PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For LayerZero (HyperLiquid FeeActivation)

13 November 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 FeeActivation contracts on the Ethereum network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	LayerZero
URL	https://layerzero.network/
Platform	Ethereum
Language	Solidity
Preliminary Contracts	https://github.com/LayerZero-Labs/devtools/tree/da5a74ced6ae95d5fdf71d45667dbf96c1059c94/packages/hyperliquid-composer/contracts
Resolution #1	https://github.com/LayerZero-Labs/devtools/pull/1774/commits/9eee4c32de854d0cc2590e13ca49129577925ae9
Resolution #2	https://github.com/LayerZero-Labs/devtools/pull/1801/commits/badaf8350c45cc81f7acf57eaefd33d4521562b6

1.2 Contracts Assessed

- `PrefundedFeeAbstraction`
- `RecoverableComposer`

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0	-	-	-
Medium	0	-	-	-
Low	3	3	-	-
Informational	6	6	-	-
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 PrefundedFeeAbstraction

ID	Severity	Summary	Status
01	LOW	`retrieveCoreUSDC` can cause silent failures	✓ RESOLVED
02	LOW	`MAX_USERS_PER_BLOCK` assumes that the maximum transaction count per block will not change	✓ RESOLVED
03	INFO	`MAX_USERS_PER_BLOCK * QUOTE_ASSET_DECIMALS` can be stored in a variable to save gas	✓ RESOLVED
04	INFO	Same revert reason used for two distinct errors	✓ RESOLVED
05	INFO	Documentation does not match the code	✓ RESOLVED
06	INFO	In cases of extreme prices, activation fee might round to 0	✓ RESOLVED
07	INFO	Precision loss due to division before multiplication	✓ RESOLVED

1.3.2 RecoverableComposer

ID	Severity	Summary	Status
08	LOW	Missing `retrieve` function for the quote token	✓ RESOLVED

1.3.3 IPreFundedFeeAbstraction

ID	Severity	Summary	Status
09	INFO	`FeeCollected` event emits wrong units	✓ RESOLVED

2 Findings

2.1 PrefundedFeeAbstraction

`PrefundedFeeAbstraction` is an extension of the Composer that automatically handles HyperCore activations for new users. It lets recipients receive tokens on HyperCore even if they have never been activated before by paying the activation fee from the contract's own pre-funded quote balance (for example, USDC on Core).

.

2.1.1 Issues & Recommendations

Issue #01	`retrieveCoreUSDC` can cause silent failures
Severity	LOW SEVERITY
Location	`if (coreBalance < (MAX_USERS_PER_BLOCK * QUOTE_ASSET_DECIMALS)) { revert InsufficientCoreAmountForActivation(); }`
Description	As `spotBalance` gives the balance at the start of the block, the following scenario in the same block would cause a silent failure. - There is 100 USDC in the composer contract. - At the start of the block, `spotBalance` will return 100 USDC. - Tx 1 in block: `RecoveryAddress` calls `retrieveCoreUSDC(100 USDC, to)`. - Tx 2 in block: User's bridging transaction which will cost 1 USDC in activation fees. This will cause this check to not revert, but because transaction 2 comes after transaction 1, it could silently revert.
Recommendation	To prevent this, `retrieveCoreUSDC` should only be callable when the contract is paused and no bridging transactions can come through.
Resolution	RESOLVED

Issue #02	**MAX_USERS_PER_BLOCK assumes that the maximum transaction count per block will not change**
Severity	LOW SEVERITY
Description	In the case there is a change in the future that allows more than 50 tx per block this will be insufficient for the coreBalance check.
Recommendation	Consider making it mutable, with a minimum of 50.
Resolution	RESOLVED

Issue #03	**MAX_USERS_PER_BLOCK * QUOTE_ASSET_DECIMALS can be stored in a variable to save gas**
Severity	INFORMATIONAL
Description	`MAX_USERS_PER_BLOCK * QUOTE_ASSET_DECIMALS` is calculated each time `_transferERC20ToHyperCore()` gets called, consuming unnecessary gas.
Recommendation	Consider storing `MAX_USERS_PER_BLOCK * QUOTE_ASSET_DECIMALS` as an immutable variable.
Resolution	RESOLVED

