PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For LayerZero (V2)

15 December 2023

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 V2 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 Consensys, ZK Polygon, Zora
Language	Solidity
Preliminary Contracts	https://github.com/LayerZero-Labs/monorepo/tree/bb86736199a7c1fd61983d28093a53ad0e27e153/packages/layerzero-v2/evm/protocol/contracts Excluded contracts: - protocol/contracts/libs/messagelib/ExecutorOptions.sol - SimpleMessageLib.sol
Resolution #1	https://github.com/LayerZero-Labs/monorepo/tree/fe287f0ef05b577818ad5daf768875907ad1ac9c/packages/layerzero-v2/evm/protocol/contracts
Resolution #2	https://github.com/LayerZero-Labs/monorepo/tree/faf36d0862245705eb51aac4ad65cc442d792ad9/packages/layerzero-v2/evm/protocol/contracts
Resolution #3	https://github.com/LayerZero-Labs/LayerZero-v2/commit/ccfd0d38f83ca8103b14ab9ca77f32e0419510ff
Resolution #4	https://github.com/LayerZero-Labs/LayerZero-v2/tree/ccfd0d38f83ca8103b14ab9ca77f32e0419510ff/protocol/contracts

Resolution #5	https://github.com/LayerZero-Labs/LayerZero-v2/tree/4b2985921af42a778d26a48c9dee7b9644812cbd/protocol/contracts
Resolution #6	https://github.com/LayerZero-Labs/LayerZero-v2/tree/6356f0a31f2125267420e7bf8403b59a55459aa3/protocol/contracts

1.2 Contracts Assessed

Name	Contract	Live Code Match
EndpointV2	Meter: 0xef02BacD67C0AB45510927749009F6B9ffCE0631 Shimmer: 0x148f693af10ddfaE81cDdb36F4c93B31A90076e1 ZkSync: 0xd07C30aF3Ff30D96BDc9c6044958230Eb797DDBF All other chains: 0x1a44076050125825900e736c501f859c50fE728c Note: The BB1, Conflux, DKF, Rarible, Tomo, XPla, and ZkSync contracts do not match the ETH one.	✓ MATCH
MessageLibManager	Dependency	✓ MATCH
MessagingChannel	Dependency	✓ MATCH
MessagingComposer	Dependency	✓ MATCH
MessagingContext	Dependency	✓ MATCH
AddressCast	Dependency	✓ MATCH
SafeCall	Dependency	✓ MATCH
BitMaps	Dependency	✓ MATCH
PacketV1Codec	Dependency	✓ MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
🔴 **High**	0	-	-	-
🟠 **Medium**	3	2	-	1
🟡 **Low**	10	6	-	4
🟣 **Informational**	20	13	2	5
Total	**33**	**21**	**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 Global Issues

ID	Severity	Summary	Status
01	INFO	The `uint` keyword should not be used	ACKNOWLEDGED

1.3.2 EndpointV2

ID	Severity	Summary	Status
02	MEDIUM	`EndpointV2` allows for inconsistent delivery behavior if a recipient contract is not yet deployed	ACKNOWLEDGED
03	MEDIUM	`setLayerZeroToken` can be used by a compromised governance owner to drain the `altFeeToken`	✓ RESOLVED
04	LOW	Malicious message could be hidden and used at a later stage and other messages might not be executed at all	✓ RESOLVED
05	LOW	`transfer` is used for gas transfers which may fail to specific recipients	✓ RESOLVED
06	LOW	`lzReceivelzCompose` no longer adheres to checks-effects-interactions, making the code less verifiable	ACKNOWLEDGED
07	INFO	Typographical issues	✓ RESOLVED

1.3.3 MessageLibManager

ID	Severity	Summary	Status
08	LOW	Decentralization is opt-in compared to potentially more appealing opt-out	✓ RESOLVED
09	LOW	It is impossible for an app to set a grace period to the previous lib if they update it to the default value or from the default value	✓ RESOLVED
10	LOW	The `unset` and `default` flag are set to the same value	ACKNOWLEDGED
11	LOW	`isSupportedEid` will sometimes be called on the default lib while other times it will not be, leading to potential inconsistency	✓ RESOLVED
12	INFO	`registerLibrary` will seldom revert with UNSUPPORTED_INTERFACE if a bad interface is provided, instead it will revert ambiguously	ACKNOWLEDGED
13	INFO	UI function `getRegisteredLibraries` can run out of gas	ACKNOWLEDGED
14	INFO	`DefaultReceiveLibraryTimeoutSet` event is inconsistent between various functions	✓ RESOLVED
15	INFO	`snapshotConfig` and `resetConfig` can likely be abstracted from the message lib interface	✓ RESOLVED
16	INFO	Typographical issues	PARTIAL

1.3.4 BlockedMessageLib

ID	Severity	Summary	Status
17	MEDIUM	`BlockedLibrary` can never be set	✓ RESOLVED
18	INFO	Fallback should implement payable to also revert on the correct error if gas tokens are sent	ACKNOWLEDGED

1.3.5 MessagingChannel

ID	Severity	Summary	Status
19	LOW	`inboundNonce` might occasionally revert ambiguously due to running out-of-gas	ACKNOWLEDGED
20	INFO	Lack of validation: `_payloadHash` lacks a non-zero check within `_inbound` which can lead to extra business logic errors with bad receive libraries	✓ RESOLVED
21	INFO	Gas optimizations	PARTIAL
22	INFO	Typographical issues	✓ RESOLVED

1.3.6 MessagingComposer

ID	Severity	Summary	Status
23	LOW	`lzCompose` will not revert if called to an address without any code	✓ RESOLVED
24	LOW	`lzCompose` no longer adheres to checks-effects-interactions, making the code less verifiable	ACKNOWLEDGED
25	INFO	`lzCompose` can be called multiple times if the hash of the message is `bytes32(1)`	✓ RESOLVED

