PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For LayerZero (VaultComposer)

23 October 2025

paladinsec.co

info@paladinsec.co

Table of Contents

Table of Contents 2

Disclaimer 3

1 Overview 4

1.1 Summary 4

1.2 Contracts Assessed 4

1.3 Findings Summary 5

1.3.1 VaultComposerSync 6

1.3.2 VaultComposerSyncNative 6

2 Findings 7

2.1 VaultComposerSync 7

2.1.1 Issues & Recommendations 8

2.2 VaultComposerSyncNative 13

2.2.2 Issues & Recommendations 14

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 VaultComposer 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/9c5ec72fd01e05ffb6945a3683ee43137d7b496e/packages/ovault-evm
Resolution #1	https://github.com/LayerZero-Labs/devtools/pull/1766/commits/cc6cb7f94ceedb811a123a6499b79f897f0a710b https://github.com/LayerZero-Labs/devtools/pull/1766/commits/0577172e4c73225c3b3dd54bc77093e7b31b0627 https://github.com/LayerZero-Labs/devtools/pull/1766/commits/8c1dfa2af98e3a419428151f96ac1eee74d90025
Resolution #2	https://github.com/LayerZero-Labs/devtools/pull/1775
Resolution #3	https://github.com/LayerZero-Labs/devtools/pull/1778 The team added a minor change to wrap vault operations. Paladin checked the change and found no issues.

1.2 Contracts Assessed

- `VaultComposerSync`
- `VaultComposerSyncNative`

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0	-	-	-
Medium	0	-	-	-
Low	5	3	-	2
Informational	2	1	-	1
Total	**7**	**4**	-	**3**

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 VaultComposerSync

ID	Severity	Summary	Status
01	LOW	Malicious actors can make `lzCompose` always refund if it has a local chain as destination	RESOLVED
02	LOW	Local compose with `minMsgValue > 0` cannot be executed	ACKNOWLEDGED
03	LOW	Use of raw `approve` instead of `safeApprove`	RESOLVED
04	LOW	If insufficient `msg.value` is supplied for a remote `send` operation, it will be refunded instead of retried	ACKNOWLEDGED
05	INFO	The receive function is not needed for the non-native composer	RESOLVED

1.3.2 VaultComposerSyncNative

ID	Severity	Summary	Status
06	LOW	`depositNativeAndSend` is not `nonReentrant`	RESOLVED
07	INFO	Local asset redemption returns WETH, not ETH	ACKNOWLEDGED

2 Findings

2.1 VaultComposerSync

`VaultComposerSync` is a synchronous composer that lets users deposit ERC-20 assets into an ERC-4626 vault and send shares (or redeem shares for assets) across chains via LayerZero OFT, with slippage checks and automatic refunds on failures. It exposes `depositAndSend` and `redeemAndSend` as user entry points, and routes delivery locally or through OFT based on dstEid.

.

2.1.1 Issues & Recommendations

Issue #01	**Malicious actors can make `lzCompose` always refund if it has a local chain as destination**
Severity	LOW SEVERITY
Description	Calls to `lzCompose` where the final destination is the local chain require `msg.value` to be 0. If the remote sender does everything correctly and sets `minMsgValue` to 0 and there are no reverts due to slippage, their message can still be refunded by a malicious actor that frontruns executor then calls `lzCompose()` with enough `msg.value` required by the refund. As `msg.value` is more than 0, `_sendLocal` will revert, `catch` will be triggered, and a refund will be executed.
Recommendation	Consider not requiring 0 `msg.value` for `_sendLocal` to avoid this griefing vector and refunds due to slippage reverts should be able to be executed when the initial destination is the local chain.
Resolution	RESOLVED

Issue #02	Local compose with `minMsgValue > 0` cannot be executed
Severity	LOW SEVERITY
Description	If a `lzCompose` call is done from a source chain other than the hub and the `dstEid` is the `VAULT_EID`, it will revert if `minMsgValue` is a non-zero value even if there is enough `msg.value` sent to it due to a check in `_sendLocal`: `if (_msgValue > 0) revert NoMsgValueExpected();` Additionally, if `minMsgValue` is non-zero, `lzCompose` must be called with a non-zero `msg.value` or it will revert in orange check. This means that such `lzCompose` messages will never be executable. Any compose message with `dstEid == VAULT_EID` and `minMsgValue > 0` will be unexecutable and always result in a refund.
Recommendation	When `dstEid == VAULT_EID`, force `minMsgValue = 0` at message construction or add an early check to revert with a clear error if it is not.
Resolution	ACKNOWLEDGED