Issue #04	Same revert reason used for two distinct errors
Severity	INFORMATIONAL
Description	Both of the following failure paths revert with the same `InsufficientCoreAmountForActivation()` error: 1. In `FeeToken._getFinalCoreAmount()` when `_coreAmount <= activationFee()` user didn't send enough to cover activation. 2. In `_transferERC20ToHyperCore()` when the composer's quote pre-fund is below the block-level threshold - operator under-funded the contract. These are semantically unrelated errors but share the same revert identifier, making it impossible to distinguish the cause on-chain or in monitoring.
Recommendation	We recommend introducing explicit, separate errors for both cases.
Resolution	RESOLVED

Issue #05	Documentation does not match the code
Severity	INFORMATIONAL
Description	In the documentation, it is written: *"The min fund amount is a suggestion and not enforced in this implementation. The recovery address can call retrieveUSDC for the entire amount to empty the contract. This is expected behavior."* However, this is enforced by the hardcoded constant of 50 of `MAX_USERS_PER_BLOCK`.
Recommendation	Consider updating the documentation.
Resolution	RESOLVED

Issue #06	In cases of extreme prices, activation fee might round to 0
Severity	INFORMATIONAL
Description	If the spot price (`rawPrice`) becomes large enough, integer division floors this value to zero. When that happens, an unactivated user still requires activation on Core, but the prefund guard is never executed, so the contract might proceed without ensuring enough quote balance. If the Core quote balance is too low, the underlying `CoreWriter` transfer can fail or silently revert.
Recommendation	We acknowledge that this is a very extreme and highly unlikely requiring the token price to be in the tens or hundreds of millions. We recommend reverting which will trigger a refund if the activation fee turns out to be 0 to mitigate such extreme cases.
Resolution	RESOLVED

Issue #07	Precision loss due to division before multiplication
Severity	INFORMATIONAL
Location	`ACTIVATION_COST_ASSET = uint64((totalCentsAmount * 10 ** baseAssetInfo.weiDecimals) / 100);` `ACTIVATION_FEE_WEI = ACTIVATION_COST_ASSET * SPOT_PRICE_DECIMALS;`
Description	There might be a loss of precision in the calculation on the second line as the value might be rounded down.
Recommendation	Consider performing multiplication before division to avoid precision loss.
Resolution	RESOLVED OpenZeppelin's `SafeMath` was implemented since `activationFee()` which does division would round down.

2.2 RecoverableComposer

`RecoverableComposer` is an extension that allows a privileged address to manually move assets from the composer on the HyperEVM and HyperCore side.

2.2.1 Privileged Functions

- `retrieveCoreERC20[recoveryAddress]`
- `retrieveCoreHYPE[recoveryAddress]`
- `retrieveCoreUSDC[recoveryAddress]`
- `recoverEvmERC20[recoveryAddress]`
- `recoverEvmNative[recoveryAddress]`

2.2.3 Issues & Recommendations

Issue #07	Missing `retrieve` function for the quote token
Severity	LOW SEVERITY
Description	Only USDC can be retrieved — if the quote token is either USDT or USDH, the quote token cannot be retrieved.
Recommendation	Consider adding a `retrieve` function for the quote token.
Resolution	RESOLVED

2.3 IPreFundedFeeAbstraction

`RecoverableComposer` is an interface contract with some `view` functions and errors.

2.3.3 Issues & Recommendations

Issue #08	**`FeeCollected` event emits incorrect units**
Severity	INFORMATIONAL

Description	The FeeCollected event is defined in IPreFundedFeeAbstraction as:
	`event FeeCollected(address indexed user, uint256 amount);`
	and documented as: "Activation fee collected in quote asset decimals."
	However, within `_transferERC20ToHyperCore`, the emitted value is:
	`uint64 feeCollected = originalAmount - coreAmount;`
	`emit FeeCollected(_to, feeCollected);`
	`originalAmount` and `coreAmount` are in core asset units, not in quote units.

Recommendation	We recommend updating the event's NatSpec and documentation to clarify that it emits asset-denominated fees.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY