



PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For LayerZero (MessageLib)

15 December 2023



paladinsec.co



info@paladinsec.co

Table of Contents

Table of Contents	2
Disclaimer	4
1 Overview	5
1.1 Summary	5
1.2 Contracts Assessed	7
1.3 Findings Summary	9
1.3.1 UltraLightNode302 [ReceiveULN302 and SendULN302]	10
1.3.2 ULNBase [SendULNBase and ReceiveULNBase]	10
1.3.3 MessageLibBase [and SendLibBase]	10
1.3.4 MessageLibBaseE2 [SendLibBaseE2 and ReceiveLibBaseE2]	11
1.3.5 UlnConfig	11
1.3.6 OutboundConfig	11
1.3.7 UltraLightNode301 [ReceiveULN301 and SendULN301]	11
1.3.8 MessageLibBaseE1 [SendLibBaseE1 and ReceiveLibBaseE1]	12
1.3.9 TreasuryFeeHandler	12
1.3.10 UlnOptions	12
1.3.11 VerifierOptions	13
1.3.12 AddressSizeConfig	13
2 Findings	14
2.1 UltraLightNode302 [ReceiveULN302 and SendULN302]	14
2.1.1 Privileged Functions	15
2.1.2 Issues & Recommendations	17
2.2 ULNBase [SendULNBase and ReceiveULNBase]	22
2.2.1 Privileged Functions	23
2.2.2 Issues & Recommendations	24
2.3 MessageLibBase	27
2.3.1 Privileged Functions	27

2.3.2 Issues & Recommendations	28
2.4 MessageLibBaseE2	31
2.4.1 Privileged Functions	32
2.4.2 Issues & Recommendations	33
2.5 UlnConfig	37
2.5.1 Issues & Recommendations	39
2.6 OutboundConfig	41
2.6.1 Issues & Recommendations	42
2.7 UltraLightNode301	44
2.7.1 Privileged Functions	45
2.7.2 Issues & Recommendations	46
2.8 MessageLibBaseE1	48
2.8.1 Privileged Functions	49
2.8.2 Issues & Recommendations	50
2.9 TreasuryFeeHandler	53
2.9.1 Issues & Recommendations	54
2.10 UlnOptions	56
2.10.1 Issues & Recommendations	57
2.11 VerifierOptions	59
2.11.1 Issues & Recommendations	60
2.12 AddressSizeConfig	61
2.12.1 Privileged Functions	61
2.12.2 Issues & Recommendations	61



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 MessageLib 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, Arbitrum, Astar, Aurora, Avalanche, Base, BSC, Canto, Celo, Coredao, Dexalot, DOS, Eon, Fantom, Fuse, Gnosis, Harmony, Hubble, Kava, Klaytn, Loot, Manta, Mantle, Merit Circle, Meter, Metis, Moonbeam, Moonriver, Nova, OKX, OPBNB, Optimism, Orderly, Polygon, Scroll, Shimmer, Telos, Tenet, ZK Consensus, ZK Polygon, Zora
Language	Solidity
Preliminary Contracts	https://github.com/LayerZero-Labs/monorepo/tree/bddcada8ab86e1708ac57f960b58a9fbaad1759c/packages/layerzero-v2/evm/messagelib/contracts
Resolution #1	https://github.com/LayerZero-Labs/monorepo/tree/97a0bbbc8666fedb5aa3f14dba806cf86b9373c1/packages/layerzero-v2/evm/messagelib/contracts
Resolution #2	https://github.com/LayerZero-Labs/monorepo/tree/faf36d0862245705eb51aac4ad65cc442d792ad9/packages/layerzero-v2/evm/messagelib/contracts
Resolution #3	https://github.com/LayerZero-Labs/monorepo/tree/6ef8a4b598e6e38b634d1a9d6830d2f31d997344/packages/layerzero-v2/evm/messagelib/contracts
Resolution #4	https://github.com/LayerZero-Labs/LayerZero-v2/tree/ccfd0d38f83ca8103b14ab9ca77f32e0419510ff/messagelib/contracts
Resolution #5	https://github.com/LayerZero-Labs/LayerZero-v2/tree/4b2985921af42a778d26a48c9dee7b9644812cbd/messagelib/contracts

The goal of this audit is to validate the integrity of the default message library contracts used within LayerZero. It should be noted that these message libraries are in no way mandatory to be used by apps and that apps who care about decentralization might want to lock down the configurations: both the message lib addresses (both for inbound and outbound) and its configurations should be snapshotted.

Readers should note that the scope of this audit contains many of the important contracts for the message library stack, but not all of them. No statements or attempts to make validations over the out-of-scope contracts or the interactions with them can be made as a result. This applies to all off-chain components as well. There are many ways in which the off-chain components might malfunction: they might relay erroneous messages or not respect certain configurations.

Paladin has only audited the on-chain components, meaning that we are unable to make any probabilistic statements about the safety of these off-chain components. We recommend the client to be extremely careful with developing these and be defensive in their security: more validations for them are better than less. No message should ever be marked as verified if it did not occur on a source chain and availability should be set as a goal as some applications might get exploited if message delivery turns unavailable (e.g. applications who assume availability of the bridge).

1.2 Contracts Assessed

Name	Contract
MessageLibBaseE1	Due to the large number of chains, please refer to this document for the addresses and code match status: https://docs.google.com/spreadsheets/d/1YxZIE8q2gwNL-7GLJ5GBQ-7w6a6X9wF1cjUyQ6bhPf4/edit#gid=0
TreasuryFeeHandler	Dependency
ReceiveULN301	Due to the large number of chains, please refer to this document for the addresses and code match status: https://docs.google.com/spreadsheets/d/1YxZIE8q2gwNL-7GLJ5GBQ-7w6a6X9wF1cjUyQ6bhPf4/edit#gid=0
ReceiveULN302	Due to the large number of chains, please refer to this document for the addresses and code match status: https://docs.google.com/spreadsheets/d/1YxZIE8q2gwNL-7GLJ5GBQ-7w6a6X9wF1cjUyQ6bhPf4/edit#gid=0
SendULN301	Due to the large number of chains, please refer to this document for the addresses and code match status: https://docs.google.com/spreadsheets/d/1YxZIE8q2gwNL-7GLJ5GBQ-7w6a6X9wF1cjUyQ6bhPf4/edit#gid=0
SendULN302	Due to the large number of chains, please refer to this document for the addresses and code match status: https://docs.google.com/spreadsheets/d/1YxZIE8q2gwNL-7GLJ5GBQ-7w6a6X9wF1cjUyQ6bhPf4/edit#gid=0
SendUlnBase	Dependency
ReceiveUlnBase	Dependency
UlnConfig	Removed
MessageLibBase	Dependency
SendLibBase	Dependency
SendLibBaseE2	Dependency
ReceiveLibBaseE2	Dependency
OutboundConfig	Dependency
AddressSizeConfig	Dependency



1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● High	1	1	-	-
● Medium	5	1	1	3
● Low	5	2	-	3
● Informational	18	13	1	4
Total	29	17	2	10

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 UltraLightNode302 [ReceiveULN302 and SendULN302]

ID	Severity	Summary	Status
01	MEDIUM	Configurational mismatch can cause messages to never get delivered	ACKNOWLEDGED
02	MEDIUM	Using identical configuration for both sending and receiving is potentially limiting for asymmetric applications	✓ RESOLVED
03	LOW	Message lib lacks direct on-chain incentive for executors and verifiers to actually do their part, as the fee is collected at the source chain when a transaction is sent out	ACKNOWLEDGED
04	INFO	Typographical issues	✓ RESOLVED

1.3.2 ULNBase [SendULNBase and ReceiveULNBase]

ID	Severity	Summary	Status
05	MEDIUM	Verifier options might get assigned to wrong verifiers if the verifier list is re-configured while messages are in-flight	ACKNOWLEDGED
06	INFO	Gas optimization	ACKNOWLEDGED
07	INFO	Typographical issue	ACKNOWLEDGED

1.3.3 MessageLibBase [and SendLibBase]

ID	Severity	Summary	Status
08	MEDIUM	Decentralization flaw: Contract owner can fully brick all outbound messages of all applications using the library by setting a bad treasury	PARTIAL
09	LOW	Availability flaw: A single optional verifier can brick all sending for an application, even if it is not a mandatory verifier	ACKNOWLEDGED
10	INFO	Gas optimization	✓ RESOLVED
11	INFO	Typographical issues	ACKNOWLEDGED

1.3.4 MessageLibBaseE2 [SendLibBaseE2 and ReceiveLibBaseE2]

ID	Severity	Summary	Status
12	LOW	Lack of on-chain incentive for execution	ACKNOWLEDGED
13	INFO	Gas optimizations	✓ RESOLVED
14	INFO	Typographical issues	✓ RESOLVED

1.3.5 UInConfig

ID	Severity	Summary	Status
15	INFO	Typographical issues	✓ RESOLVED
16	INFO	Gas optimizations	✓ RESOLVED

1.3.6 OutboundConfig

ID	Severity	Summary	Status
17	INFO	Typographical issues	✓ RESOLVED
18	INFO	Gas optimizations	✓ RESOLVED

1.3.7 UltraLightNode301 [ReceiveULN301 and SendULN301]

ID	Severity	Summary	Status
19	HIGH	The deliver function can be frontrun to prevent normal execution through an insufficient execution gas amount	✓ RESOLVED
20	INFO	Typographical issues	✓ RESOLVED

1.3.8 MessageLibBaseE1 [SendLibBaseE1 and ReceiveLibBaseE1]

ID	Severity	Summary	Status
21	MEDIUM	Decentralization flaw: Contract owner can fully brick all outbound messages of all applications using the library by setting a bad treasury or layerZeroToken	ACKNOWLEDGED
22	INFO	Typographical issues	✓ RESOLVED

1.3.9 TreasuryFeeHandler

ID	Severity	Summary	Status
23	LOW	Governance risk: Apps must make sure to not give infinite allowance to this contract as this could result in the allowance being drained if the LayerZero governance is compromised	✓ RESOLVED
24	INFO	Typographical issues	PARTIAL

1.3.10 UlnOptions

ID	Severity	Summary	Status
25	LOW	decode does not handle a zero _options length	✓ RESOLVED
26	INFO	Typographical issues	✓ RESOLVED
27	INFO	Gas optimizations	✓ RESOLVED

1.3.11 VerifierOptions

ID	Severity	Summary	Status
28	INFO	Gas optimizations	✓ RESOLVED
29	INFO	Typographical issues	ACKNOWLEDGED

1.3.12 AddressSizeConfig

No issues found.

2 Findings

2.1 UltraLightNode302 [ReceiveULN302 and SendULN302]

UltraLightNode302 is used as the primary default message library within the LayerZero infrastructure. This means that for any app using the default configuration, it will be the contract delivering messages to the endpoint and sending out messages to the off-chain executors.

For the delivery flow, the primary function to be used is `deliver`. When `deliver` is called, a packet header and payload hash must be provided, and these two form the identity of that message. To be deliverable, enough verifiers must have verified this message. The number of verifiers that must verify the message and which verifiers are absolutely required to verify the message can be completely configured by the receiving app. That being said, a default configuration is again applicable if it is not overwritten. The default configuration can be changed at any time by the LayerZero governance which will cause any apps that did not snapshot the configuration (using `snapshotConfig`) to automatically update. If apps care about decentralization, it is a very good idea to configure their own configuration which can be done through interacting with the endpoint (which will then call this ULN to ask it to update the app's config).

For the sending flow, the contract relies heavily on the underlying contracts and uses `ULNBase` to calculate the fees for the signers while relying on `MessageLibBase` and `MessageLibBaseE2` to get the fees for the other components (treasury and executor). Significant logic is therefore executed to calculate the exact fee on-chain that must be paid for the message to be transmitted via the configured verifiers and executor.

UltraLightNode302 inherits all logic defined within ULNBase, MessageLibBase and MessageLibBaseE2. Any issues marked within these subcontracts apply here as well, though it should be noted that they will not be repeated here to keep the report brief. It also uses the external UlnConfig (and OutboundConfig) contract, also within this report, to manage the configuration of applications. It is unclear why an external contract was chosen as this contract is exclusively used for individual ULNs.

UPDATE 1: After the initial resolution round, changes to this contract were made. This contract was split up into ReceiveUln302 and SendUln302.

- ReceiveUln302: deliver is renamed to commitVerification, _deliverable is removed in favor of split out between _assertHeader and the now non-view _checkVerifiableAndEmpty. deliverable is renamed to verifiable and uses similar functions to the above, except for the view alternative of the check verify. getConfig and setConfig were redesigned with new configuration systems.
- SendUln302: Most of the logic was directly copied from the original codebase. getConfig and setConfig were redesigned with new configuration systems.

UPDATE 2: The receive contract moved to the new name for the verify function. Its interface was also renamed.

UPDATE 3: The _payloadHash argument was added to _verifiable from receiveLibbaseE2.

2.1.1 Privileged Functions

- setDefaultConfig [owner]
- transferOwnership [owner]
- renounceOwnership [owner]



2.1.2 Issues & Recommendations

Issue #01	Configurational mismatch can cause messages to never get delivered
Severity	 MEDIUM SEVERITY
Description	<p>uInConfig defines the verifiers and executors used by an application. This configuration needs to be stored on both the sending chain (to call these contracts to ask them the exact fee to pay) and the recipient chain (to validate the message against these addresses).</p> <p>For sending, the config is fetched from (sender, dstEid):</p> <pre>(UInConfigStruct memory config, OutboundConfigStruct memory outboundConfig) = uInConfig.getUInAndOutboundConfig(_sender, _dstEid);</pre> <p>For receipt, the config is fetched from (receiver, srcEid):</p> <pre>UInConfigStruct memory config = uInConfig.getUInConfig(_packetHeader.receiverB20(), _packetHeader.srcEid());</pre> <p>This means that the sender must have an identically matching configuration to the recipient. Though the verifiers may have different addresses, they must refer to identical verifiers (eg. the same entity deploying on each chain with a different address).</p> <p>This specifically poses a challenge if this configuration is ever adjusted. If the configuration is adjusted, it must be adjusted first at one of the chains and then at the other. This means that messages might get sent with a configuration which is no longer present on the destination chain.</p> <p>If this configuration refers to old verifiers, these old verifiers might get a fee collected while the new verifiers may not want to sign this transaction as they never received that fee. The new message might never get signed while the old verifiers are unable to publish it either. This is because the on-chain receipt logic validates that only the latest signers are allowed to sign the message.</p>

Recommendation Consider whether it makes sense to redesign the system in a future version where the configuration only lives on the destination chain to avoid this duplication of data.

Alternatively, consider strongly recommending (“enforcing”) an off-chain protocol where the source chain must always upgrade first. Note that this is impossible as long as the configurations are symmetrical. Moving to asymmetrical configurations would be a necessary step to take as well for this to work.

If the source chain is always updated first, this means that messages will be sent out with verifiers which are not yet configured on the recipient. Ideally, with the correct off-chain verifier logic, these verifiers will submit these messages to the destination as soon as that destination updates their configuration.

Doing the upgrade the other way around would require the new signers to sign transactions they never got fees for. This could be a protocol as well but of course has misaligned incentives.

Resolution

 ACKNOWLEDGED

The client has indicated they are aware of this and it is designed that way. Applications must use the same entities across all chain deployments as their verifiers.



Issue #02**Using identical configuration for both sending and receiving is potentially limiting for asymmetric applications****Severity** MEDIUM SEVERITY**Description**

The `uInConfig` defines the verifiers and executors used by an application. This configuration needs to be stored on both the sending chain (to call these contracts to ask them the exact fee to pay) and the recipient chain (to validate the message against these addresses).

For sending, the config is fetched from `(sender, dstEid)`:

```
(UInConfigStruct memory config, OutboundConfigStruct memory outboundConfig) = uInConfig.getUInAndOutboundConfig(_sender, _dstEid);
```

For receipt, the config is fetched from `(receiver, srcEid)`:

```
UInConfigStruct memory config =  
uInConfig.getUInConfig(_packetHeader.receiverB20(),  
_packetHeader.srcEid());
```

For example when you have app A on chain 1 communication with app B on chain 2, app A will use the following configurations:

For sending: `uIn config (A, 2)`

For receiving: `uIn config (A, 2)`

This means that the same configuration is re-used for both sending and receiving. This can be extremely problematic when the application wishes to use a different set of verifiers for the sending as for the receiving.

Recommendation

Consider whether it makes sense to split up the configuration into an outbound and inbound portion. Since this was already done to some extent, it may make sense to simply add verifiers and optional verifiers logic to the outbound configuration instead.

Resolution RESOLVED

The send and receipt message libraries have been split up, resolving this for us.

Issue #03

Message lib lacks direct on-chain incentive for executors and verifiers to actually do their part, as the fee is collected at the source chain when a transaction is sent out

Severity

 LOW SEVERITY

Description

When an application sends out a transaction, it pays all the relevant verifiers and the executor up-front. This means that regardless of whether these parties do their part, they will have collected the fee.

Typically, this will not pose a problem as applications would quickly move to different verifiers and executors if any of them misbehave. However, there might be incentive for example an executor to “exit-scam” at certain points in time.

This could specifically occur if an airdrop of let’s say 100 ETH was sent with a transaction. Typically, the executors are used to airdrop small amounts to the recipient chain as part of the execution logic, so they levy an equivalent fee on the source chain to make up for that cost.

If some airdrop like this would suddenly transfer a large sum of wealth, the executor would be able to collect this large fee up-front then give up and never execute that message at the destination. This would mean they fully collect the fee which was intended to be paid back to the app in the form of the airdrop.

Note that this airdrop feature is just an example — this misalignment might be abused through other means as well, depending on what features are entrusted to the executors and validators.

Recommendation

Consider redesigning the system to only grant the fees at the receipt chain.

Resolution

 ACKNOWLEDGED

Applications should select their providers based on reliability metrics.

Issue #04	Typographical issues
Severity	<div data-bbox="456 203 485 232"></div> INFORMATIONAL
Description	<p data-bbox="448 286 485 315"><u>L9</u></p> <pre data-bbox="448 331 1126 409">import {OutboundConfigStruct} from "../../OutboundConfig.sol";</pre> <p data-bbox="448 448 938 483">This import appears to be unused.</p> <p data-bbox="448 533 520 546">---</p> <p data-bbox="448 600 1406 728">It appears that supportsInterface might return false for some 302 specific functions. It may make sense to overwrite this function to add the 302 interface on top of it.</p> <p data-bbox="448 777 520 790">---</p> <p data-bbox="448 844 1362 1066">The verify function should be moved in the external functions group as it is currently in the internal group which can be misleading. Throughout the codebase, this gets done through an overwritable abstract internal function, overwritten within UltraLightNode302.</p>
Recommendation	Consider fixing the typographical issues.
Resolution	<div data-bbox="456 1178 485 1207"></div> RESOLVED



2.2 ULNBase [SendULNBase and ReceiveULNBase]

ULNBase is a component of the default message lib, `UltraLightNode302`. Its primary responsibility is to manage the logic to validate verifier signatures.

Apart from that, it also provides the storage variable for the `UlnConfig` contract address that is used throughout the message lib. It finally defines the internal function to decode provided sending options into executor and verifier options.

UPDATE 1: After the first resolution round, intrusive changes were made to this contract. ULNBase has been redesigned from the ground-up and Paladin has validated this new iteration. The team also added `SendULNBase` and `ReceiveULNBase` with specific functionality to the split-up libraries.

- `SendULNBase`: Several functions were renamed but remained nearly identical from a logical perspective. Several functions were modified to use the new configuration system.
- `ReceiveULNBase`: Several functions were renamed and redesigned. Confirmations are now deleted as a gas optimization. We note that if a DVN is added as both an optional and required DVN, this will cause issues. The client solved this in a later update.

UPDATE 2: After the second round, the new version included some cleaned up configuration parameters and overall has cleaner style now.

- `ReceiveULNBase`: Verification logic has been hardened to support verification with 0 confirmations. Previously, due to 0 being used as a magic value for "not confirmed", this would cause serious issues. We pointed out a small issue where a memory value gets mutated that got fixed in a later update.

