PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For LayerZero (HyperLiquid Composer)

21 April 2025

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 HyperLiquid composer contracts on the Hyperliquid 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	Hyperliquid
Language	Solidity
Preliminary Contracts	https://github.com/LayerZero-Labs/devtools/tree/cdc138d29371271a7ff8d09858ed09afad07ce35/packages/hyperliquid-composer/contracts
Resolution #1	https://github.com/LayerZero-Labs/devtools/tree/8e22aaa6004a0b6d8498a7c305cc04b5878b2d16
Resolution #2	https://github.com/LayerZero-Labs/devtools/commit/f383a5099d494e12ab242b85af9be96a04a66d2b

1.2 Contracts Assessed

Name	Contract	Live Code Match
`HyperLiquidComposer`		N/A
`HyperLiquidComposerCore`		N/A
`HyperLiquidComposerCodec`		N/A

The contracts will be deployed by Layer Zero's partners, so users should check the deployed contracts against the audited GitHub commit in this report.

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0	-	-	-
Medium	0	-	-	-
Low	4	4	-	-
Informational	5	5	-	-
Total	**9**	**9**	**-**	-

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 HyperLiquidComposer

ID	Severity	Summary	Status
01	LOW	Using `transfer` instead of `safeTransfer` for ERC20 tokens	RESOLVED
02	INFO	Insufficient validation	RESOLVED
03	INFO	Refunding `tx.origin` instead of the executor argument in the `lzCompose` function	RESOLVED
04	INFO	Typographical issues	RESOLVED

1.3.2 HyperLiquidComposerCore

ID	Severity	Summary	Status
05	LOW	Using `transfer` instead of `safeTransfer` for ERC20 tokens	RESOLVED
06	INFO	Insufficient validation	RESOLVED
07	INFO	Typographical issues	RESOLVED

1.3.3 HyperLiquidComposerCodec

ID	Severity	Summary	Status
08	LOW	Not all decimal differences are considered during transfers	RESOLVED
09	LOW	It is assumed that `amount / scale` is lower than `type(uint64).max`	RESOLVED

2 Findings

2.1 HyperLiquidComposer

`HyperLiquidComposer` is the entrypoint of the `lzCompose` call.

2.1.1 Privileged Functions

- `refundERC20[onlyComposer]`
- `refundNativeTokens[onlyComposer]`

2.1.2 Issues & Recommendations

Issue #01	Using `transfer` instead of `safeTransfer` for ERC20 tokens
Severity	LOW SEVERITY
Description	In `_sendAssetToHyperCore()`, ERC20 transfers are executed by calling `transfer()`. It is always recommended to use OpenZeppelin's `safeTransfer()` method as it properly handles interactions with unconventional tokens (such as USDT) which might revert through `transfer()`.
Recommendation	Use `safeTransfer()` instead of the `transfer()` when sending ERC20 tokens.
Resolution	RESOLVED

Issue #02	Insufficient validation
Severity	INFORMATIONAL
Description	Within `_sendAssetToHyperCore()`, consider checking if amounts.evm is zero when calling `quoteHyperCoreAmount` before calling `transfer` (such as when the transfer is in amount lower than `decimalDiff`) to evade potential scenarios where `transfer` can revert for zero amounts on some tokens. Also check if `amounts.core` is zero and if it is, skip the `sendSpot()` call in cases where there are only dust amounts that should be refunded.
Recommendation	Consider implementing the above recommendations.
Resolution	RESOLVED

Issue #03	**Refunding `tx.origin` instead of the executor argument in the `lzCompose` function**
Severity	INFORMATIONAL
Description	There is an edge case where the LayerZeroV2 `endpoint.lzCompose()` could be called by a contract X that needs the signatures of a couple of admins before a function on this contract X could call `endpoint.lzCompose()` successfully and send native funds with the call. The function of contract X could be called by a random user by providing the admin signatures or gathering enough votes for the function to become executable. As HyperLiquidComposer refunds `tx.origin`, this user would get the native assets in case of a refund.
Recommendation	Consider using the executor that is currently commented out in `HyperLiquidComposer.lzCompose()` to refund the native assets instead of `tx.origin` if you think such a scenario is likely to be problematic.
Resolution	RESOLVED

Issue #04	Typographical issues
Severity	INFORMATIONAL

Description	`/// @dev If the addresses are invalid, the function will emit an error message and try a complete refund to the receiver else the sender`
	
	should be
	
	`If the receiver is invalid, the function will emit an error message and try a complete refund to the sender.`

Recommendation	Consider fixing the typographical issues.
Resolution	RESOLVED

2.2 HyperLiquidComposerCore

`HyperLiquidComposerCore` is the core implementation of the Composer that handles validation of inputs, addresses, quoting hyper core balances, and refunds tokens.

