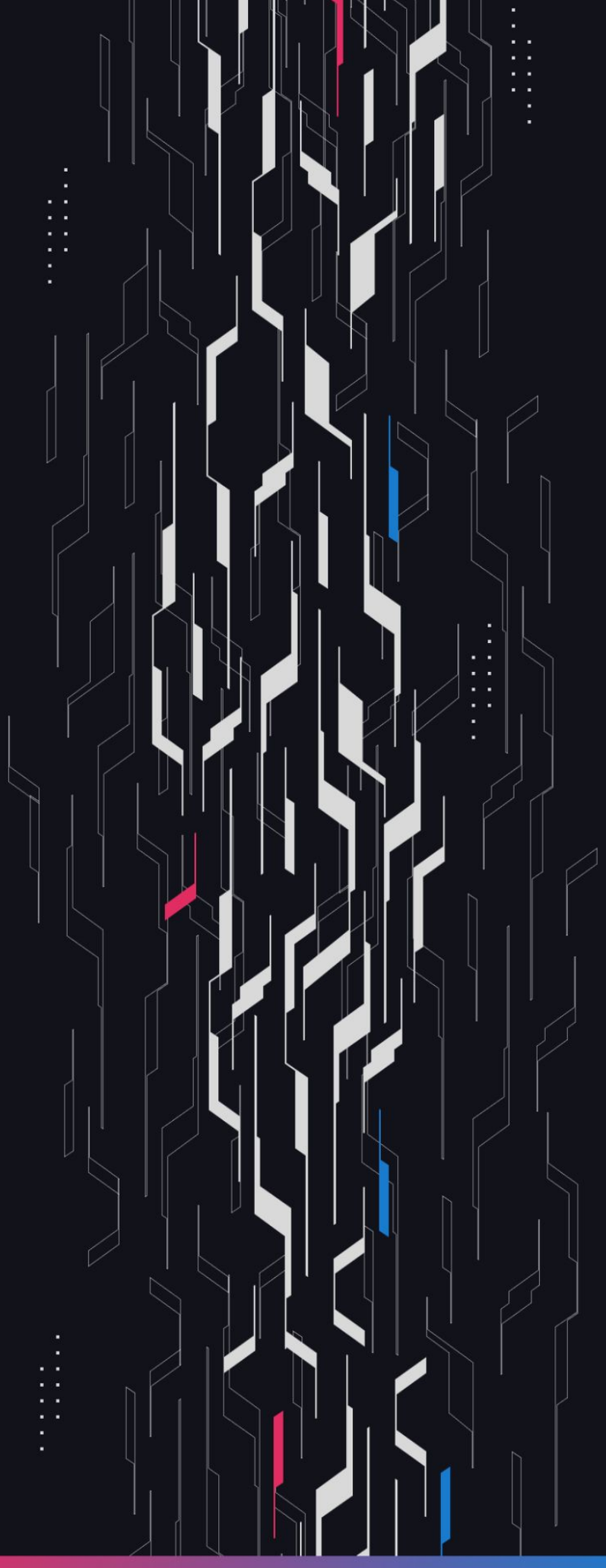
GA GUARDIAN

# Orderly

## Omnichain

## Security Assessment

July 1st, 2024



# Summary

**Audit Firm** Guardian

**Prepared By** Daniel Gelfand, Zdravko, Wafflemakr, Osman Ozdemir, Giraffe, Michael Lett

**Client Firm** Orderly

**Final Report Date** July 1st, 2024

## Audit Summary

Orderly engaged Guardian to review the security of its Omnichain contracts, supporting staking, vesting, and revenue claiming across multiple chains. From June 3rd to June 17th, a team of 6 auditors reviewed the source code in scope. All findings have been recorded in the following report.

I**ssues Detected** Throughout the engagement 10 High/Critical issues were uncovered and promptly remediated by the Orderly team. Several issues impacted the fundamental behavior of the protocol, following their remediation Guardian believes the protocol to uphold the functionality described for the Omnichain product.

**Security Recommendation** Given the number of High and Critical issues detected, Guardian supports an independent security review of the protocol at a finalized frozen commit. Furthermore, the Orderly team should drastically increase tests for cross-chain staking and revenue claims without the use of setters for key state variables. The engagement exposed multiple blind spots that should be thoroughly tested and presented numerous opportunities for system malfunction.

Notice that the examined smart contracts are not resistant to internal exploit. For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/omnichain-ledger-fuzz

# Table of Contents

# Project Overview

## Project Summary

| | |
|---|---|
| Project Name | Omnichain Ledger |
| Language | Solidity |
| Codebase | https://gitlab.com/orderlynetwork/orderly-v2/omnichain-ledger |
| Commit(s) | Initial Commit: d68be172d909441ee31674a6b3cd26b143adc8bd<br>Final Commit: d5d0f7a76406a2c72355c6d2e30e45dba80c3952 |

## Audit Summary

| | |
|---|---|
| Delivery Date | July 1, 2024 |
| Audit Methodology | Static Analysis, Manual Review, Test Suite, Contract Fuzzing |

## Vulnerability Summary

| Vulnerability Level | Total | Pending | Declined | Acknowledged | Partially Resolved | Resolved |
|---|---|---|---|---|---|---|
| Critical | 5 | 0 | 0 | 0 | 0 | 5 |
| High | 5 | 0 | 0 | 0 | 0 | 5 |
| Medium | 14 | 0 | 0 | 5 | 0 | 9 |
| Low | 17 | 0 | 0 | 7 | 0 | 10 |

# Audit Scope & Methodology

## Vulnerability Classifications

| **Severity** | Impact: *High* | Impact: *Medium* | Impact: *Low* |
|---|---|---|---|
| Likelihood: *High* | 🔴 Critical | 🟠 High | 🟡 Medium |
| Likelihood: *Medium* | 🟠 High | 🟡 Medium | 🟢 Low |
| Likelihood: *Low* | 🟡 Medium | 🟢 Low | 🟢 Low |