UPDATE 3: The small issue in `ReceiveUlnBase` above was resolved.

2.2.1 Privileged Functions

- `setDefaultConfig [ owner ]`
- `transferOwnership [ owner ]`
- `renounceOwnership [ owner ]`

2.2.2 Issues & Recommendations

Issue #05	Verifier options might get assigned to wrong verifiers if the verifier list is re-configured while messages are in-flight
Severity	 MEDIUM SEVERITY
Location	<pre>L91, 92 100 and 101 (and _assignJobToVerifiers) uint8 verifiersLength = _config.verifierCount + _config.optionalVerifierCount; for (uint i = 0; i < verifiersLength;) { [...] for (uint j = 0; j < _verifierIds.length; ++j) { if (_verifierIds[j] == i) {</pre>
Description	<p>When sending transactions to LayerZero, applications are permitted to pass in an array of options which will be sent to the verifiers when they are called to calculate their fees. Such parameters could include the desire to execute precrimes etc., which costs an additional fee. During the actual consumption of these options, they need to be assigned to the correct verifier as an app might not provide an option for every verifier. This is done by letting the options be identified by the id of this verifier.</p> <p>However, the issue lies in the fact that this id changes over time as it simply refers to the index in the [verifiers..., optionalVerifiers] array which changes over time as the app reconfigures this array.</p> <p>Transactions might be sent to the application with options assuming a specific structure of this array, but by the time these transactions arrive, the array order might have been changed completely.</p> <p>If this happens, options will be assigned to the wrong verifiers. These verifiers might not know how to deal with them or simply ignore them. Critical expected behavior might therefore simply not execute silently.</p>
Recommendation	Consider using ids which cannot change — e.g. the verifier addresses.

Resolution

 ACKNOWLEDGED

The client is already aware of this but finds alternative solutions less optimal than what has currently been done and is therefore choosing to retain the current setup.

Issue #06**Gas optimization****Severity**

 INFORMATIONAL

Description

The logic which is used to assign options to verifiers executes in an quadratically expensive complexity if the option size increases: $O(n^2)$. This is extremely costly if for example 50 options need to be provided.

This complexity can be reduced to $O(n)$ by requiring the options to be sorted. In this case, no inner for-loop would be required. We have shared a raw template of how this would look to the client and have explained to them that even if enforced options need to be incorporated, these can be sorted in without too much gas cost as long as they are ordered themselves.

Recommendation

Consider implementing the gas optimization.

Resolution

 ACKNOWLEDGED

Issue #07	Typographical issue
Severity	 INFORMATIONAL
Location	<u>L19-20</u> event PayloadSigned(address verifier, bytes header, uint confirmations, bytes32 proofHash); event VerifierFeePaid(address[] verifiers, address[] optionalVerifiers, uint[] fees);
Description	It may make sense to index the verifier address in the first event. Likewise for the second event, but indexing an array may not make sense, so it would require breaking up the event which might be expensive.
Recommendation	Consider fixing the typographical issue.
Resolution	 ACKNOWLEDGED



2.3 MessageLibBase

MessageLibBase defines various fee calculating utilities for the on-chain sending fee calculations. It provides the internal logic to calculate the fees which must be paid for the validators, executor and treasury. It also defines the storage for these fees to accumulate in, which can then be withdrawn within the MessageLibBaseE2 withdrawFee function.

This contract is used as a dependency for UltraLightNode302, alongside its parent MessageLibBaseE2 and ULNBase. This means that it is one of the components of the default library for sending and receiving messages.

UPDATE 1: After the initial resolution round, this contract was refactored into various subcontracts:

- MessageLibBase: A simple new base contract that defines a few immutable variables and an onlyEndpoint modifier. We have no additional comments due to the simplicity of this contract.
- SendLibBase: Incorporates the sending functionality. We do not have any specific concerns regarding this contract.

2.3.1 Privileged Functions

- transferOwnership [owner]
- renounceOwnership [owner]

2.3.2 Issues & Recommendations

Issue #08	Decentralization flaw: Contract owner can fully brick all outbound messages of all applications using the library by setting a bad treasury
Severity	 MEDIUM SEVERITY
Description	<p>Upon sending messages within the LayerZero network, fees need to be paid to pay the verifiers, executors and the LayerZero treasury. All these fees are configured directly by the verifiers, executors and treasury themselves. A bad verifier could for example ask for an enormous fee, therefore bricking the sending pipeline.</p> <p>Though we consider that a flaw, the application owner can simply kick this bad verifier by setting a custom configuration.</p> <p>However, it turns out that the LayerZero treasury can do exactly the same thing, and this treasury is freely changeable by the contract owner, but not by the applications.</p> <p>This means that if the MessageLibBase owner is ever compromised, that owner has the freedom to break all applications using this library permanently. It can do this in two ways:</p> <ol style="list-style-type: none">1. Deny service by reverting the fee call or making it run out of gas2. Return an enormous amount, which is presently uncapped if the LayerZero token fee switch is turned on. This latter is less of an issue since the users can typically provide a boolean to pay fully in gas tokens.
Recommendation	<p>Consider using a constant gas <code>safecall</code> to call the treasury. Consider setting a cap to the LayerZero token fee if desired/seen appropriate.</p>
Resolution	 PARTIALLY RESOLVED
	<p>This is done via a gas-capped call, resolving the first error case.</p>

Issue #09

Availability flaw: A single optional verifier can brick all sending for an application, even if it is not a mandatory verifier

Severity

LOW SEVERITY

Description

The message library allows for the configuration of optional verifiers. This is a list of verifiers who do not all need to sign a transaction. For example, only a threshold of 3 out of the 5 optional verifiers might need to sign a message.

However, all of the 5 verifiers need to be available and functional on the source chain to communicate their fees. If any one of these optional verifiers has an issue (similar to the treasury issue) where it starts returning an enormous fee or reverts, the whole sending flow bricks until this verifier is kicked.

Recommendation

The primary resolution to this issue would be a system redesign where the fee logic is not as tightly coupled to the sending chain as it is now.

Though Paladin has communicated its ideal view of such a design with the client, we understand it is quite late for such changes, alongside the fact that the client might prefer their current design.

Resolution

ACKNOWLEDGED

This is by design and applications must chose reliable verifiers.