1.3.7 MessagingContext

ID	Severity	Summary	Status
26	INFO	Typographical issue	✓ RESOLVED
27	INFO	`address(1)` can theoretically re-enter into the contract, though practically impossible as this address is not expected to ever be owned	✓ RESOLVED
28	INFO	Gas optimizations	✓ RESOLVED

1.3.8 AddressCast

ID	Severity	Summary	Status
29	INFO	`toBytes` assembly block erroneously accesses a potentially misconfigured memory slot resulting in potentially dirty bytes	✓ RESOLVED
30	INFO	Gas optimizations	✓ RESOLVED

1.3.9 SafeCall

ID	Severity	Summary	Status
31	INFO	Outdated comment	✓ RESOLVED
32	INFO	Gas optimization	✓ RESOLVED

1.3.10 BitMaps

ID	Severity	Summary	Status
33	INFO	Gas optimization	✓ RESOLVED

1.3.11 PacketV1Codec

No issues found.

2 Findings

2.1 Global Issues

The issues listed in this section apply to the protocol as a whole. Please read through them carefully and take care to apply the fixes across the relevant contracts.

2.1.1 Issues & Recommendations

Issue #01	**The `uint` keyword should not be used**
Severity	INFORMATIONAL
Description	In the most of the contracts, the `uint` keyword is used. We recommend only using the `uint256` keyword to make the code clearer.
Recommendation	Replace all instances of the `uint` keyword with `uint256`.
Resolution	ACKNOWLEDGED The client has opted not to make this change as it is purely stylistic and does not affect users. They prefer to keep unnecessary modifications to a minimum to keep the codebase as secure as possible which is something we respect.

2.2 EndpointV2

`EndpointV2` The the core contract within the LayerZero V2 infrastructure. It allows the sending and receiving of messages from and to OAPPs.

Sending logic

An application can simply call `send` or `sendWithAlt` on the endpoint to send data to any contract on any LayerZero supported chain. When these functions are called with a sufficient fee, a packet will be emitted on the source chain which will be picked up by the relayers and oracles to eventually be transmitted to the destination chain for receipt on the destination endpoint.

Each endpoint can be configured to accept sending fees in the native gas token of that chain or an alternative ERC20 token. The LayerZero team can also configure the LZO token which effectively allows for people to pay a portion or all of the fee in the LZO token.

LayerZero allows message senders to freely configure the sending library which is used — such a library is crucial and must be carefully chosen.

For sending messages, the sender needs the absolute minimum configuration by default and uses the default sending library responsible for taking care of the encoding and transmission of the packet alongside other things like the fee calculation. If the LayerZero team changes the default library however, the new library would automatically be used by the apps which did not overwrite the default. This can be a significant governance risk and we recommend apps to carefully weigh the risks of not configuring explicit libraries.

We reiterate that libraries are crucial within LayerZero and that by default, all apps just use the default library which can be changed. If a bad library is configured, fake or adjusted packets can be emitted, which can lead to complete drainage of the applications.

When sending messages, each message is tagged with a nonce. Nonces are unique for each { sender, receiver, source endpoint ID, destination endpoint ID } combination and increment with every message over that combination. Nonces start at 1 for the first message and are set to 0 before any message is sent.

This means that if a sender sends to a new chain, the nonce would start at 1 again for those messages, though messages sent to a recipient on a previous chain still increment with their own nonce.

When sending messages, sufficient approval for the alternative token and LZO token need to be provided as these pull the necessary funds from the sender. The native gas fee is simply provided with the call and automatically refunded partially at the end if excessive gas was provided.

Receiving

Similar to sending messages, apps can configure a receiving library to handle the receipt of messages if they want. If not configured, a default library is used which is automatically upgraded if the LayerZero team configures a new one.

From the endpoint's perspective, it assumes that the receipt library does any and all validation, checks and relaying of messages. This means that there is no validation at the endpoint level that a message actually was transmitted on a source chain. Instead, it is the receipt library who validates this.

It is expected that the default receipt library does this in two parts using:

1. An oracle to transmit the block headers from the source chain to the receipt endpoint chain
2. A proof library which can validate proofs that the original transaction with a specific nonce (or its emission log) was included in that block.

Then relayers will transmit the proofs and packets from the original chain to the recipient chain whenever the block hashes are transmitted by the oracle. This means that within the expected LayerZero implementation, as long as the receive library is correctly coded, relaying fake messages would require both the oracle and relayers to be compromised. Note that fake messages can be relayed by bad/exploitable receive libraries, especially through a governance compromise which simply sets the default receive library to a bad one, exploiting all apps which did not overwrite the default.

When a message is validated by the receipt library (out-of-scope for this portion of the audit by Paladin), the receipt library calls deliver on the endpoint which will register all the details of the message to allow it to be executed.

On delivery, unverified `extraData` can be added by whoever executes the message. If such extra data is desired (e.g., this could be an off-chain oracle value, like Coinbase's signed oracle values), it must be validated (i.e. the signature must be checked). This is completely untrusted as anyone can call `lzReceive`.

Update 1: After the initial resolution round, this contract has been adjusted. Some functions and events were renamed.

UPDATE 2: After additional resolution round, this contract has been adjusted. Primarily, the contract has been refactored to allow for a contract extension called the `EndpointV2Alt` to be used for chains which want to pay for gas fees using a non-native token (e.g. an ERC20 token). Several stylistic changes were also made — requirements were replaced with if-reverts. `delivery` was renamed to `verify` and several `view` functions were adjusted.