## Impact

**High** Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.

**Medium** A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.

**Low** Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

## Likelihood

**High** The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.

**Medium** An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.

**Low** Unlikely to ever occur in production.

# Audit Scope & Methodology

## Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

# Invariants Assessed

During Guardian's review of the Omnichain contracts, fuzz-testing with Foundry was performed on the protocol's main functions. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 1,000,000+ runs up to a depth of 500 with a prepared Foundry fuzzing suite.

| ID | Description | Tested | Passed | Remediation | Run Count |
|---|---|---|---|---|---|
| OC-01 | User staked ORDER & esORDER should never exceed total staked amount | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-02 | Total Valor Emitted Should Be Less Than Maximum Valor Emissions | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-03 | User Vault Chain ORDER Balance Should Increase By Amount Claimed On claimReward | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-04 | Sender Vault ORDER Balance Should Decrease By Amount When Staking | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-05 | Increments of valor emissions should Be Less Than or Equal To Total Valor Emitted When Interacting with Ledger Chain Balance | ✅ | ❌ | ✅ | 1,000,000+ |
| OC-06 | User Collected Valor Should Increase By Pending Valor When Interacting with Ledger Chain Balance | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-07 | Pending Valor Should Have Been Reset To 0 When Interacting with Ledger Chain Balance | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-08 | AccValorPerShareScaled Should Not Decrease When Interacting with Ledger Chain Balance | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-09 | Total Staked Amount Should Increase By Amount When Increasing Ledger ORDER/esORDER Balance | ✅ | ✅ | ✅ | 1,000,000+ |

# Invariants Assessed

| ID | Description | Tested | Passed | Remediation | Run Count |
|---|---|---|---|---|---|
| OC-10 | Total Staked Amount Should Decrease When Decreasing Ledger ORDER/esORDER Balance | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-11 | If isEsOrder is true User Staked Order Balance Should Stay The Same When Staking | ✅ | ✅ | N/A | 1,000,000+ |
| OC-12 | If isEsOrder is false User Staked Order Balance Should Increase When Staking | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-13 | User Staked Order Balance Should Decrease By Amount When Creating An Order Unstake Request | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-14 | If isEsOrder is true User Staked esOrder Balance Should Increase When Staking | ✅ | ✅ | N/A | 1,000,000+ |
| OC-15 | If isEsOrder is false User Staked esOrder Balance Should Stay The Same When Staking | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-16 | Pending Order Balance Should Increase By Amount When Creating An Order Unstake Request | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-17 | Unlock Timestamp After Should Equal block.timestamp + 7 days When Creating An Order Unstake Request | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-18 | User Ledger Order Balance Should Increase By Pending Order Balance Before When Canceling An Unstake Request | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-19 | Total Staked Amount Should Increase By Pending Order Balance Before When Canceling An Unstake Request | ✅ | ✅ | ✅ | 1,000,000+ |

# Invariants Assessed

| ID | Description | Tested | Passed | Remediation | Run Count |
|---|---|---|---|---|---|
| OC-20 | User Pending Order Balance Should Equal 0 When Canceling An Unstake Request/Withdrawing Order | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-21 | User Unlock Timestamp After Should Equal 0 When Canceling An Unstake Request/Withdrawing Order | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-22 | User Vault Order Balance Should Increase By Amount Received When Withdrawing Order or claiming a vesting request | ✅ | ❌ | ✅ | 1,000,000+ |
| OC-23 | User Staked esOrder Balance Should Decrease By Amount when calling esOrderUnstakeAndVest | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-24 | Vesting Request requestId Should Equal currentRequestIdBefore when calling esOrderUnstakeAndVest | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-25 | Vesting Request esOrderAmount should equal amount requested when calling esOrderUnstakeAndVest | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-26 | Vesting Request unlockTimestamp should equal block.timestamp + vestingLockPeriod when calling esOrderUnstakeAndVest | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-27 | User currentRequestId Should Equal currentRequestIdBefore + 1 when calling esOrderUnstakeAndVest | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-28 | User staked esOrder balance should increase by request esOrderAmount when cancelling a vesting request | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-29 | Vesting request length for user should decrease by 1 when cancelling a vesting request | ✅ | ✅ | ✅ | 1,000,000+ |

# Invariants Assessed

| ID | Description | Tested | Passed | Remediation | Run Count |
|---|---|---|---|---|---|
| OC-30 | RequestId should have been deleted when cancelling or claiming a vesting request | ✅ | ❌ | ✅ | 1,000,000+ |
| OC-31 | Vesting request length for user should equal 0 when cancelling all vesting requests | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-32 | Order Collector Balance Should Increase By Unclaimed Order When claiming a partial vesting request | ✅ | ❌ | ✅ | 1,000,000+ |
| OC-33 | User Collected Valor should Change by Pending Valor - Amount When redeeming Valor | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-34 | Collected Valor should decrease by amount When redeeming Valor | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-35 | Batch redeemedValorAmount Should Increase By Amount When redeeming Valor | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-36 | Batch userChainedValorAmount Should Increase By Amount When redeeming Valor | ✅ | ✅ | ✅ | 1,000,000+ |
| OC-37 | ORDER token amount received on the Vault chain should be 0 when claiming Usdc Revenue | ✅ | ❌ | ✅ | 1,000,000+ |
| OC-38 | User ChainedUsdcRevenue Should Be Reset when claiming Usdc Revenue | ✅ | ❌ | ✅ | 1,000,000+ |
| OC-39 | User should have received USDC on the Vault chain When claiming Usdc Revenue | ✅ | ❌ | ✅ | 1,000,000+ |