Issue #03	Use of raw `approve` instead of `safeApprove`
Severity	LOW SEVERITY
Description	The IERC20 `approve` is called directly in `_initializeAssetToken` and `_initializeShareToken`. While WETH and OZ ERC20s behave as expected, raw `approve` is less tolerant of non-standard ERC20s with, for example, non-standard return values. This is mostly a compatibility issue rather than a security bug given the expected tokens.
Recommendation	If broader token support is ever desired, switch to SafeERC20's `forceApprove` and `safeApprove` patterns for robustness.
Resolution	RESOLVED

Issue #04	**If insufficient `msg.value` is supplied for a remote send operation, it will be refunded instead of retried**
Severity	LOW SEVERITY
Description	Users can specify a minimum amount of `msg.value` and if the supplied `msg.value` is not enough, this check will trigger a revert in order for the operation to be retried: `if (bytes4(_err) == InsufficientMsgValue.selector) {` If `lzCompose` triggers a remote `send` operation but the user-supplied minimal `msg.value` amount is insufficient, the revert will not be caught by the above check as `handleCompose` will revert in `Endpoint.send`, resulting in a triggered refund instead of allowing the user to retry with bigger `msg.value` amount. Another impact of users not specifying enough minimum `msg.value` for a remote send is that a remote send could require a bigger `msg.value` amount due to `extraOptions` and bigger `composeMessage`. In such cases, the `Errors.LZ_InsufficientFee` revert should occur because of a small margin. When this triggers a refund, no compose message or extra options will be added in the `refundSendParam` and thus the `msg.value` being sent to the endpoint will have a bigger excess because of this. This excess will go to the executor of the compose message (`tx.origin`) as profit.
Recommendation	Consider allowing users to retry the compose message execution in case of `LZ_InsufficientFee` or document this behaviour as a known risk.
Resolution	ACKNOWLEDGED

Issue #05	The `receive` function is not needed for the non-native composer
Severity	INFORMATIONAL
Location	`receive() external payable virtual {}`
Description	Non-native composers should not be receiving native currency outside the defined payable functions.
Recommendation	Consider removing the `receive` function for the non-native composer.
Resolution	RESOLVED

2.2 VaultComposerSyncNative

`VaultComposerSyncNative` is a native-asset variant where the vault's asset is WETH and the asset OFT represents native ETH. It wraps ETH to WETH on compose and supports `depositNativeAndSend` so users can deposit with ETH directly. It unwraps WETH when sending the asset cross-chain so Stargate `NativePools` receive ETH.

2.2.2 Issues & Recommendations

Issue #06	`depositNativeAndSend` is not `nonReentrant`
Severity	LOW SEVERITY
Description	`depositNativeAndSend` is not marked `nonReentrant`, unlike its counterpart `depositAndSend` which is protected by `ReentrancyGuard`. This function calls external contracts — the WETH contract via `deposit`, and the vault via `_depositAndSend` — which could allow a malicious or reentrant callback to call `depositNativeAndSend` again before the first execution finishes
Recommendation	Apply the `nonReentrant` modifier to `depositNativeAndSend` to enforce the same reentrancy protection as other state-changing entry points. This will guard against any unforeseen reentrancy via external calls.
Resolution	RESOLVED

Issue #07	Local asset redemption returns WETH, not ETH
Severity	INFORMATIONAL
Description	In the native variant, redeeming shares with `dstEid == VAULT_EID` transfers the ERC-20 WETH to the recipient via the local path and it does not unwrap to ETH. On remote destinations, the asset leg is delivered as native ETH via the `NativePool` flow. This creates a behavioral mismatch — remote recipients get ETH, local recipients get WETH.
Recommendation	Consider documenting this behavior, or add an opt-in flag to unwrap to ETH for local sends when the recipient can accept native.
Resolution	ACKNOWLEDGED The team commented: "The user will get the vault asset token which is WETH. Transforming WETH to ETH is a special case since the composer is talking to an OFT that takes in ETH instead of WETH."

PALADIN
BLOCKCHAIN SECURITY