UPDATE 3: After the third round, this contract was adjusted again. Token payments were inverted so that the native payments now comes under `_payToken` for send. We commend this as we also like ordering interactions where the most reentrancy prone ones come last. Receipt OAPP can now block receipt of a specific path by returning false on `allowInitializePath`. Note that it can only block the first nonce therefore it should not go from `true` to `false` again as that will not fix anything.

UPDATE 4: After the fourth round, this contract was modified again. The primary change is in the `delegate` functionality where apps can denote a delegate address that is authorized to adjust the app's configurations. This address should be highly trusted and ideally a smart contract, as these configurations are extremely privileged to the point where they can fully configure the security of the app. Next, messages can now still be re-verified if they are in `lazyInboundNonce` (this was previously not possible), as long as they are not executed yet. This allows for the delivery mechanism to be more flexible with re-organizations according to the client, though we recommend them to be extremely careful with such behavior as all delivered executions should be from the same source fork. Finally, a small change to the verifiable `view` function was made to make it more closely resemble the verification function requirements.

2.2.1 Privileged Functions

- `deliver [ receive libraries ]`
- `clear [ oapps delegates ]`
- `setLayerZeroToken [ owner ]`
- `recoverToken [ owner ]`
- `skip [ oapps delegates ]`
- `registerLibrary [ owner ]`
- `setDefaultSendLibrary [ owner ]`
- `setDefaultReceiveLibrary [ owner ]`
- `setDefaultReceiveLibraryTimeout [ owner ]`
- `setSendLibrary [ oapps delegates ]`
- `setReceiveLibrary [ oapps delegates ]`
- `setReceiveLibraryTimeout [ oapps delegates ]`
- `setConfig [ oapps delegates ]`
- `snapshotConfig [ oapps delegates ]`
- `resetConfig [ oapps delegates ]`
- `transferOwnership [ owner ]`
- `renounceOwnership [ owner ]`

2.2.2 Issues & Recommendations

Issue #02	**`EndpointV2` allows for inconsistent delivery behavior if a recipient contract is not yet deployed**
Severity	● MEDIUM SEVERITY
Description	Recipients can freely choose which receipt library they use to avoid potentially using a compromised default library, which can be set by the `EndpointV2` owner at any point in time. However, these recipient OAPPs cannot configure themselves before they are deployed, and there ends up being certain edge cases where these apps would use the wrong library inadvertently: 1. deliver to 0x1 2. deploy 0x1 3. `lzReceive` to 0x1 or 1. deliver to 0x1 2. `lzReceive` to 0x1 3. deploy 0x1 Either of these cases is likely undesirable for the application (0x1). Especially, in the first case, if the default library is compromised, the compromised library would be able to preload malicious transactions to to-be-deployed apps which hardcode their libraries during deployment.
Recommendation	Consider whether it makes sense to make application registration explicit, where applications cannot receive any messages until they explicitly register their libraries to the endpoint. This would come with the added benefit that decentralization is no longer opt-in. During registration, the app simply chooses their preferred approach. In our opinion, being able to deliver to yet-to-be-deployed contracts appears futile and safeguarding against it is probably not a terrible idea.

Resolution	✓ RESOLVED The client added a safeguard at the protocol level for this. `allowInitializePath` is now called on the receiver before the first message on the path is called, requiring the receiver to explicitly permit receipt.

Issue #03	**`setLayerZeroToken` can be used by a compromised governance owner to drain `altFeeToken`**
Severity	● MEDIUM SEVERITY
Description	Though the comments explicitly state this should not be possible as users are not supposed to approve the endpoint for any other token than the LZO token, users in fact can: there is a new configuration where users approve the alternative fee token if that endpoint is configured not to use the native gas token. In that case, after the LZO token has been enabled, the LayerZero governance (if compromised) could set the LZO token to the alternative fee token. Thus, any approved alternative fee tokens from users would get drained instead of the expected LZO token. This is especially problematic if the alternative fee token is significantly more valuable than the LZO token.
Recommendation	Consider adding a requirement to the LayerZero token setter that it may not equal to this alternative fee token.
Resolution	✓ RESOLVED The recommended requirement has been introduced.

Issue #04	Malicious message could be hidden and used at a later stage and other messages might not be executed at all
Severity	● LOW SEVERITY
Description	Within LayerZeroV2, messages can be delivered in any order and can be executed following any order as long as all the previous messages were delivered. Messages may even never be executed at all. This is a risk as certain messages might not be executable today, but then as soon as some parameter that was causing them to revert is changed, they all become executable at once. If any sort of retry logic is employed at the source, this might cause issues at the application level. Furthermore, as the delivery and execution are now done in two calls, a malicious relayer could deliver a message without them or anyone else executing it to completely hide the message. Unless there is good frontend tooling, it will be tedious for the application to ensure that all messages were actually executed. Finally, allowing a message to be replayed without any deadline is very risky as a malicious user could use this to hide messages and for example mint a lot of tokens at a later stage such as after they revoked the ownership.
Recommendation	Consider forcing the delivery and the execution of the message in the same call instead of doing it in stages, or at least give the application the freedom to request this. Consider alternatively providing sufficient off-chain tooling and interfaces so that application managers can quickly notice if any transaction is not delivered. We recommend an application dashboard showing the in-flight messages sorted by how long ago they were sent. This would allow for manual execution or debugging if anything goes wrong with delivery (or nobody wants to deliver). Finally, it might be a good idea to consider documenting the incentives for whoever is executing messages. Right now it appears that the economical incentivization of the executors is out-of-scope.

Resolution	✓ RESOLVED The client has indicated that this is by design and that they would need to fundamentally give up their design principles to resolve this. They will be clear in their communication to app developers as to ensure that missed executions are caught properly. No changes were made.