# Findings & Resolutions

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| C-01 | Removed Request Is Always The Last | Logical Error | Critical | Resolved |
| C-02 | Tokens Stuck In Proxyledger When Claiming Vesting | Validation | Critical | Resolved |
| C-03 | Redeeming USDC Revenue Functionality Broken | DoS | Critical | Resolved |
| C-04 | Inability To Transmit Messages To Vault Chain | Logical Error | Critical | Resolved |
| C-05 | Stepwise Jump In User Pending Valor | Logical Error | Critical | Resolved |
| H-01 | Valor Emitted Without Cap | Logical Error | High | Resolved |
| H-02 | Claim Rewards Callback Message Not Executable | Logical Error | High | Resolved |
| H-03 | Partial Vesting Claim Requests DoS'ed | DoS | High | Resolved |
| H-04 | Dust Amount DoS | DoS | High | Resolved |
| H-05 | Missing whenNotPaused Modifier Causes Loss Of Funds | Logical Error | High | Resolved |
| M-01 | No Storage Gaps In Upgradable Contracts | Upgradability | Medium | Resolved |
| M-02 | Missing Validations For Grants | Validation | Medium | Resolved |
| M-03 | Signatures in Valor can be reused | Logical Error | Medium | Resolved |

# Findings & Resolutions

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| M-04 | Vesting Claims DoS'ed With OFT Token Update | DoS | Medium | Resolved |
| M-05 | Staking esOrder Without Balance | Logical Error | Medium | Resolved |
| M-06 | Users May Receive Rewards For Only 1 Batch | Logical Error | Medium | Resolved |
| M-07 | Incorrect Calculation Of USDC Required Per Batch | Logical Error | Medium | Resolved |
| M-08 | Incorrect Access Control For setTotalUsdcInTreasure | Access Control | Medium | Resolved |
| M-09 | Valor May Be Redeemed At An Undesirable Rate | Access Control | Medium | Acknowledged |
| M-10 | Claiming Rewards Possible When Distributor Paused | Logical Error | Medium | Resolved |
| M-11 | USDC Claims Will Fail When Using OFT Wrappers | Logical Error | Medium | Acknowledged |
| M-12 | getRemainingBalance Should Round Up | Logical Error | Medium | Acknowledged |
| M-13 | Recall Can Be Frontrunned | Gaming | Medium | Acknowledged |
| M-14 | Lack Of Withdraw Functions | Logical Error | Medium | Acknowledged |
| L-01 | Implementation Contracts Can Be Initialized By Anyone | Upgradability | Low | Resolved |
| L-02 | fixBatchValorToUsdcRate May Use Stale Rate | Logical Error | Low | Resolved |

# Findings & Resolutions

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| L-03 | Incorrect Error Message During WithdrawOrder | Typo | Low | Resolved |
| L-04 | Use safeTransfer Instead Of transfer | Optimization | Low | Resolved |
| L-05 | Misleading Comments | Optimization | Low | Resolved |
| L-06 | Debug Code In Production | Optimization | Low | Acknowledged |
| L-07 | Unused Errors | Optimization | Low | Resolved |
| L-08 | Typo | Typo | Low | Resolved |
| L-09 | CEI is not followed in LockedTokenVault.claim() | Reentrancy | Low | Resolved |
| L-10 | Owner Can Revoke Ownership | Logical Error | Low | Acknowledged |
| L-11 | Owner Can Steal Undistributed Funds | Logical Error | Low | Acknowledged |
| L-12 | USDC Revenue Claims Are Lost | Logical Error | Low | Acknowledged |
| L-13 | Redundant Parameter In claimRewards() | Logical Error | Low | Resolved |
| L-14 | Contracts Without Receive Can't Use ProxyLedger | DoS | Low | Acknowledged |
| L-15 | Users Can't Fully Claim Vestings | Logical Error | Low | Resolved |

# Findings & Resolutions

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| L-16 | Block Reorgs May Lead To Unexpected Errors | Reorganization | Low | Acknowledged |
| L-17 | Message Fees Lost | Logical Error | Low | Acknowledged |

# C-01 | Removed Request Is Always The Last

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | ● Critical | Vesting.sol: 145 | Resolved |

## Description PoC

Vesting._cancelVestingRequest() and Vesting._claimVestingRequest try to remove the current request by:
1. Overriding it in the storage with the last request in the array.
2. Deleting the last request.

It fails to execute the first step because it only updates the local variable to point to the last request, but no real storage update is done.

As result, the removed request is always the last one. A malicious user can use that to create a cancel request with large value followed by many 1-wei cancel requests. They can then cancel their large request many times because on each cancel, one of the 1-wei requests will be removed instead of the real one. This will cause the user's staked balance to grow indefinitely and result in stolen funds.

## Recommendation

Do not assign values from storage to storage just before deleting them. Cache the value to move to memory, assign it to its new index, and then pop the last element from the storage.

## Resolution

Orderly Team: The issue was resolved in commit 366de22.

# C-02 | Tokens Stuck In Proxyledger When Claiming Vesting

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Validation | ● Critical | OmnichainLedgerV1.sol: 226-236 | Resolved |

## Description PoC

When claiming vested ORDER, the OmnichainLedgerV1 sends `PayloadType.ClaimVestingRequestBackward` message to the ProxyLedger. However, the payload validation in ProxyLedger does not implement the `ClaimVestingRequestBackward` payload and instead reverts.

This will result in tokens being stuck forever in the ProxyLedger because they will be minted to the ProxyLedger when the endpoint's `lzReceive` gets executed and a compose message will be stored that will call the ProxyLedger. The said compose message will always revert and the user will lose their tokens.

## Recommendation

Change the `ProxyLedger` to support the `ClaimVestingRequestBackward` payload.

## Resolution

Orderly Team: The issue was resolved in commit c643f97.