Issue #10	Gas optimization
Severity	 INFORMATIONAL
Location	<u>L148-149</u> <pre>require(_to != address(0x0) && _amount <= fees[msg.sender], Errors.INVALID_ARGUMENT); fees[msg.sender] -= _amount;</pre>
Description	The fee storage variable is read twice.
Recommendation	Consider implementing the gas optimization.
Resolution	 RESOLVED

Issue #11	Typographical issues
Severity	 INFORMATIONAL
Description	<u>L32</u> <pre>// accumulated fees for workers and treasury</pre> <p>The fees for the executor are accumulated within this mapping as well.</p> <u>L35</u> <pre>event ExecutorFeePaid(address executor, uint fee);</pre> <p>The executor should probably be indexed.</p> <u>L45</u> <pre>localEid = _localEid;</pre> <p>This can probably be fetched from the endpoint to make the configuration less error-prone.</p>
Recommendation	Consider fixing the typographical issues.
Resolution	 ACKNOWLEDGED

2.4 MessageLibBaseE2

MessageLibBaseE2 is the component for the UltraLightNode302 which defines most of its external functions. It also defines a couple of utility functions which are used internally within the message lib.

The core function this dependency defines is `send`, which is the central entry point function called by `EndpointV2` to send out messages. This function is required to calculate the gas fee for sending out this message.

It also defines the external function `withdrawFee` for verifiers, executors and the treasury to take out accrued fees (either in native gas tokens or the alternative token, depending on how the endpoint is configured) and defines `withdrawLzTokenFee` which allows the treasury to take out accrued fees in the form of the LayerZero tokens once the token has been enabled.

UPDATE 1: After the initial resolution round, the client has decided to split up this contract for the sending and receiving library.

- `SendLibBaseE2`: Basic changes were done though most of the logic was ported over.
- `ReceiveLibBaseE2`: Basic changes were done though most of the logic was ported over. One of the view functions now returns `true` and `false` instead of a status enum.

UPDATE 2: After the first resolution round, a `_payloadHash` argument was added to `_verifiable`.

2.4.1 Privileged Functions

- `setTreasury [ owner ]`
- `withdrawLzTokenFee [ treasury ]`
- `transferOwnership [ owner ]`
- `renounceOwnership [ owner ]`



2.4.2 Issues & Recommendations

Issue #12	Lack of on-chain incentive for execution
Severity	 LOW SEVERITY
Description	<p>There is no on-chain incentive for verifiers to actually sign transactions for the destination chain execution. All fees are namely paid up-front regardless of whether they commit to their promise to create a validation signature.</p> <p>This leads to misalignment requiring app configurations to carefully monitor behavior of verifiers and kick any who stop producing verifications (which is of course always necessary regardless).</p>
Recommendation	<p>The ideal resolution for this issue is to move the fee distribution logic to the destination chain, where verifiers who are actually part of the signing committee get an extra bonus.</p> <p>However, as such a resolution is difficult to implement at this stage of the project, we recommend clearly documenting this and that only reputable verifiers must be used, as they are not properly incentivized on-chain.</p>
Resolution	 ACKNOWLEDGED
	<p>The client has indicated that applications should choose an executor based on reliability metrics.</p>

Issue #13	Gas optimizations
Severity	<div data-bbox="454 199 636 232">  INFORMATIONAL </div>
Description	<p data-bbox="454 282 549 315"><u>L53-57</u></p> <pre data-bbox="454 327 1273 544"> MessagingReceipt memory receipt = MessagingReceipt(_packet.guid, _packet.nonce, MessagingFee(totalNativeFee, 1zTokenFee)); </pre> <p data-bbox="454 577 1337 667">We do not see the point of including the guid and nonce at this level, as the endpoint is already aware of these.</p> <p data-bbox="454 748 501 781"><u>L71</u></p> <pre data-bbox="454 792 1173 875"> address altTokenAddr = ILayerZeroEndpointV2(endpoint).altFeeToken(); </pre> <p data-bbox="454 909 1326 999">This should be cached in the constructor within an immutable variable to save significant gas.</p>
Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	<div data-bbox="454 1106 596 1140">  RESOLVED </div>



DescriptionL13 and 16

```
import "@layerzerolabs/lz-evm-protocol-v2/contracts/  
messagelib/libs/ExecutorOptions.sol";  
import "../interfaces/ILayerZeroExecutor.sol";
```

These imports appear to be unused.

L24 and 25

```
event NativeOrAltTokenFeeWithdrawn(address worker, address  
receiver, uint amount);  
event LzTokenFeeWithdrawn(address lzToken, address receiver,  
uint amount);
```

Indexing the receiver and worker variables might be valuable.

L38-42

```
function send(  
[...]  
) public virtual onlyEndpoint returns (MessagingReceipt  
memory, bytes memory) {
```

The virtual keyword can be removed, and this function can be made external.

L121-123

```
bytes memory executorOptions;  
WorkerOptions[] memory otherWorkerOptions; (executorOptions,  
otherWorkerOptions) = _getExecutorAndOtherOptions(_options);
```

This appears to be writable on 1 or 2 lines, placing the variable types on the left hand side of the last line.

L129

```
_assertMessageSize(msgSize, maxMsgSize);
```

We are not sure what the point of this check is given that this message is orchestrated by the sender while maxMsgSize comes from that sender's configuration as well. This check could be easily done at the application layer if it was necessary by the application. We believe most applications will know the exact message size and do not need to do such checks.

L143

```
// if endpoint didnot revert on deliverable check, it will  
return true if the message is deliverable. false if the  
message has been delivered
```

"did not" should have a space in between.

—

setTreasury lacks an event.

Recommendation	Consider fixing the typographical issues.
-----------------------	---

Resolution	RESOLVED
-------------------	---

Most of the issues have been resolved.

2.5 UlnConfig

UlnConfig is used to store OAPP configurations for the message lib. It extends the underlying OutboundConfig contract, integrating all of those configuration values for an ULN as well.

It defines the storage variables for the default configurations and OAPP-specific configurations that override these defaults. A separate configuration can be set for each chain.

The variables which can be configured are:

- `inboundConfirmations`: The minimum number of confirmations on the source chain for a message to be considered valid by the app. Must be high enough to avoid re-order risk.
- `verifiers`: A list of oracle contracts that must always attest transactions for the app.
- `optionalVerifiers`: A list of oracle contracts that must partially attest transactions for the app. At least the configured threshold number of optional verifiers within this list must validate transactions.
- `verifierCount`: The number of verifiers in the above verifiers configurations, used for potential gas saving purposes (though we don't think this is necessary).
- `optionalVerifierCount`: The number of verifiers in the above `optionalVerifierCount` configurations, used for potential gas saving purposes (though we do not think this is necessary).
- `optionalVerifierThreshold`: The minimum number of optional verifiers that must verify a message for it to be valid.
- `useCustomVerifiers`: Whether the default verifiers are overridden.
- `useCustomOptionalVerifiers`: Whether the default optional verifiers are overridden

These variables can be separately overwritten (e.g. `useCustomVerifiers` would be set to `true` in such a case), while the remaining variables can be pulled from the default configuration if desired.

`UlnConfig` is configurable solely through the endpoint, which allows for apps to call configuration functions that then call the `messagelib` and finally this `UlnConfig` where the actual configurations get stored. Apps can also call utility functions on the endpoint that can do things like snapshot the default configuration to avoid default changes from being adopted. This effectively freezes the current configuration and opts-out to automatic updates which could very well be a risk if the default configuration keys are compromised.