Issue #05	**`transfer` is used for gas transfers which may fail to specific recipients**
Severity	● LOW SEVERITY
Description	The `transfer` function should be avoided as it might revert when sending to specific contracts that need more gas than what is hardcoded by the transfer Solidity opcode. This is specifically the case with certain proxy implementations and vault contracts.
Recommendation	Consider using `.call()` instead. Make sure to verify the success of the call. Ensure that the whole codebase adheres to checks-effects-interactions and that no remaining reentrancy vulnerabilities are present with interactions after these `.call` statements.
Resolution	✓ RESOLVED The client would prefer to keep `transfer` to minimize the attack surface with regards to reentrancy. No changes were made. We have re-iterated our concern with special contracts and chains with different gas costs to the client and they have indicated that they will be careful with such integrations. As this is a security audit, this issue is marked as resolved as indeed moving to `call` increases the attack surface.

Issue #06	**`lzReceivelzCompose` no longer adheres to checks-effects-interactions, making the code less verifiable**
Severity	● LOW SEVERITY
Description	The client made a post-audit adjustment where the refund logic for failure within the `lzReceivelzCompose` function is now executed before the `composedMessage` is re-enabled. Though we validated that there is no user risk in this change to the best of our knowledge, moving away from checks-effects-interactions makes the codebase more error prone and makes validation of its safety more difficult.
Recommendation	Consider reverting to checks-effects-interactions
Resolution	✓ RESOLVED `lzReceive` no longer does try-logic and instead just reverts on failure. The payload clearage adheres to CEI.

Issue #07	Typographical issues
Severity	● INFORMATIONAL

Description	L38
	`/// @return native fee and lz token fee. if altFeeToken is enalbed, the native fee is in the form of the altFeeToken`
	
	"enabled" is mis-spelled.
	
	L67
	`/// @dev protected by send non-enentrancy guard`
	
	The comment should say "non-reentrancy" instead of "non-enentrancy".
	
	L69
	`/// @dev this design provides a saparation of concern and users only need to approve endpoint instead of messaging libraries`
	
	"separation" is mis-spelled.
	
	L271
	`function _payToken(address _token, address _sender, uint _required, uint _supplied, address _receiver) internal {`
	
	The `_supplied` parameter is more adequately called `_cap` here since this function is pull-based compared to the push-based `_payNative`.
	
	———
	
	Several events do not adhere to checks-effects-interactions. These events should ideally be moved above all interactions (eg. within send) wherever possible.
	
	———
	
	`recoverToken` should emit an event for easier indexing.

	The `FeePaid` event can be combined with the `PacketSent` event to save a bit of gas and make the code more readable. —— The tokens can be marked as `IERC20` to make their types more explicit within the storage section and within the various function parameters.
Recommendation	Consider fixing the typographical issues.
Resolution	✓ RESOLVED Most of these issues were resolved.

2.3 MessageLibManager

`MessageLibManager` is a dependency within `EndpointV2` and is responsible for keeping track of the default libraries used to transmit and validate incoming and outgoing transactions, as well as keeping track of non-default libraries which can be set by apps to avoid going with a potentially ill-fit default library and more importantly do not automatically follow upgrades.

UPDATE 1: After the initial resolution round, this contract was modified. The `isSendLib` and `isReceiveLib` functions were added and changed to block number expiry. We have indicated our preference to timestamp-based expiry to the client.

UPDATE 2: After the second resolution round, this contract was modified. The `isValidReceiveLibrary` function has been changed to an alternative which reverts instead of returning a boolean. An initialize mechanism was introduced to allow apps to be configured without them being able to receive messages. However, this mechanism was replaced with the path validation function in a later update.

UPDATE 3: After the third round, this contract was modified again. Several updates with regards to initialization were reverted back to the original code.

UPDATE 4: The setter functions now use the authorization pattern from the endpoint, allowing third parties who have been approved to set functionality.

2.3.1 Privileged Functions

- `registerLibrary [ owner ]`
- `setDefaultSendLibrary [ owner ]`
- `setDefaultReceiveLibrary [ owner ]`
- `setDefaultReceiveLibraryTimeout [ owner ]`
- `setSendLibrary [ oapps ]`
- `setReceiveLibrary [ oapps ]`
- `setReceiveLibraryTimeout [ oapps ]`
- `setConfig [ oapps ]`
- `snapshotConfig [ oapps ]`
- `resetConfig [ oapps ]`
- `transferOwnership [ owner ]`
- `renounceOwnership [ owner ]`

2.3.2 Issues & Recommendations

Issue #08	Decentralization is opt-in compared to potentially more appealing opt-out
Severity	LOW SEVERITY
Description	Within the current deployment, all apps by default are at the mercy of the owner of the `EndpointV2`. This is because this owner can override the default libraries over time, which are then used by any app which has not configured their own libraries. This means that if the keys of the owner (likely to be a multi-signature contract) are ever compromised, all apps which did not call `setReceiveLibrary` and `setSendLibrary` will effectively be fully exploited. We assume that this set will be large in case there is insufficient education regarding this behavior (and probably even large with sufficient education) though we do also understand the benefits of opting into this centralization as it allows for rolling upgrades and security fixes to automatically apply to the apps of other projects.
Recommendation	Consider documenting these trade-offs extremely carefully to not mislead projects. An analysis could be done on the current state of apps to see how many actually configure frozen libraries and how many roll with upgrades to see the degree of "education" which is necessary. If rolling security fixes are not a big business logic requirement, an "opt-out" decentralization model might be more than adequate. The team will need to carefully weigh their options to figure out what the right approach is. Perhaps an explicit declaration at the OAPP level where the projects explicitly choose is most appealing.
Resolution	✓ RESOLVED The client has made the opt-in less forced as now the apps can adjust their configuration freely and only enable themselves after being fully configured.