# C-03 | Redeeming USDC Revenue Functionality Broken

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| DoS | ● Critical | Revenue.sol: 187 | Resolved |

## Description

In order to claim USDC revenue from ledger, the owner should mark the batch as claimed using `batchPreparedToClaim`. This will also reduce the `totalValorAmount` as much as the redeemed amount in the batch.

The issue is that `totalValorAmount` is always zero, causing `fixedValorToUsdcRateScaled` to be zero as well. Consequently, attempts to call `batchPreparedToClaim` fail with the `BatchValorToUsdcRateIsNotFixed` error, making it impossible to mark the batch as claimed and preventing users from executing USDC claims.

## Recommendation

Increase `totalValorAmount` by `pendingValor` when `_updateValorVarsAndCollectUserValor` is called.

## Resolution

Orderly Team: The issue was resolved in commit 17e40bf.

# C-04 | Inability To Transmit Messages To Vault Chain

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | ● Critical | LedgerOCCManager.sol: 106 | Resolved |

## Description PoC

LedgerOCCManager relays message requests from the OmnichainLedgerV1 to the vault chain using ledgerSendToVault. This function will call the orderTokenOft contract with the native fee and set LedgerOCCManager as the refund recipient.

These relay requests will fail due to the fact that:
1. The ProxyLedger does not specify any msg.value sent to the lzCompose function during execution.
bytes memory options = OptionsBuilder.newOptions.addExecutorLzReceiveOption(_oftGas, 0).addExecutorLzComposeOption(0, _dstGas, 0);

2. Owner can't fund LedgerOCCManager with native currency, as there is no receive function.

3. If there are any excess fees, the refund transaction will revert.
Therefore, LedgerOCCManager has no native currency to pay the message fees, preventing users to execute any function on ledger chain with a backward message.

## Recommendation

If the message fee is meant to be paid by Orderly, add a receive function to LedgerOCCManager contract to allow deposits and refunds to be processed and a privileged withdraw function. Otherwise, consider specifying the addExecutorLzComposeOption with the msg.value that needs to be sent by the executor in order to pay for the backward message.

## Resolution

Orderly Team: The issue was resolved in commit a2f7330.

Guardian Team: payloadType2BackwardFee[message.payloadType] was added but can lead to DoS when non-zero. Change the messaging fee to MessagingFee memory msgFee = MessagingFee(lzFee, 0);

# C-05 | Stepwise Jump In User Pending Valor

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | ● Critical | Valor.sol: 78 | Resolved |

## Description

User's pending valor is calculated based on the current accrued valor share, user's staked balance and claimed valor. The accrued valor share uses `valorPerSecond` and elapsed time to calculate these shares.

The issue relies on the admin being able to update this `valorPerSecond` state variable, using the permissioned `setValorPerSecond`, without realizing the accumulated valor with the old rate. Any rate hike or decrease will directly affect user's claimable valor (positively or negatively).

## Recommendation

Consider moving the `setValorPerSecond` function to the `Staking.sol` contract and call `updateValorVars` before the rate change.

## Resolution

Orderly Team: The issue was resolved in commit 2d44e9e.

# H-01 | Valor Emitted Without Cap

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | High | Staking.sol: 244 | Resolved |

## Description

The system uses two variables to calculate the valor emission, `totalValorEmitted` and `maximumValorEmission`, and performs a check to validate if the new emission and the total emitted will not surpass the max value.

The issue is that `totalValorEmitted` state variable is never updated, so it's value is always 0. Therefore, the valor cap validations are non existent, and the system can emit valor indefinitely.

## Recommendation

Update the `totalValorEmitted` state variable, by adding the `valorEmission` in `_getCurrentAccValorPreShareScaled`: `totalValorEmitted += valorEmission`

## Resolution

Orderly Team: The issue was resolved in commit 17e40bf.

Guardian Team: `_doValorEmission` is a public function, and calling it directly without calling `updateValorVars` will result in broken internal accounting and inaccurate valor-to-share ratios since it will emit valor without updating `accValorPerShareScaled`. Change the visibility of the `_doValorEmission` function to internal.

Furthermore, with the introduced changes valor will be emitted when paused or when there are no stakers.

# H-02 | Claim Rewards Callback Message Not Executable

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | High | OmnichainLedgerV1.sol: 164 | Resolved |

## Description PoC

Users can claim `ORDER` and `esORDER` rewards based on the merkle distributions.
The issue arises when claiming `esORDER`, as the amount claimed will be staked, but will also try to send a `ClaimRewardBackward` payload to the Vault chain.

The internal `vaultRecvFromLedger` in `ProxyLedger` contract will fail as this callback performs the following check: `require(message.token == LedgerToken.ORDER && message.tokenAmount > 0, "InvalidClaimRewardBackward");`

Therefore, every esORDER claim request will block the LayerZero pathway for the destination chain, as no new messages can be executed before clearing the failed one.

## Recommendation

Avoid sending the message back to Vault Chain when claiming `esORDER` rewards, by early returning after `_stake` or creating an else statement.

## Resolution

Orderly Team: The issue was resolved in commit ec1350c.

# H-03 | Partial Vesting Claim Requests DoS'ed

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| DoS | High | OmnichainLedgerV1.sol: 238 | Resolved |

## Description

Partial vesting claim requests occur when users claim before the vesting linear periods. When these claims are executed ORDER tokens are sent to the user in vault chain, but unclaimed amounts are transferred directly to the `orderCollector` in the ledger chain.

The issue is that `OmnichainLedgerV1` will never have ORDER token balance, so the `safeTransfer` will always fail as long as there is unclaimed amount. Therefore, users will need to wait for the full 90 days vesting period to be over as no partial claims are possible.

## Recommendation

Move the `safeTransfer` call to the `LedgerOCCManager` and create a function in `OmnichainLedgerV1` so that unclaimed amounts can be transferred to the collector.

## Resolution

Orderly Team: The issue was resolved in commit c643f97.