The owner of `UlnConfig` can change the default configuration which means it has full control.

UPDATE: During the subsequent resolution rounds, this contract was factored into the other contracts and therefore removed.



2.5.1 Issues & Recommendations

Issue #15	Typographical issues
Severity	 INFORMATIONAL
Description	<p><u>L78 and 91</u></p> <pre>(bool useCustomVerifiers, address[] memory verifiers) = abi.decode(_config, (bool, address[])); (bool useCustomOptionalVerifiers, address[] memory optionalVerifiers, uint8 threshold) = abi.decode(If useCustomVerifiers is set to false, we recommend validating that verifiers has a length of zero for configurational consistency. A similar requirement should be added for the optional verifiers.</pre> <p><u>L204</u></p> <pre>function getDefaultConfigByType(uint32 _remoteEid, uint32 _configType) external view returns (bytes memory) { This function adds a line storage config in three separate branches. This line can be written once at the top of the function to reduce the body size. Since this is simply a storage reference, we do not expect it to increase gas cost for the branch which does not use the variable.</pre> <p>——</p> <p>Many of the setters lack events, making indexing more difficult.</p> <p>——</p> <p>The encoding and decoding bytes are not consistent with each other. This means that the bytes from getConfigByType cannot be provided to the setConfigByType function. This is not necessarily a problem but the team should be aware of it.</p> <p>——</p> <p>getConfigByType can be marked as external.</p>

setDefaultConfig enforces that verifiers and optionalVerifiers are unique, but it does not enforce that the 2 lists do not have duplicates. A specific function could be added to check that the two lists do not contain any duplicates. Note that the impact of a duplicate is not that large hence we added this issue here.

Recommendation Consider fixing the typographical issues.

Resolution



Most of the issues were resolved.

Issue #16 Gas optimizations

Severity



Location

L141-142, 186 and 189

```
if (customConfig.inboundConfirmations != 0) {  
    rtnConfig.inboundConfirmations =  
    customConfig.inboundConfirmations;  
  
    if (config.inboundConfirmations == 0) {...}  
    return (abi.encode(config.inboundConfirmations), false);  
}
```

Description

Many of the hybrid config getters fetch the custom storage configurations twice: once to check whether they exist and once to return them. Consider caching the value to a local variable to save some gas.

Recommendation Consider implementing the gas optimizations mentioned above.

Resolution



2.6 OutboundConfig

OutboundConfig is an underlying component of UlnConfig and is used to store the outbound-related (sending) OAPP configurations for the message lib.

It defines the storage variables for the default configurations and OAPP- specific configurations that override these defaults. A separate configuration can be set for each destination chain.

The variables which can be configured are:

- `maxMessageSize`: The maximum size outbound messages are allowed to be
- `outboundConfirmations`: The required number of confirmations on the destination for the message to be valid, to avoid re-order risk
- `executor`: The address of the executor to actually execute the message action

Each of these variables can be separately overwritten, while the remaining variables can be pulled from the default configuration if desired.



2.6.1 Issues & Recommendations

Issue #17	Typographical issues
Severity	 INFORMATIONAL
Description	<p><u>L31 and 58</u></p> <pre>OutboundConfigStruct storage config = outboundConfig[_oapp] [_eid];</pre> <pre>OutboundConfigStruct storage config = defaultOutboundConfig[_eid];</pre> <p>This storage reference is written in multiple branches. Instead, to simplify the contract, it can be written once at the top of the function. Given that its simply a reference we do not expect this to affect gas.</p> <p>---</p> <p>Many of the setters lack events, making indexing more difficult.</p> <p>---</p> <p>The OAPP setters allow for setting configuration values to 0. However, 0 is used to indicate that the OAPP will use the default value which is not 0. It is therefore impossible for OAPP to set the config to 0. We do not think this poses an issue, however it should be documented that setting any value to 0 will use the default value.</p>
Recommendation	Consider fixing the typographical issues.
Resolution	 RESOLVED
	<p>Almost all the issues have been resolved, however the final point where we mention that the zero value can be set has not been changed. We still recommend that the client documents this.</p>

Issue #18	Gas optimizations
Severity	 INFORMATIONAL
Location	Lines 32, 36, 39, 43, 46, 50, 78, 79, 81, 82, 84 and 85
Description	Many of the hybrid config getters fetch the custom storage configurations twice: once to check whether they exist and once to return them. Consider caching the value to a local variable to save some gas.
Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	 RESOLVED



2.7 UltraLightNode301

UltraLightNode301 is a message lib implementation extremely similar to UltraLightNode302. The core difference is that this implementation will be used with the v1 endpoint, while 302 is supposed to be used with the v2 endpoint.

These two endpoints differ in some ways: the v1 endpoint does not directly support paying treasury fees in the LayerZero token for example. Next, the v1 endpoint does not split up delivery and execution, meaning that only one transaction always occurs at the destination chain. UltraLightNode301 is adjusted to accommodate these behaviors.

UPDATE 1: After the initial resolution round, changes to this contract were made. This contract has been split up into a send and receive portion as detailed below.

- **SendUln301:** Mostly directly copied from the send logic within the original code of Uln301. getConfig and setConfig were redesigned with the new configuration systems.
- **ReceiveUln301:** Some functions got renamed from the original receive functions but their logic remains mostly identical. Small changes and optimizations were added.

Update 2: After the second resolution round, basic stylistic changes to this contract were made. The contract moved to a new name for the verify function.

2.7.1 Privileged Functions

- `setDefaultConfig [ owner ]`
- `setLayerZeroToken [ owner ]`
- `setTreasury [ owner ]`
- `setAddressSize [ owner ]`
- `transferOwnership [ owner ]`
- `renounceOwnership [ owner ]`



2.7.2 Issues & Recommendations

Issue #19	The deliver function can be frontrun to prevent normal execution through an insufficient execution gas amount
------------------	--

Severity

 HIGH SEVERITY

Description

The deliver function is used once all the validators and enough optional validators have signed the message to confirm it is valid. The executor which gathers the different signatures must set the `_gasLimit` to a reasonable value to ensure the execution of the payload.

However, as the executor is not restrained to any address, a malicious user could frontrun the executor's transaction and update `_gasLimit` to 0 to create an execution which is guaranteed to fail. This would be annoying for most OAPPs as the users would need to retry their transaction themselves (except if an off-chain component can detect this). If any funds were locked/burned on the source chain, the user could then lose those funds forever.

In a more realistic scenario, if this vector was not documented and handled, a protocol relying on message availability would eventually get exploited. This could for example be the case with a protocol using LayerZero to relay oracle values to a lending platform. An exploiter could freeze this relaying and potentially exploit the protocol as it would start using an outdated price.