Issue #09	**It is impossible for an app to set a grace period to the previous lib if they update it to the default value or from the default value**
Severity	● LOW SEVERITY
Location	L240 `require(oldLib != _DEFAULT_LIB && _newLib != _DEFAULT_LIB, Errors.ONLY_NON_DEFAULT);`
Description	As this check requires both libs to be different from the default lib, an app will not be able to add a grace period for the old library if it wishes to revert to the default library. Likewise if an app wanted to update from the default library to a non-default one. This issue is categorised as low severity as the comment documents that behavior. We however still raise a concern as we assume that the intended usage of this grace period is to allow in-flight messages to still deliver with the correct library, a case which may occur when moving away or back to the default library. It should be noted that in either direction it appears possible to still achieve a grace period by going to the new library in two steps: first through the real address of the default library and then to the default library zero address (or vice versa). This means that if there was a business logic reason to have this requirement, this reason is circumventable, which may be undesired.
Recommendation	Consider whether this is fine, if not, consider letting applications set a deadline when they go to or from a default library.
Resolution	✓ RESOLVED The client has indicated that this is fine by them given that there are methods to still set a timeout for this library. No changes were made.

Issue #10	The `unset` and `default` flag are set to the same value
Severity	● LOW SEVERITY
Description	`_DEFAULT_LIB` and `_UNSET_LIB` are set to the same value. This is not recommended as it is ambiguous and is impossible for an app to set the value to one of the two flags without being in the second state as well.
Recommendation	Consider using two different values, or to remove one of the two flags.
Resolution	● ACKNOWLEDGED Although one of the flags has been removed and hardcoded instead, they both still reference the same value.

Issue #11	**`isSupportedEid` will sometimes be called on the default lib while other times it will not be, leading to potential inconsistency**
Severity	● LOW SEVERITY
Description	The `onlySupportedEid` function checks a library whether it supports a specific EID only if that provided library is not the default library (signified as the zero address). However, sometimes the default library is provided to this modifier as a full address, e.g. possibly within `setSendLibrary` and `setReceiveLibrary`. In both these functions, the library can be provided as the zero address or as a hardcoded address but the `eid` support will only be checked in the latter case. It is hard for us to come up with a reason why it should be checked there but not in the former case. A secondary inconsistency is that certain functions like `getConfig` miss a `onlySupportedEid` modifier altogether. We are unsure what the methodology was to choose which functions are or are not guarded.
Recommendation	Consider whether this is desired. A more sensible check is to resolve the default library and still call the function in case the zero address is provided. Also consider which functions should be guarded with `onlySupportedEid` — as of now this appears inconsistent.
Resolution	✓ RESOLVED The client has indicated they prefer not to have such checks as they can be checked during the setting of the libs. The check has been removed from `defaultLib` to be consistent.

Issue #12	**`registerLibrary` will seldom revert with `UNSUPPORTED_INTERFACE` if a bad interface is provided, instead it will revert ambiguously**
Severity	● INFORMATIONAL
Location	L118 `require(IERC165(_lib).supportsInterface(type(IMessageLib).interfaceId), Errors.UNSUPPORTED_INTERFACE);`
Description	If `_lib` does not provide a function called `supportsInterface` which returns the expected data type (a boolean), Solidity will panic due to the return length being different. Since most contracts do not support IERC165, we expect this case to happen most often and this error will therefore not be used very often. The implications of this are minimal but it is certainly an imperfection.
Recommendation	Consider using `ERC165Checker` by OpenZeppelin. This library does a return length check and handles other cases like excessive gas usage as well.
Resolution	● ACKNOWLEDGED

Issue #13	UI function `getRegisteredLibraries` can run out of gas
Severity	● INFORMATIONAL

Location	`Line 55`
	`function getRegisteredLibraries() external view returns (address[] memory)`

Description	The contract contains functionality that, due to their implementation nature, can revert. As the state of the contract expands, this functionality might become so expensive that the gas cost does not fit in a single block and would become impossible to call. As RPCs also have various rate limiting methods, the functionality might become inaccessible even sooner.
Recommendation	Consider redesigning the functionality mentioned above in a way where it can be executed with a fixed gas cost.
Resolution	● ACKNOWLEDGED

Issue #14	`DefaultReceiveLibraryTimeoutSet` event is inconsistent between various functions
Severity	● INFORMATIONAL
Description	`DefaultReceiveLibraryTimeoutSet` event is always emitted within `setDefaultReceiveLibraryTimeout` while within it is only emitted within `setDefaultReceiveLibrary` in one of the two branches. This behavioral difference appears accidental and is inconsistent.
Recommendation	Consider adjusting either function to make the event behavior consistent with one another.
Resolution	✓ RESOLVED

Issue #15	**`snapshotConfig` and `resetConfig` can likely be abstracted from the message lib interface**
Severity	● INFORMATIONAL
Description	The message lib interface needs to define two functions: `snapshotConfig` and `resetConfig`. It appears likely to us that at the very least, `resetConfig` can simply be implemented at the endpoint level instead of requiring a message lib implementation. For example, an implementation could likely be constructed at the endpoint level using the existing `lib.getDefaultConfig` and `lib.setConfig`. Note that this issue is not a concern and only a small design recommendation to carefully consider whether these two interface functions can potentially be removed to simplify the codebase a bit at the library level.
Recommendation	Consider whether these two functions can be removed at the library level and replaced with general functions at the endpoint function. This reduces the overall level of cost in the codebase.
Resolution	✓ RESOLVED These functions have been removed in favor of a more elegant configuration system introduced during the updates.