# H-04 | Dust Amount DoS

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| DoS | High | Global | Resolved |

## Description PoC

When the `OFT` token receives a request to send `amountLD` of tokens, it will deduct some dust amount from that. After that a slippage check is performed and if `amountLD - dustAmount` is less than a given `minAmountLD`, the transaction will revert.

The problem found in `OCCManager` and `LedgerOCCManager` is that both `amountLD` and `dustAmount` are set to the same value. This will result in inability to transfer tokens when there is a dust amount present, meaning no staking, withdrawing, claiming can be performed with these amounts.

Ignoring the UX issues, this can be quite problematic for unstaking and vesting `esOrder`. Imagine that a user has some `esAmount` that they unstake and vest. After the vesting period ends, the user is not able to claim their order tokens because `esAmount` has dust amount to be removed.

## Recommendation

A possible solution may be to pass the cleared `amountLD` as `minAmountLD`. However, this can lead to some minor token losses when sending a message from the Ledger side to the Vault side. If you want to mitigate these, you can add an additional state variable that tracks dust amounts and adds them to the respective users' balances.

## Resolution

Orderly Team: The issue was resolved in commit bdc99c5.

# H-05 | Missing whenNotPaused Modifier Causes Loss Of Funds

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | High | Global | Resolved |

## Description PoC

Several contracts in the repository inherit Pausable from OpenZeppelin and correctly implement the `pause` and `unpause` functions.
However, the modifier `whenNotPaused` is not implemented in core contracts such as OrderOFT, OrderAdapter and ProxyLedger.

The modifier should be added to user-facing functions to allow the owner to pause operations in the event of an emergency for example. A more critical issue also exists because OmnichainLedgerV1 correctly implements the `whenNotPaused` modifier while ProxyLedger does not.

If the owner were to pause both contracts, users would still be able to call functions like `stake` on the ProxyLedger which would execute successfully but revert on destination chain.
User's tokens would lose funds as their tokens would be burned on source chain, minted on destination chain but never staked on the user's behalf.

## Recommendation

Add modifier `whenNotPaused` to core functions such as `send`, `stake`, and `sendUserRequest`.

## Resolution

Orderly Team: The issue was resolved in commit bb802a9.

# M-01 | No Storage Gaps In Upgradable Contracts

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Upgradability | Medium | Global | Resolved |

## Description

The system uses upgradable contracts. The following parent contracts don't use storage gaps, which will result in a corrupted storage if a variable is added/removed:

Omnichain:
- VaultOCCManager
- LedgerAccessControl
- OCCAdapterDataLayout and LzTestData
- ChainedEventIdCounter
- MerkleDistributor
- Valor
- Staking
- Revenue
- Vesting

## Recommendation

Add storage gaps to the upgradable contracts.

## Resolution

Orderly Team: The issue was resolved in commit 8dae135.

Guardian Team: Storage gaps and used storage slots should add up to 50 which is not the case for most of the current contracts.

# M-02 | Missing Validations For Grants

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Validation | Medium | LockedTokenVault.sol: 66 | Resolved |

## Description

`grant` only validates if the array params match in length, but there are some crucial validations that should be enforced to correctly grant tokens to a holder:
• Validate that the start and cliff times are in the future.
• Ensure that the cliff time is not before the start time.
• Ensure that the amounts being granted are greater than zero.
• `duration > 0`
• `cliffTime < startTime + duration`

Additionally, when the owner creates a `regrant`, it will add the new amount to the exiting grant, but it will overwrite the timestamps and durations. Setting a shorter duration can allow the user to claim all tokens immediately.

## Recommendation

Consider adding the validations above to correctly create a token grant.

## Resolution

Orderly Team: Resolved.

# M-03 | Signatures in Valor can be reused

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | Medium | Valor.sol: 98 | Resolved |

## Description

The `TREASURE_UPDATER_ROLE` can call `Valor.dailyUsdcNetFeeRevenue` to report for USDC revenue by using a signed message.

However, there are no checks if the signature has already been used which allows for one signature to be reused unlimited times.

## Recommendation

Add a nonce to the signature, hash it and check if it's been already used.

## Resolution

Orderly Team: The issue was resolved in commit be65ca2.

Guardian Team: `data.timestamp` was added to the signature in an attempt to fix M-03. However, signatures can still be reused as the same timestamp can just be passed. Validate whether the signature was used prior.

# M-04 | Vesting Claims DoS'ed With OFT Token Update

| Category | Severity | Location | Status |
|---|---|---|---|
| DoS | Medium | OmnichainLedgerV1.sol: 74 | Resolved |

## Description

`orderTokenOft` is set in the `OmnichainLedgerV1` initialize function, but admin can use `setOrderTokenOft` to update this token address at a later stage.
There are a couple of issue that arise when performing this update:

• `LedgerOCCManager` does not contain an admin function to update the `orderTokenOft` address. If address is updated on `OmnichainLedgerV1` and not in `LedgerOCCManager`, users will still be able to stake with the old token address.

• `LedgerOCCManager` will hold `orderTokenOft` tokens sent from vault chains. If the token address is updated, any withdraw or claim will fail as the contract does not have balance of the new token.

• When users request vesting claims, the `unclaimedOrderAmount` will be sent to the `orderCollector`. In case admin updates the `orderTokenOft`, user claims will be DoS'ed as the contract may not have sufficient amount of the new token to transfer the collector.

## Recommendation

Avoid changing the `orderTokenOft` address. Alternatively, add `setOrderTokenOft` to the `LedgerOCCManager` contract or always read the updated address from the `OmnichainLedgerV1`. Be aware that same amount of OFT tokens should me minted as the previous OFT token balance in `LedgerOCCManager` to avoid issues with the stakes and vests.

## Resolution

Orderly Team: The issue was resolved in commit c643f97.