2.2.1 Privileged Functions

- `refundERC20[onlyComposer]`
- `refundNativeTokens[onlyComposer]`

2.2.2 Issues & Recommendations

Issue #05	Using `transfer` instead of `safeTransfer` for ERC20 tokens
Severity	LOW SEVERITY
Description	In `_sendAssetToHyperCore()`, ERC20 transfers are executed by calling `transfer()`. It is always recommended to use OpenZeppelin's `safeTransfer()` method as it properly handles interactions with unconventional tokens (such as USDT) which might revert through `transfer()`.
Recommendation	Use `safeTransfer()` instead of the plain `transfer()` when sending ERC20 tokens.
Resolution	RESOLVED

Issue #06	Insufficient validation
Severity	INFORMATIONAL
Description	Within the constructor, ensure there is a check that `_endpoint` is not `address(0)`. — `refundERC20()` does not need the `payable` modifier.
Recommendation	Consider implementing the above recommendations.
Resolution	RESOLVED

Issue #07	Typographical issues
Severity	INFORMATIONAL
Description	`/// @notice Refunds the native tokens to the refund address` This should refer to ERC20, not native tokens. — `VALID_COMPOSE_MESSAGE_LENGTH_PACKED` and `VALID_COMPOSE_MESSAGE_LENGTH_ENCODE` could be removed since they are not used inside `HyperLiquidComposer` and `HyperLiquidComposerCore`.
Recommendation	Consider fixing the typographical issues.
Resolution	RESOLVED

2.3 HyperLiquidComposerCodec

`HyperLiquidComposerCodec` is a library used for the conversion of types (asset bridge address, hyper core asset value) and the validation of EVM addresses.

2.3.1 Issues & Recommendations

Issue #08	Not all decimal differences are considered during transfers
Severity	LOW SEVERITY

Description	Within `into_hyperAssetAmount()`, the following calculation is performed:
	`uint256 scale = 10 ** _asset.decimalDiff;`
	…
	`uint64 amountCore = uint64(_amount / scale);`
	…
	
	The `decimalDiff` variable is always used with the assumption that asset decimals on the EVM would be bigger or equal than the HIP token on Core.
	
	Proper transfers are not possible if the situation was reversed, for example if `Token1` has 6 decimals on EVM and 10 decimals on Core.
	
	There is no way to configure the contract for such a case.

Recommendation	Consider if it is a plausible scenario where assets can have more decimals on Core than on EVM and if yes, update the logic or explicitly warn in the docs that such cases are not supported.
Resolution	RESOLVED

Issue #09	It is assumed that amount / scale is lower than `type(uint64).max`
Severity	LOW SEVERITY
Description	In cases where the decimal difference is 0 (e.g. HyperCore and HyperEVM assets have the same decimals) then `scale` within `into_hyperAssetAmount()` would be 1. If the token is in 18 decimals then it would be very easy for the user to transfer more than `type(uint64).max` — 20e18 would be enough. In this case on L65, the direct conversion from `uint256` to `uint64` would be unsafe (reducing `_amount` to `type(uint64).max`). This would cause `sendSpot()` to be called with less than the token amount that would be transferred to the asset bridge. This issue is labeled as low since it should be unlikely that a token with 18 decimals on the EVM side would not be configured to have a lower amount of decimals on Core. Still it would be safer if `_amount` / `scale` is not converted into `uint64` without checking if the result is indeed less than `uint64`.
Recommendation	Consider refunding all EVM funds to the sender or receiver if the result of `_amount` / `scale` is more than `type(uint64).max`.
Resolution	RESOLVED

PALADIN
BLOCKCHAIN SECURITY