Recommendation

Consider adding the gas limit to the payload itself to ensure a minimum amount of gas forwarded by the executor. Another solution would be to restrict the executor to a set of trusted addresses. However, if an executor is ever compromised, the issue will arise again.

Finally, retry logic could be incorporated off-chain in the executors. It is important that clients are aware of such behavior and do not silence failure. This requirement is unfortunately not met for the majority of V1 apps since they pretty much all silently ignore any failure and cache the message internally. Simply adding logic to the executors to call `retryPayload` is therefore insufficient for non-blocking apps.

Resolution



301 execution is now exclusive to the configured outbound executor of the recipient. Note that this is different from 302 execution which is not explicitly limited to execution like this.

Issue #20

Typographical issues

Severity



Description

L7 and 12

```
import "@layerzerolabs/lz-evm-v1-0.7/contracts/interfaces/ILayerZeroEndpoint.sol";  
import "../libs/UlnOptions.sol";
```

These imports appear unused.

L17

```
/// @dev ULN301 will be deployed on EndpointV1 and is for  
backward compatability with ULN302 on EndpointV2. 301 can  
talk to both 301 and 302
```

This comment should be corrected to use "compatibility".

L39 and 71

```
uInConfig.setConfigByType(_eid, _oapp, uint32(_configType),  
_config);  
(config, ) = uInConfig.getConfigByType(_eid, _oapp,  
uint32(_configType));
```

Using SafeCast here would be valuable to make over flow explicitly revert, instead of returning a wrong type.

—

Unlike ULN302, the ULN301 lacks the `isSupportedEid` function. Consider adding it if it was an oversight.

Recommendation

Consider fixing the typographical issues.

Resolution



2.8 MessageLibBaseE1

MessageLibBaseE1 is a component of the default message lib for the v1 endpoint, UltraLightNode301. The contract defines most of the message lib's external functions. It also defines a couple of utility functions which are used internally within the message lib.

The core function this dependency defines is `send`, which is the central entry point function called by the v1 Endpoint to send out messages. This function is required to calculate the gas fee for sending out messages.

It defines the external function `withdrawFee` for verifiers, executors and the treasury to take out accrued fees (either in native gas tokens or the alternative token, depending on how the endpoint is configured).

It should be noted that the Endpoint V1 nonce is fully ignored for outbound messages.

UPDATE 1: After the initial resolution round, changes to this contract were made.

This contract has been split up into a `send` and `receive` portion:

- **SendLibBaseE1:** Basic changes were done compared to the `send` functionality of the original contract, such as stylistic changes and using the new GUID library. `setLayerZeroToken` was renamed to `setLzToken`. `setAddressSize` was removed. The `Transfer` library is now used for token and gas token transfers. `_sendToWorkers` is refactored into `_payWorkers` without any functional change.
- **ReceiveLibBaseE1:** This contract is mostly refactored compared to the original contract's `receive` functionality. This now pulls in the executor validation, which originally occurred at a higher level.

UPDATE 2: After the second resolution round, small typographical fixes and event renames were made in both contracts.

UPDATE 3: After the third resolution round, a small defensive requirement got introduced to the SendLibBaseE1 contract, reverting if the addresses have 0 bytes.

2.8.1 Privileged Functions

- `setLayerZeroToken [ owner ]`
- `setTreasury [ owner ]`
- `setAddressSize [ owner ]`
- `transferOwnership [ owner ]`
- `renounceOwnership [ owner ]`



2.8.2 Issues & Recommendations

Issue #21	Decentralization flaw: Contract owner can fully brick all outbound messages of all applications using the library by setting a bad treasury or layerZeroToken
Severity	 MEDIUM SEVERITY
Description	<p>Upon sending messages within the LayerZero network, fees need to be paid to pay the verifiers, executors and the LayerZero treasury. All these fees are configured directly by the verifiers, executors and treasury themselves. A bad verifier could for example ask for an enormous fee, bricking the sending pipeline.</p> <p>Though we consider that a flaw, luckily the application owner can simply kick such a bad verifier by setting a custom configuration.</p> <p>However, it turns out that the LayerZero treasury can do exactly the same thing, and this treasury is freely changeable by the contract owner, but not by the applications.</p> <p>This means that if the MessageLibBase owner is ever compromised, that owner can to break all applications using this library permanently. It can do this by:</p> <ol style="list-style-type: none">1. Denying service by reverting the fee call or making it run out of gas.2. Return an enormous amount, which is presently uncapped if the LayerZero token fee switch is turned on. This is less of an issue since the users can typically provide a boolean to pay fully in gas tokens.3. Revert due to the LayerZero token being set to a bad contract.
Recommendation	<p>Consider using a constant gas <code>safecall</code> to call the treasury. Consider setting a cap to the LayerZero token fee if desired/seen appropriate. Consider only allowing the LayerZero token to be set once.</p>
Resolution	 ACKNOWLEDGED
	<p>The client will address this at a later stage.</p>

Description

Many of the setters lack events, making indexing more difficult.

——

Various events could have their receiver and user indexed.

L5, 11 and 18

```
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";  
import "@layerzerolabs/lz-evm-protocol-v2/contracts/  
messagelib/libs/ExecutorOptions.sol";  
import "../../interfaces/ILayerZeroExecutor.sol";
```

These imports appear unused.

L25

```
using AddressCast for bytes32;
```

It seems like this is not directly used, `AddressCast.toAddress()` is used instead. Consider either removing this line or adjusting the `toAddress` call to use this utility.

L28

```
address internal immutable treasuryFeeHandler;
```

This can be `ITreasuryFeeHandler` to avoid casting it later on.

L52

```
address _nonceContract,
```

This can be cast to `INonceContract` or `NonceContract` to avoid casting later on. We would prefer an interface to be used here however.

L53

```
uint32 _localEid
```

This can be fetched from the endpoint directly to reduce configurational risk.

L188 - 190

```
bytes memory executorOptions;  
WorkerOptions[] memory otherWorkerOptions;  
(executorOptions, otherWorkerOptions) =  
_getExecutorAndOtherOptions(_options);
```

This appears to be writable on 1 or 2 lines, placing the variable types on the left hand side of the last line.

L229

```
ILayerZeroEndpoint lzEndpoint =  
ILayerZeroEndpoint(endpoint);
```

This variable is unnecessary and wastes a tiny amount of gas. Consider casting the endpoint directly where it is used, or update the immutable variable to be ILayerZeroEndpoint instead of address.

nonceContract and treasuryFeeHandler should be made public to allow them to be inspected easily within the explorer.

Recommendation	Consider fixing the typographical issues.
-----------------------	---

Resolution	RESOLVED
-------------------	---

Most of the issues have been resolved.

2.9 TreasuryFeeHandler

TreasuryFeeHandler is a simple utility used by the 301 message library to handle the collection of LayerZero token treasury fees.

Since the V1 endpoint does not directly integrate with the token, this separate contract is used to collect the fees. OAPPs must manually approve the LayerZero token to the handler for sending to function.

UPDATE 1: After the initial resolution round, basic stylistic changes to this contract were made. As with all other changes, these have been validated.



2.9.1 Issues & Recommendations

Issue #23	Governance risk: Apps must make sure to not give infinite allowance to this contract as this could result in the allowance being drained if the LayerZero governance is compromised
Severity	 LOW SEVERITY
Description	<p>Any message library configured by the applications can instantiate a pull of LayerZero tokens from that application into the TreasuryFeeHandler. Up to the open allowance from the app to the handler can be pulled.</p> <p>If an application uses the default library flow, the LayerZero team's ownership address can freely adjust the default message lib. If this governance address is ever compromised, it could be upgraded to a malicious address which calls TreasuryFeeHandler to drain all app LayerZero token allowances. This can be trivially safeguarded by having apps not give excess allowance to this contract.</p>
Recommendation	Consider documenting that apps should simply give only the allowance required to this contract. Even better, consider doing this in the primary default application dependency.
Resolution	 RESOLVED The client will document this recommendation.

Issue #24	Typographical issues
Severity	INFORMATIONAL
Description	<p><u>L16</u></p> <pre>constructor(address _endpoint) {</pre> <p>This parameter can ILayerZeroEndpoint to avoid casting it later on.</p> <p><u>L21</u></p> <pre>function payFee(address _lzToken, address _sender, uint _required, uint _supplied, address _treasury) external {</pre> <p>The _lzToken parameter can be provided as IERC20 to avoid casting it later on.</p> <p><u>L28</u></p> <pre>// lz token payment address must equal the sender</pre> <p>The comment seems outdated. Consider updating it.</p>
Recommendation	Consider fixing the typographical issues.
Resolution	PARTIALLY RESOLVED <p>The comment on line 28 has been removed.</p>

2.10 UlnOptions

UlnOptions is responsible for decoding bytes sent to the send function into validator and executor options. The first two bytes of the options must indicate the option type which can be either the legacy types 1 or 2, or the current type 3. The current type is a concatenation of worker options. Each worker option is encoded as [worker_id: 1 byte][option_size: 2 bytes][option_data: option_size bytes].

The legacy types are encoded as follows:

- Type 1: [execution_gas: 32 bytes]
- Type 2: [execution_gas: 32 bytes][amount: 32 bytes][receiver: min(remaining bytes, 32 bytes)]

It is therefore clear that at first with the legacy types, each type served a specific purpose with type 2 extending type 1 with an airdrop quantity of gas tokens. However, with type 3, a general format was introduced which encodes arbitrary option data. The current UlnOptions implementation translates type 1 and 2 into a type 3 compatible byte sequence, homogenizing all three types.

It should be noted that the UlnOptions library does not strictly enforce correctness of the underlying option data. This must be handled by the OAPP and message library.

UPDATE 1: After the initial resolution round, basic stylistic changes to this contract were made. As with all other changes, these have been validated.

2.10.1 Issues & Recommendations

Issue #25	decode does not handle a zero _options length
Severity	 LOW SEVERITY
Location	<u>L24</u> <code>uint16 optionsType = uint16(bytes2(_options[0:2]));</code>
Description	The decode function assumes options are always supplied. However, it may make sense for applications to start sending messages without any options. In this case, they must still provide two bytes of data for the function not to revert.
Recommendation	Consider adding an early return in case the length of _options is zero.
Resolution	 RESOLVED A requirement has been added instead. This is fine as well.

Issue #26	Typographical issues
Severity	 INFORMATIONAL
Location	<u>L125</u> <code>// option_type: uint8, executionGas: uint128</code>
Description	This is inconsistent with the rest of the comments as the rest use execution_gas as the variable.
Recommendation	Consider fixing the typographical issues.
Resolution	 RESOLVED

Severity

 INFORMATIONAL

Description

L46

```
if (workerId != lastWorkerId) {
```

This can be an else-if to make the logic more explicit and potentially save some gas.

L67

```
if (cursor == _options.length) {
```

This check can be moved under the equality requirement outside of the loop. The if-statement would need to be replaced with a check that some workerId was set in the while loop however, as zero options might be provided.

——

Overall, we do also feel that these options libraries consume significant gas in their attempt not to make any assumptions on the input structure. We feel like this trade-off might not be ideal for on-chain execution, especially on gas expensive chains like Ethereum.

Recommendation

Consider implementing the gas optimizations mentioned above.

Resolution

 RESOLVED

2.11 VerifierOptions

`VerifierOptions` defines the necessary logic to decode a concatenated list of verifier options into a more structured list of such options. The specific structure which is applied is that any options which are for a specific verifier are grouped to that specific verifier's index.

The structure of the raw `VerifierOptions` bytes is that it is a concatenation of verifier options. Each verifier option has the following structure: `[worker_id: 1 byte][option_length: 2 bytes][verifier_idx: 1 byte][option_data: option_length - 1]`. It should be noted that the `worker_id` is typically constantly set to the verifier type 2, `option_length` includes `verifier_idx` and `option_data` should be encoded as `[option_type: 1 byte][option: option_length - 2 bytes]`.

As with `UlnOptions`, no explicit verification on the actual data is done, nor is there direct validation on the option size to be sensible. This means that sizes of example 0 would implicitly revert. If desired, it may make sense to add such validation in.

UPDATE 1: After the initial resolution round, basic stylistic changes to this contract were made. Notably, the unchecked block in `_insertVerifierOptions` was removed.

2.11.1 Issues & Recommendations

Issue #28	Gas optimization
Severity	 INFORMATIONAL
Location	<u>L79</u> <code>if (cursor == _options.length) {</code>
Description	This logic could be handled outside of the for-loop to save some gas. Explicit care should be taken if this is moved outside of the loop however.
Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	 RESOLVED

Issue #29	Typographical issues
Severity	 INFORMATIONAL
Description	<u>L19</u> <code>/// @param _options [verifier_id][verifier_option]</code> <code>[verifier_id][verifier_option]...</code> We believe this should state <code>worker_id</code> instead of <code>verifier_id</code> . <u>L67</u> <code>// this operation slices out options whenever the the scan</code> <code>finds a different lastVerifierIdx</code> The comment contains a typographical error as "the" is written twice.
Recommendation	Consider fixing the typographical issues.
Resolution	 ACKNOWLEDGED

2.12 AddressSizeConfig

AddressSizeConfig is a simple dependency used in the base send and receive libs to configure address bytecode sizes per destination endpoint id. These sizes can then be used for validation.

The maximum address size is 32 bytes. Address sizes can only be configured once.

2.12.1 Privileged Functions

- setAddressSize
- transferOwnership
- RenounceOwnership

2.12.2 Issues & Recommendations

No issues found.





PALADIN
BLOCKCHAIN SECURITY