# M-05 | Staking esOrder Without Balance

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | Medium | ProxyLedger.sol: 129 | Resolved |

## Description

`ProxyLedger` allows users to stake ORDER or esORDER. When the `isEsOrder` flag is set to true, users will effectively stake esORDER on the `OmnichainLedgerV1` contract, but will also be charged ORDER tokens, due to the fact that `vaultSendToLedger` will deduct the staked amount from the user balance.

Therefore, user will end up with an esORDER stake, and ORDER tokens will be sent to the `LedgerOCCManager` contract. Although user will be able to unstake, vest and re-claim the ORDER tokens, this is an unexpected behavior as esORDER staking should not be triggered directly.

Additionally, if future upgrades give more reward weight to esORDER staking, users will be able to game the system by creating an esORDER stake with ORDER tokens.

## Recommendation

Always use `LedgerToken.ORDER` in the ProxyLedger `stake()` and remove the `isEsOrder` param.

## Resolution

Orderly Team: The issue was resolved in commit edebe01.

# M-06 | Users May Receive Rewards For Only 1 Batch

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | Medium | Revenue.sol | Resolved |

## Description

The intended batch flow is as follows:
1. Users redeem valor in batch A.
2. Batch A finishes.
3. A trusted role marks the batch as claimable.

Whenever a user redeems their valor or claim USDC, `_collectUserRevenueForClaimableBatch` is called to increase their withdrawable amount by adding the withdrawable amounts of the first claimable batch.

If there are users that redeem their valor in between steps 2 and 3 from above (so `batch A` is finished and `batch B` is active) when they claim USDC when batch B becomes claimable they will be given the reward from only the first claimable batch even though they are entitled to rewards from two batches.

## Recommendation

Accumulate rewards for all claimable batches when claiming

## Resolution

Orderly Team: The issue was resolved in commit d73c7ed.

# M-07 | Incorrect Calculation Of USDC Required Per Batch

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | Medium | Revenue.sol: 142 | Resolved |

## Description

The `Revenue` contract tracks the amount of USDC tokens required per batch id, for users to redeem. It uses the `fixedValorToUsdcRateScaled` to calculate the USDC amount based on the batch valor amount.

This USDC amount calculation is missing the `VALOR_TO_USDC_RATE_PRECISION` correction, so the value returned is greater than expect, thus invalid. Additionally, if the batch id is not claimable yet, the `fixedValorToUsdcRateScaled` value is not set (value 0), so all amounts will be 0 due to the multiplication.

## Recommendation

Add a division by `VALOR_TO_USDC_RATE_PRECISION` to fix the units.

## Resolution

Orderly Team: The issue was resolved in commit d73c7ed.

# M-08 | Incorrect Access Control For setTotalUsdcInTreasure

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Access Control | Medium | Valor.sol: 110 | Resolved |

## Description

`setTotalUsdcInTreasure` was meant to be called by the `DEFAULT_ADMIN_ROLE` and not the `TREASURE_UPDATER_ROLE`, as the latter will use `dailyUsdcNetFeeRevenue` which validates signatures and checks timestamps.

## Recommendation

Update the access control modifier in `setTotalUsdcInTreasure` to use `onlyRole(DEFAULT_ADMIN_ROLE)`

## Resolution

Orderly Team: The issue was resolved in commit 6828c6a.

# M-09 | Valor May Be Redeemed At An Undesirable Rate

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Access Control | Medium | Revenue.sol: 203-221 | Acknowledged |

## Description

A user may observe that redeeming valor in `batch A` is quite profitable so they initiate a `redeemValor` request. However, due to their transaction taking quite a long time to be executed and delivered to the Orderly Network, their redeem request may now end up in `batch B`. In `batch B` the redeeming terms may not be as good as the user desired.

For example, a lot of valor accumulated without much profit or the admin called `Valor.setTotalUsdcInTreasure` to decrease the usdc. The user will not be able to stop the transaction and in result, they will redeem at undesirable rate.

## Recommendation

Add an additional `batchId` parameter to the `redeemValor` function and revert the tx if it doesn't match the current `batchId`.

## Resolution

Orderly Team: Users will be warned about this behavior.

# M-10 | Claiming Rewards Possible When Distributor Paused

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | Medium | MerkleDistributorL1.sol: 240 | Resolved |

## Description

According to the comments in MerkleDistributor:
Contract is pausable by owner. It allows to pause claiming rewards.

However, `claimRewards` doesn't have the `whenNotPaused` modifier. This is most likely because `claimRewards` calls function `updateRoot` which implements it. However, the call to `updateRoot` is conditional - it happens only if there is an upgrade possible. Otherwise, claiming is still possible - even in paused state.

## Recommendation

Add the `whenNotPaused` modifier to the `claimRewards` function.

## Resolution

Orderly Team: The issue was resolved in commit 3be4fb4.

# M-11 | USDC Claims Will Fail When Using OFT Wrappers

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | Medium | ProxyLedger.sol: 245 | Acknowledged |

## Description

In order to make a batch claimable, admins need to update the `totalUsdcInTreasure` using `dailyUsdcNetFeeRevenue`. If the USDC amount added has 6 decimal precision (as most USDC tokens), then the user will receive a 6 decimal amount when the `ClaimUsdcRevenueBackward` payload is executed.
`bool success = IERC20(usdcAddr).transfer(message.receiver, message.tokenAmount);`

The protocol plans to allow claims in multiple chains, using both native USDC or OFT wrappers. The issue is that OFT tokens have 18 decimals by default. Therefore, users will receive less tokens than expected when using OFT wrappers (If the protocol launches on BSC, the pegged USDC token has 18 decimals).

Additionally, as the ProxyLedger will transfer native USDC tokens, it should use `safeTransfer` instead.

## Recommendation

Create the OFT token wrapper for USDC that will be used in the vault chains, overriding `decimals` to use 6 decimals instead of 18. Ensure that admin updates daily USDC revenue with correct decimals.