Issue #16	**Typographical issues**
Severity	● INFORMATIONAL

Description	L82
	`/// @dev called when the endpoint checks if the msglib attempting to delever the msg is the configured msglib of the Oapp`
	
	"deliver" is mis-spelled as "delever".
	
	L104-110
	`if (timeout.lib == _actualReceiveLib && timeout.expiry > block.timestamp) {`
	`    // timeout lib set and has not expired`
	`    return true;`
	`}`
	
	`// returns false by default`
	`return false;`
	
	Branching is unnecessary here, the `if` statement can just be returned.
	
	——
	
	The whole file lacks type usage for addresses. Often times address parameters can be cast to example `IMessageLib` to make them more semantically correct and avoid casting.

Recommendation	Consider fixing the typographical issues.
Resolution	● PARTIALLY RESOLVED The first typographical issue was fixed.

2.4 BlockedMessageLib

`BlockedMessageLib` is an emergency library which can be set as a sender or receiver library to block all message flow fully. It reverts on all interactions.

UPDATE: During the four updates, this contract has effectively remained unchanged other than it declaring its message lib type.

2.4.1 Issues & Recommendations

Issue #17	**`BlockedLibrary` can never be set**
Severity	● MEDIUM SEVERITY
Description	`BlockedLibrary` does not appear to implement the `isSupportedEid` function. This causes all library sets to fully revert if an app tries to set the blocked library as a send or receive library. The blocked library can therefore not be used at all by apps on a discretionary basis. Even setting it as a default library appears impossible.
Recommendation	Consider implementing the necessary function on the `BlockedLibrary`.
Resolution	✓ RESOLVED The missing function has been added.

Issue #18	**Fallback should implement `payable` to also revert on the correct error if gas tokens are sent**
Severity	● INFORMATIONAL
Description	The contract reverts with a non-implemented error for any function called which is not implemented. However, if such a function is called with some gas attached, the contract will revert ambiguously instead.
Recommendation	Consider adding `payable` to the fallback.
Resolution	● ACKNOWLEDGED

2.5 MessagingChannel

`MessagingChannel` is a dependency used within `EndpointV2` to keep track of the inbound and outbound nonces.

When sending outbound messages, which is initiated by calling send or `sendWithAlt` on the endpoint, a simple incremental nonce is used. This nonce starts at 1 for the first message and is incremented by 1 every time a message gets send.

These nonces for each combination of `{ sender, receiver, dstEid, srcEid }` are being kept track of. Note that within the sending endpoint, there is no `srcEid` parameter for `outboundNonce`. This is because the `srcEid` is simply the endpoint id of that endpoint. This is also why the inbound nonces do not have a `dstEid` parameter within the endpoint, as that is the `eid` of that endpoint.

To summarize sending: each time an OAPP calls `send` (or `sendWithAlt`) on the endpoint, a nonce is generated. This nonce is always one greater than the previous nonce used for the `{ sender, receiver, dstEid, srcEid }` combination.

When this nonce arrives at the destination endpoint, the corresponding payload gets stored in the corresponding `{ receiver, sender, srcEid }` mapping called `inboundPayloadHash` on the recipient endpoint.

It should finally be noted that `MessagingChannel` enforces the execution and receipt order of messages. Unlike the original LayerZero design, this order enforcement has been loosened:

- Messages can be delivered in any order.
- Messages can be executed in any order, but only if all previous nonces have been delivered already.

- Messages can only be re-delivered if no subsequent messages (or the message itself) have yet been executed .

UPDATE 1: After the initial resolution round, this contract was modified. Basic modifications were made, such as removing the Origin struct from function arguments in favor of the underlying values. It also replaced the `_getGuid` function with a global library function `GUID.generate`.

UPDATE 2: After the second round, this contract was modified. `lazyInboundHash` was made `public` which is fine and added `unverify()` which is more intrusive. `unverify` effectively allows verification DVNs to mark their message as no longer verified.

UPDATE 3: After the third round, this contract was modified. `_inboundVerifiable` was removed.

UPDATE 4: After the fourth round, a large number of changes were made. The inbound function for verification now explicitly prevents `EMPTY_PAYLOAD_HASH`, which is good as it reduces the state space and this value was being used as a special type. The authorization pattern from the endpoint has been introduced to functions such as `skip`. `unverify` has been split up into `burn` and `nillify`, an intrusive change which allows verifiers to mark a message as burned or nillified.

2.5.1 Privileged Functions

- `skip [ oapp ]`

2.5.2 Issues & Recommendations

Issue #19	**`inboundNonce` might occasionally revert ambiguously due to running out-of-gas**
Severity	● LOW SEVERITY
Description	The only way to figure out whether `inboundNonce` will run out of gas is by optimistically calling it and assuming the revert is due to gas issues. This seems like a severe limitation on the interface and we recommend adding a safer interface which is consistent even with larger jumps in the nonce space. This issue is only rated as low as the actual critical paths allow for execution even if `inboundNonce` temporarily reverts due to gas issues. This is because the `lazyInbounNonce` moves up over time with message delivery, allowing for `inboundNonce` to run with less gas again.
Recommendation	Consider adding an alternative safe function `inboundNonceCapped(..., uint256 cap)` which returns a boolean if the cap was exceeded. This function might be useful for third-party apps, even if its not needed right now by LayerZero.
Resolution	● ACKNOWLEDGED The client has indicated that the executor will work to keep this in appropriate magnitude.

