proxy.

Typical privilege escalations are if the contract ever has open approvals (eg. an ERC20 token approval). Upgradeability would allow for inadvertently draining these and is in our opinion not acceptable.

Do not forget to use the upgradeable OpenZeppelin dependencies in this case.



It should be noted that this is purely an informational issue and it will be resolved in all cases, given whatever the client prefers.



Resolution

 RESOLVED

The client has indicated that the sender does not have any longevity requirement and can simply be redeployed if changes are desired. It should be noted that sometimes upgrades are desired to fix stuff like taking out stuck tokens. This would not be possible in the current iteration. No changes were made.

Issue #05	Typographical errors
Severity	INFORMATIONAL
Description	<p>We have consolidated the typographical issues into a single issue to keep the report brief and readable.</p> <p><u>Line 36</u></p> <pre>constructor(address _lzEndpoint) {</pre> <p>This can be provided as ILayerZeroEndpoint to make the type more explicit.</p> <p><u>Line 42</u></p> <pre>/// @dev The estimated fees the are the min required</pre> <p>This sentence is gramatically incorrect: "the are".</p> <p><u>Line 117</u></p> <pre>/// @param chainId The LayerZero chainId for the pending config change</pre> <p>This parameter description has been incorrectly copied from the setter into the getter. It is not accurate for the getter.</p>
Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED <p>All typographical errors have been fixed.</p>

Issue #06 Gas optimization	
Severity	 INFORMATIONAL
Location	<u>Line 69</u> emit StorePayload(lastStoredPayloadNonce, remoteChainId, payload, adapterParams, msg.value, reason);
Description	The lastStoredPayloadNonce is unnecessarily fetched from storage here. Consider caching it above when incremented.
Recommendation	Consider implementing the gas optimization mentioned above.
Resolution	 RESOLVED The recommendation was implemented.

Issue #07 Lack of indexing for event parameters	
Severity	 INFORMATIONAL
Location	<u>Lines 31 and 34</u> event ClearPayload(uint64 nonce, bytes32 executionHash); event StorePayload(uint64 nonce, uint16 remoteChainId, bytes payload, bytes adapterParams, uint value, bytes reason);
Description	Essential identifying parameters within events should be marked as indexed. This allows for off-chain components to filter events only including these values.
Recommendation	Add indices to the key variables within the events you might want to filter on.
Resolution	 RESOLVED

2.2 OmnichainGovernanceExecutor

OmnichainGovernanceExecutor allows the execution of proposals from the sender. It is the receipt contracts from messages bridged over LayerZero from the sender.

It should be noted that anyone can retry messages at multiple layers of the executor flow. This means that the governance should be exceptionally careful executing contracts which use `tx.origin`.

It should be noted that the underlying LzApp and NonblockingLzApp contracts that this contract relies on are excluded from the explicit scope. This means that any potential vulnerabilities in these contracts might not be covered within this audit.





2.2.1 Privileged Functions

- `lzReceive [layer zero endpoint]`
- `setConfig`
- `setSendVersion`
- `setReceiveVersion`
- `forceResumeReceive`
- `setTrustedRemote`
- `setTrustedRemoteAddress`
- `setPrecrime`
- `setMinDstGas`
- `transferOwnership`
- `renounceOwnership`



2.2.2 Issues & Recommendations

Issue #08	Allowing anyone to retry might be considered an excessive privilege
Severity	 LOW SEVERITY
Description	<p>Whenever a transaction fails execute, it is stored in a queue to allow anyone to retry this transaction. This is done to allow a smooth retry if for example insufficient gas was provided.</p> <p>When cached, anyone can retry that transaction themselves. We do not see any reason why the attack surface should be widened here as the governance could simply keep track of an allowlist of EOAs which are permitted to retry transactions.</p>
Recommendation	Consider reducing the attack surface by only allowing retries by a set of addresses defined in a configurable allowlist.
Resolution	 RESOLVED <p>The client has indicated they want anyone to be able to call this. No changes were made and we recommend the users to be careful regarding issues such as <code>tx.origin</code> side-effects.</p>

Description

EDIT: The LayerZero team has indicated that there is in fact a fallback retry function within the endpoint which does not need to be called by the app. The description below will therefore not brick the executor and this issue's severity has been modified from high to informational.

The executor documentation indicates a plan to eventually let the executor govern itself. E.g. this would allow for the executor to update its configuration, but only by going through the mainnet governor, which is of course ideal.

Line 8

/// @dev The owner of this contract controls LayerZero configuration. When used in production the owner should be set to a Timelock or this contract itself

Although this is indeed ideal, there is a huge caveat. In certain instances, a message can still get stuck and block the message queue at the Layer Zero level. To retry this transaction, the app itself needs to make such a request to Layer Zero's endpoint.

However, until this occurs, no further transactions can be executed. Which means that if the owner is the contract itself, it can only execute transactions by going through the message flow, which is... halted! The contract is at that point inadvertently bricked.

While this may sound like an edge-case given that a non-blocking Layer Zero app is used, within the current version of this app, it is actually quite easy to break the non-blocking behavior:

Lines 25-30

```
bool success, bytes memory reason) =
address(this).excessivelySafeCall(gasleft(), 150,
abi.encodeWithSelector(this.nonblockingLzReceive.selector,
_srcChainId, _srcAddress, _nonce, _payload));
// try-catch all errors/exceptions
if (!success) {
    failedMessages[_srcChainId][_srcAddress][_nonce] =
keccak256(_payload);
    emit MessageFailed(_srcChainId, _srcAddress, _nonce,
_payload, reason);
}
```

As seen above, a storage store occurs after the transaction is executed. Such a store call takes about 20k gas. If not enough gas is available at this point, either by insufficient gas having been sent to cover it or by the underlying protocol doing say an infinite loop, this store will not occur.

This will cause the transaction to be cached for retrieval not at the app layer but at the layer zero layer. This bricks the non-blocking behavior the non-blocking app is supposed to have.



Recommendation Consider updating the `excessivelySafeCall` so that it always leaves enough gas for the remainder of the function to execute. Consider ensuring that enough gas is always provided to execute that remainder.

Consider not relying on this to always function, instead, allow some other trusted party to “circuit break” and resume a transaction without being the owner.

Resolution



The client has set aside 30k gas which is not sent to the safe call. They have also explained that there is a retry function in emergencies on the endpoint which can be called by anyone.

Issue #10	Contract can call its own nonblockingLzReceive function through _executeTransaction
Severity	 INFORMATIONAL
Description	<p>The nonblockingLzReceive function is public and can only be called by the contract itself. The way this is supposed to work is that whenever the contract receives a message, it will call itself to actually execute the proposal. This allows the contract to fail gracefully.</p> <p>However, there might be a misconfiguration possible where the actual proposal to execute is just calling the nonblockingLzReceive function. This would be permitted.</p> <p>Although we see no concern with such a flow, it might make sense to explicitly prevent this to reduce any attack surface.</p>
Recommendation	Consider whether this is any concern. Consider adding a reentrancy guard to the _executeTransaction function to resolve this nicely.
Resolution	 RESOLVED
	A reentrancy guard has been added as was recommended.



Issue #11	Lack of validation
Severity	<div data-bbox="459 206 486 235"></div> INFORMATIONAL
Location	<p data-bbox="451 286 564 315"><u>Line 20</u></p> <pre data-bbox="451 331 1366 495">(address[] memory targets, uint[] memory values, string[] memory signatures, bytes[] memory calldatas) = abi.decode(_payload, (address[], uint[], string[], bytes[]));</pre>
Description	<p data-bbox="451 537 1398 759">The contract contains functions with parameters which are not properly validated. Having unvalidated parameters could allow the governance or users to provide variable values which are unexpected and incorrect. This could cause side-effects or worse exploits in other parts of the codebase.</p>
Recommendation	<p data-bbox="451 801 1347 831">Consider validating the lengths of the above arrays to be equal.</p>
Resolution	<div data-bbox="459 873 486 902"></div> ACKNOWLEDGED



Issue #12

The contract might not be a 100% drop-in replacement of a timelock as it does not contain a fallback function

Severity

 INFORMATIONAL

Description

The Uniswap/Compound Timelock contains a fallback function, which means that if any contract calls a random function on the Timelock, this Timelock will not revert. However, the executor does not contain a fallback function, which means it can only accept specific function selectors, and normal gas token transfers.

If any team assumes that the executor will work exactly like their typical timelock, they might be surprised that suddenly something does not work anymore because of the lack of a fallback function.

Recommendation

Be mindful of this behavior. If desired, a fallback function can be added but the team should be careful with the Layer Zero stack as perhaps in the future the stack might want to use different function signatures which would then not revert due to the fallback.

Resolution

 RESOLVED

The client has indicated they are fine with this and desire just having a receive function and no fallback.





PALADIN
BLOCKCHAIN SECURITY