## Resolution

Orderly Team: Only Order token will be OFT wrapped.

# M-12 | getRemainingBalance Should Round Up

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | Medium | LockedTokenVault.sol: 175 | Acknowledged |

## Description

Vestings are linearly unlocked after a cliff period during the release duration. Claimable balance of a vesting is calculated by subtracting the remaining balance from the total balance.

However, remaining balance of a vesting is calculated using the `mulDiv` formula from `Math` Library and this formula rounds down by default. Rounding down the remaining balance means rounding up the claimable balance.

Total claimable amounts will not be affected from this but intermediary claims will give users slightly more tokens than it should.

## Recommendation

Roundings should be in favour of the protocol. Use the `mulDiv` function with selective rounding option from the same library.

## Resolution

Orderly Team: Acknowledged.

# M-13 | Recall Can Be Frontrunned

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Gaming | Medium | LockedTokenVault.sol: 95 | Acknowledged |

## Description

Owner has a right to `recall` a grant and this function returns unclaimed tokens to the owner from the holder. Returning unclaimed tokens:

1. Creates unfair situations between holders. Let's assume there are two different holders with exact same cliff and release duration parameters but one of them claimed unlocked part of his vesting and the other didn't claim anything yet. Recalling from both of these holders results in different user balances.

2. Creates a surface for frontrunning attacks. Users can frontrun the `recall` and claim their vestings' `claimable` portion just before the recall.

## Recommendation

Consider adding a new functionality to recall only the remaining portion of a vest alongside `recall`.

## Resolution

Orderly Team: Acknowledged.

# M-14 | Lack Of Withdraw Functions

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | Medium | Global | Acknowledged |

## Description

LedgerOCCManager, OCCManager (and potentially OmnichainLedgerV1) are expected to hold ORDER / USDC tokens which are burned or transferred when users claim.

The issue lies with the lack of a withdraw function for Owner to recover these tokens from the contract. Owner may wish to do this during a migration to a new contract.
Without a withdraw function, these tokens may become permanently stuck in the contract.

## Recommendation

Implement a withdraw function for the contracts which are expected to hold ERC-20 tokens.

## Resolution

Orderly Team: Will consider add withdraw or do it during upgrades.

# L-01 | Implementation Contracts Can Be Initialized By Anyone

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Upgradability | Low | Global | Resolved |

## Description

ProxyLedger, OmnichanLedgerV1 and LedgerOCCManager are all implementation contracts designed to be called through a proxy contract. However, the initialize function can be directly called by anyone.

This would allow a malicious actor to set storage variables such as `lzEndpoint` and `owner` on the implementation contract. While no direct risks were found, it is general good practice to prevent `initialize` from being called.

## Recommendation

Use `disableInitializer` in a constructor. See OpenZeppelin's guide for more details.

## Resolution

Orderly Team: The issue was resolved in commit cdb53b6.

# L-02 | fixBatchValorToUsdcRate May Use Stale Rate

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | Low | Revenue.sol: 171 | Resolved |

## Description

Users earn valor and collect USDC using their valors. The amount of USDC to collect is determined based on USDC to valor rate and this rate is fixed for every batch by the admin.
However, fixing the rate for a batch is done a few days after the batch is finished (It is expected to be 2-3 days after), and the rate at that time is used.

Revenue earned during these days, which should be allocated for the current batch, are allocated for the previous batch. Ideally, the rate at the time when a batch is finished should be used for fairness in between batches. Also, there will be unfair distribution between batches if the fixing time is not exactly the same for every batch.

## Recommendation

Consider using the rate exactly at the time when a batch is finished and also consider using smart contract automation systems like Chainlink Keeper.

## Resolution

Orderly Team: The current sequence should avoid the problem.

# L-03 | Incorrect Error Message During WithdrawOrder

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Typo | Low | ProxyLedger.sol: 236 | Resolved |

## Description

The error message in `vaultRecvFromLedger` for `PayloadDataType.WithdrawOrderBackward` is "`InvalidClaimRewardBackward`". The contract also has error messages alluding the `OrderlyBox`, but this is not the correct contract and won't help debugging transactions.

## Recommendation

Correct the error message to "`InvalidWithdrawOrderBackward`" and update the revert string contract.

## Resolution

Orderly Team: The issue was resolved in commit c643f97.

# L-04 | Use safeTransfer Instead Of transfer

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Optimization | Low | Global | Resolved |

## Description

`MerkleDistributorL1` and `OmnichainLedgerV1` contracts use `safeTransfer` for ERC20 transfers but `ProxyLedger` and `OCCManager` contracts use regular `transfer`.

## Recommendation

Consider using `safeTransfer` and `safeTransferFrom` in mentioned contracts too.

## Resolution

Orderly Team: The issue was resolved in commit 4cabfc4.

# L-05 | Misleading Comments

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Optimization | Low | ProxyLedger.sol: 156, 157 | Resolved |

## Description

Comments above the `buildOCCMessage` function in `ProxyLedger` contract explain `payloadType` params. However, some of these comments are not correct and they are misleading.
In the comment, `RedeemValor` is incorrectly matched with enum 6 while it should have been 9. `ClaimUsdcRevenue` is also not enum 7 but it is 10. Furthermore, remaining functions are not mentioned in the comments.

## Recommendation

Consider updating the Natspec comment.

## Resolution

Orderly Team: The issue was resolved in commit 6b49009.

# L-06 | Debug Code In Production

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Optimization | Low | Global | Acknowledged |

## Description

`_msgPayload` and `_options` variables in `LedgerOCCManager:L114-115` and `OCCManager:104-105` are not used in production but used only for testing purposes.

Also, the commented line that has been used for testing in `LedgerOCCManager:179` left in the production.

## Recommendation

Consider removing these lines from production code.

## Resolution

Orderly Team: Acknowledged.