Issue #20	Lack of validation: `_payloadHash` lacks a non-zero check within `_inbound` which can lead to extra business logic errors with bad receive libraries
Severity	● INFORMATIONAL
Description	The `_inbound` function is called whenever a message is delivered to the endpoint. One of its parameters is the hash of the payload. This hash should never be all zero bytes since this hash is considered a "non-existent" payload. Although it is statistically improbable (technically impossible) enough for such a hash to ever be generated, a bad receipt library might call this function with such a parameter regardless. If this happens, this might cause additional business logic error at the relayer/receipt library level if it's not reverted. This issue has been rated as informational since libraries need to be correct for the system to function anyway.
Recommendation	Consider whether it makes sense to add a non-zero check to this parameter to more formally compartmentalize the zero bytes payload as the non-existent payload. This might even just be an assertion as to make the contract more formally correct. This issue will be resolved regardless however, as a counter-argument of gas-cost for an assertion which will never revert in practice with a good library is understandable as well.
Resolution	✓ RESOLVED The client has indicated they do not want to validate this for gas reasons. No changes have been made and we accept this argument.

Issue #21	Gas optimizations
Severity	● INFORMATIONAL

Description	L125-126
	`bytes memory path = abi.encodePacked(eid, _sender.toBytes32(), _dstEid, _receiver);`
	`guid = keccak256(abi.encodePacked(_nonce, path));`
	
	Using `abi.encodePacked` twice here appears like a needless waste of gas on the compiler settings we experimented with.
	
	——
	
	`_clearPayload` appears to read the storage slot for the slot which will be cleared twice: once when confirming the hash is equal to the actual hash and once within the for loop to ensure that the hash does not equal zero. This can be optimized.

Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	● PARTIALLY RESOLVED The first recommendation has been resolved.

Issue #22	Typographical issues
Severity	● INFORMATIONAL

Description	L74
	`/// @dev after skipping the lazyInboundNonce is set to the provided nonce, which makes the inboundNonce also the provided nonce`
	
	A comma should be added after "after skipping".
	
	L92
	`// require the _nonce to be no greater than the inboundNonce`
	
	This line might be outdated as this does not appear to be what is enforced in the subsequent code.

Recommendation	Consider fixing the typographical issues.
Resolution	✓ RESOLVED

2.6 MessagingComposer

`MessagingComposer` is a component of the endpoint which allows apps to effectively let `lzCompose` messages be executed on the chain itself to any address they desire. The interesting thing about this is that an app can queue a message, which can then be executed in a later function.

Though not particularly useful on the EVM since apps can simply call the recipient directly, this feature was added for compatibility with chains which do not have dynamic dispatch. On these chains, a composer like this is of course invaluable.

UPDATE 1: After the initial resolution round, this contract was modified. An index was added to the namespace of messages. The function calling the destination no longer does try-catch behavior, and simply reverts on failure. We commend this change as it reduces the attack surface greatly.

UPDATE 2: After the second round, an event was renamed.

2.6.1 Issues & Recommendations

Issue #23	**`lzCompose` will not revert if called to an address without any code**
Severity	● LOW SEVERITY
Description	`safeCall` does not check that the address called has code. This means that an `lzCompose` to an address without any code will not revert and will succeed.
Recommendation	Consider whether this is desired (e.g. maybe for gas transfers to EOAs), if not, the SafeCall lib would need to be updated to check this edge case.
Resolution	✓ RESOLVED The client has indicated this is fine with them. No changes have been made.

Issue #24	**`lzCompose` no longer adheres to checks-effects-interactions, making the code less verifiable**
Severity	● LOW SEVERITY
Description	The client has made a post-audit adjustment where the refund logic for failure within the `lzCompose` function is now executed before the `composedMessage` is re-enabled. Though we validated that there is no user risk in this change to the best of our knowledge, moving away from checks-effects-interactions makes the codebase more error prone and makes validation of its safety more difficult.
Recommendation	Consider reverting to checks-effects-interactions
Resolution	✓ RESOLVED The contract no longer relies on `try` behavior, which is great. This change was made in one of the subsequent rounds.

Issue #25	**`lzCompose` can be called multiple times if the hash of the message is `bytes32(1)`**
Severity	● INFORMATIONAL
Description	As the replay protection is only done by checking that the hash stored and the one recalculated are the same, it is possible for a message that is hashed to `bytes32(1)` to be replayed multiple times. This scenario is very unlikely, but could theoretically happen. This issue is 100% theoretical as LayerZero has bigger problems if targeting such hashes is possible. However, for the sake of formal verification, it may be nice to add an assertion.
Recommendation	To make this formally impossible, a simple check in `deliverComposedMessage` that the hash is not `bytes32(1)` would suffice. Such assertions might be valuable throughout the codebase if formal validity is desired. Alternatively, this issue can be resolved without changes given that not being able to target specific hashes like 1 is a fair assumption to make.
Resolution	✓ RESOLVED The client has indicated that they would not like to include this edge-case given the gas consumption and given that it is sufficiently improbable to never occur this issue has been marked as resolved.

2.7 MessagingContext

`MessagingContext` is a dependency inherited within `EndpointV2`. It includes the simple logic to store the sender and destination id of any ongoing outbound transaction in storage.

This dependency also serves the purpose of a reentrancy guard for outbound transactions, as a check is made to ensure that the send context has been reset before a new outbound message can be sent.

Specifically, `MessagingContext` defines a modifier `sendContext` which stores the destination EID and sender address in the `_sendContext` storage slot. If a value was already stored, this modifier reverts, preventing reentrancy into the send functions.

UPDATE 1: After the initial resolution round, this contract was modified. Non-intrusive changes such as requirement adjustments were made, and this contract was marked as abstract.

2.7.1 Issues & Recommendations

Issue #26	**Typographical issue**
Severity	● INFORMATIONAL
Location	L9 `/// the context includes the remote eid and the sender/ receiver address`
Description	This comment appears to be incorrect as the receiver address is not stored within the context slot.
Recommendation	Consider fixing the typographical issue.
Resolution	✓ RESOLVED

Issue #27	**`address(1)` can theoretically re-enter into the contract, though practically impossible as this address is not expected to ever be owned**
Severity	● INFORMATIONAL
Description	From a formal verification perspective, it may make sense to prevent `address(1)` from ever using the context modifier. This way, it becomes formally guaranteed that the active context does not collide with `NOT_ENTERED`.

Recommendation	Consider adjusting the modifier if desired:
	`uint newContext =  (uint(_dstEid) << 160) | uint160(_sender);`
	`require(newContext != _NOT_ENTERED);`
	`_sendContext = newContext;`
	`_;`
	`_sendContext = _NOT_ENTERED;`

Resolution	✓ RESOLVED The client has decided not to modify this as it is realistically impossible to occur. We agree with this statement.

Issue #28	Gas optimizations
Severity	● INFORMATIONAL

Location	L29 - 31
	`function getSendContext() external view returns (uint32, address) {`
	`    return isSendingMessage() ? (uint32(_sendContext >> 160), address(uint160(_sendContext))) : (0, address(0));`
	`}`

Description	`_sendContext` could be stored to save gas. To save even more gas, one could update `isSendingMessage` to use an `internal` function that takes the value as an input so that this value can be cached and read only once.
Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	✓ RESOLVED

2.8 AddressCast

`AddressCast` is a small set of utility functions to cast variables between different types, used by the various portions of the system.

UPDATE 1: After the resolution round, changes were made to this contract. The changes were not intrusive and limited to error reporting being changed.

2.8.1 Issues & Recommendations

Issue #29	**`toBytes` assembly block erroneously accesses a potentially misconfigured memory slot resulting in potentially dirty bytes**
Severity	● INFORMATIONAL
Description	The `toBytes` assembly block shifts the bytes of the address to the left to line them up with a shorter byte size. However, dirty bytes of a next memory slot can be shifted within this assembly block. This is not a huge concern but might cause issues if the resulting variable is accessed through assembly again. We raise this issue explicitly to remind the development team that there are better methods of writing functions like this which do not need to access adjacent memory.
Recommendation	Consider implementing a redesigned assembly block similar to the one recommended in the gas optimizations below.
Resolution	✓ RESOLVED

Issue #30	Gas optimizations
Severity	INFORMATIONAL

Location	L21 - 31
	`function toBytes(bytes32 _addressBytes32, uint256 _size) internal pure returns (bytes memory result) {`
	`require(_size > 0 && _size <= 32, Errors.INVALID_SIZE);`
	`bytes memory data = abi.encodePacked(_addressBytes32); // copy to memory`
	`result = new bytes(_size);`
	`unchecked {`
	`uint256 offset = 64 - _size; // 32 + 32`
	`assembly {`
	`mstore(add(result, 32), mload(add(data, offset)))`
	`}`
	`}`
	`}`

Description	The `toBytes` function first copies the `bytes32` data to memory before copying it again and taking care of the offset. These two operations can be done simultaneously to save some gas.
	`function toBytes(bytes32 _addressBytes32, uint256 _size) internal pure returns (bytes memory result) {`
	`require(_size > 0 && _size <= 32, Errors.INVALID_SIZE);`
	`result = new bytes(_size);`
	`unchecked {`
	`uint256 offset = 256 - _size * 8;`
	`assembly {`
	`mstore(add(result, 32), shl(offset, _addressBytes32))`
	`}`
	`}`
	`}`

Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	✓ RESOLVED

2.9 SafeCall

`SafeCall` addresses a vulnerability within Solidity when calls are made to untrusted contracts. In these cases, the untrusted contracts can return data which cause the gas limitation to fail. This is solved within this library by not loading all of the returned data if it is larger than 100 bytes. This library was copied from https://github.com/nomad-xyz/ExcessivelySafeCall/blob/main/src/ExcessivelySafeCall.sol.

2.9.1 Issues & Recommendations

Issue #31	Outdated comment
Severity	INFORMATIONAL
Location	L39 `// limit our copy to 256 bytes`
Description	The comment seems outdated as it only copies 100 bytes.
Recommendation	Consider updating the comment.
Resolution	✓ RESOLVED

Issue #32	Gas optimization
Severity	INFORMATIONAL
Location	L22 `uint16 _maxCopy = 100;`
Description	`_maxCopy` could be a constant to save a bit of gas and make the code cleaner.
Recommendation	Consider implementing the gas optimization mentioned above.
Resolution	✓ RESOLVED

2.10 BitMaps

`BitMaps` allows for packing up to 256 bits within a single uint256 variable, encodable and decodable by a 0-255 index.

2.10.1 Issues & Recommendations

Issue #33	Gas optimization
Severity	INFORMATIONAL
Location	L13 & 21 `uint256 mask = 1 << (index & 0xff);`
Description	As the index variable is already `uint8`, `index & 0xff` is redundant. Consider removing this unnecessary operation. Note that cleaning of dirty bits is an extremely good and necessary practice, but to the best of our knowledge only necessary within assembly blocks, as Solidity already cleans these where necessary.
Recommendation	Consider implementing the gas optimization mentioned above.
Resolution	✓ RESOLVED

2.11 PacketV1Codec

`PacketV1Codec` can be used to encode and decode LayerZero packets. It is not used within the codebase as the audit covers the V2 codebase.

UPDATE 1: After the initial resolution round, changes to this contract were made. The packet version is now hardcoded to 1 and the encode2 function has been split up into an encodeHeader and encodePayload function. Both of these changes are not considered intrusive.

2.11.1 Issues & Recommendations

No issues found.

PALADIN
BLOCKCHAIN SECURITY