# L-07 | Unused Errors

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Optimization | Low | Global | Resolved |

## Description

`ValorPerSecondExceedsMaxValue` custom error in `Valor` contract, `VestingPeriodIsOutOfRange` and `DepositNotEnough` custom errors in `Vesting` contract are defined but never used.

## Recommendation

Remove unused errors.

## Resolution

Orderly Team: The issue was resolved in commit 45179fb.

# L-08 | Typo

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Typo | Low | MerkleDistributorL1.sol: 103 | Resolved |

## Description

There is a typo in L103 of the `MerkleDistributorL1` contract. “But it there…” should be “But if there…”.

## Recommendation

Update the comment in the mentioned line.

## Resolution

Orderly Team: The issue was resolved in commit 45179fb.

# L-09 | CEI is not followed in LockedTokenVault.claim()

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Reentrancy | Low | LockedTokenVault.sol: 112-115 | Resolved |

## Description

The claim function in the Vesting contract doesn't follow the CEI pattern - it first transfers tokens to the claimer and only after that updates their balance in the mapping.

If in the future this contract is used for other tokens with hooks functionality, an attacker can hijack the execution flow when claiming and drain the funds in the contract.

## Recommendation

Follow the CEI pattern by first updating the `claimedBalances` and then transferring the tokens.

## Resolution

Orderly Team: Resolved.

# L-10 | Owner Can Revoke Ownership

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | Low | Global | Acknowledged |

## Description

All `Ownable` contracts allow the owner to renounce their ownership. This can leave the contracts in an unexpected state and hinder the functioning of the protocol.

## Recommendation

Consider utilizing `Ownable2Step`.

## Resolution

Orderly Team: This operation will be carefully checked.

# L-11 | Owner Can Steal Undistributed Funds

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | Low | MerkleDistributorL1.sol: 285-288 | Acknowledged |

## Description

MerkleDistributorL1.withdraw lets the admin withdraw the unclaimed funds after the distribution ends. However, an admin can at any time update the merkle root with endTimestamp = startTimestamp + 1 and immediately withdraw the funds.

## Recommendation

Consider adding a required gap between startTimestamp and endTimestamp in proposeRoot.

## Resolution

Orderly Team: Acknowledged.

# L-12 | USDC Revenue Claims Are Lost

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | Low | ProxyLedger.sol: 245 | Acknowledged |

## Description

Users will start a `claimUsdcRevenue` transaction from the `ProxyLedger` but the contract might not have enough USDC balance. Therefore, users will effectively trigger claim on ledger chain, but transaction can't be completed on vault chain.

## Recommendation

Consider validating the claimed amount against the contract USDC balance before triggering the claim transaction.

## Resolution

Orderly Team: Orderly is responsible to deposit enough usdc first.

# L-13 | Redundant Parameter In claimRewards()

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | Low | ProxyLedger.sol: 80 | Resolved |

## Description

ProxyLedger.claimReward accepts isEsOrder parameter and sets the token of the payload to either ORDER or esORDER. This is redundant because users claim different distributions which are already set for a given token.

## Recommendation

Consider removing the `isEsOrder` parameter and use the `LedgerToken.PLACEHOLDER` in the payload instead.

## Resolution

Orderly Team: The issue was resolved in commit 5b102ad.

# L-14 | Contracts Without Receive Can’t Use ProxyLedger

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| DoS | Low | OCCManager.sol: 107 | Acknowledged |

## Description

The OCCManager sets the msg.sender as the recipient for the LayerZero refunds when sending a tx from Vault -> Ledger. This means contracts with no receive function will not be able to send messages whenever there is a refund happening.

## Recommendation

Either document that smart contracts interacting with the system must be able to receive funds, or let the user pass a `refund` parameter and use it instead of `msg.sender`.

## Resolution

Orderly Team: Will inform our users.

# L-15 | Users Can’t Fully Claim Vestings

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | Low | Vesting.sol: 219 | Resolved |

## Description

After the lock period and linear periods ends, users should be able to claim 100% of `esOrderAmount`. Although, if the amount is an odd number, the total claimed will be 1 wei less, due to solidity rounding:
`return _vestingRequest.esOrderAmount / 2 + (_vestingRequest.esOrderAmount * vestedTime) / vestingLinearPeriod / 2;`

## Recommendation

Consider returning `_vestingRequest.esOrderAmount` when `vestedTime > vestingLinearPeriod`.

## Resolution

Orderly Team: The issue was resolved in commit 7141809.

# L-16 | Block Reorgs May Lead To Unexpected Errors

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Reorganization | Low | Global | Acknowledged |

## Description

The protocol intends to use multiple chains as `vault` chains. One of this chains is `Polygon` known for having many reorgs in the blockchain. These reorgs occur when an alternative version of the blockchain gains consensus, effectively rewriting a part of the blockchain's transaction history.

The following scenario may occur in a block reorg:
• claim rewards backward message is sent to Polygon
• mints ORDER tokens to ProxyLedger in one tx
• `lzCompose` tx sends ORDER tokens to the user. - A reorg may occur, discarding the ORDER mint tx, but including the ORDER transfer tx. Users will receive double amount ORDER tokens when the `lzCompose` is re-submitted

## Recommendation

Consider documenting the issue and monitoring as impact may be limited with LayerZero default configurations.

## Resolution

Orderly Team: Will set larger block confirmations for polygon.

# L-17 | Message Fees Lost

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Error | Low | MerkleDistributor.sol: 307 | Acknowledged |

## Description

A user may send a claimReward request and pays for all fees, but the merkle root undergoes update before their transaction is executed. Therefore, user will pay message fees but won't be able to claim the rewards. They will need to send a second transaction with the new merkle root in order to claim the rewards.

## Recommendation

Consider documenting the issue so the users are aware of this.

## Resolution

Orderly Team: Will be added to documentation and FAQ.

# Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

# About Guardian Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits