GA GUARDIAN

Orderly Vault

Cross-chain Vault

Security Assessment

February 24th, 2025

Summary

Audit Firm Guardian

Prepared By Roberto Reigada, Kiki, Osman Ozdemir,

Zdravko Hristov, Mark Jonathas, Michael Lett

Client Firm Orderly

Final Report Date February 24th, 2025

Audit Summary

Orderly engaged Guardian to review the security of their cross-chain, share-based yield aggregator smart contracts. From the 2nd of January to the 20th of January, a team of 6 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Issues Detected Throughout the engagement 13 High/Critical issues were uncovered and promptly remediated by the Orderly team.

Security Recommendation Given the number of High and Critical issues detected as well as additional code changes made after the main review, Guardian recommends that an independent security review of the protocol at a finalized frozen commit is conducted before deployment.

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/orderly-strategy-vault-fuzzing
https://github.com/GuardianAudits/orderly-contract-evm-fuzzing
https://github.com/GuardianAudits/orderly-evm-cross-chain-fuzzing

Table of Contents

Project Overview

Project Summary

Project Name	Orderly Vault
Language	Solidity
Codebase	https://gitlab.com/orderlynetwork/orderly-v2/strategy-vault https://gitlab.com/orderlynetwork/orderly-v2/contract-evm/-/tree/dev?ref_type=heads https://gitlab.com/orderlynetwork/orderly-v2/evm-cross-chain/-/tree/dev/contracts
Commit(s)	Initial commit: - strategy-vault@9885d366f14058466e63819574097d0a15c8d3b3 - contract-evm@b4bcf2af2055c3dac9009ee587bf5d35a06be16a - evm-cross-chain@0420c122390cbef76f33187b004cc86966e05377 Final commit: - strategy-vault@76da69a3d1c9f4faca868fa77e2a2ca0e726d7c4 - contract-evm@710f2d3b8fb23e6482ebd39d3c7be896b160f627 - evm-cross-chain@7d71522351a74a0290fc7ac8ae49a4a7ce9554ec

Audit Summary

Delivery Date	February 24, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
🔴 Critical	7	0	0	0	0	7
🟠 High	6	0	0	2	0	4
🟡 Medium	16	0	0	7	1	8
🟢 Low	35	0	0	22	0	13

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 Orderly Vault, fuzz-testing with Foundry was performed on the protocol's main functionalities. 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 10,000,000+ runs with a prepared Foundry fuzzing suite.

ID	Description	Tested	Passed	Remediation	Run Count
PV-01	Deposit to ProtocolVault should deduct user tokens	✅	✅	✅	10M+
PV-02	Receiver account token info unallocatedAssets should increase by amount when deposit is called on the ProtocolVault	✅	✅	✅	10M+
PV-03	Receiver account token info assets should increase by amount when deposit is called on the ProtocolVault	✅	✅	N/A	10M+
PV-04	Deposit to ProtocolVault should deduct strategy tokens	✅	✅	✅	10M+
PV-05	Strategy token info unallocatedAssets should increase by amount when deposit is called on the ProtocolVault	✅	✅	✅	10M+
PV-06	Receiver account token info frozenShares should increase by amount when withdraw is called on the ProtocolVault	✅	✅	✅	10M+
PV-07	NotEnoughWithdrawShares check should not be bypassed	✅	❌	✅	10M+
PV-08	Strategy token info frozenShares should increase by amount when withdraw is called on the ProtocolVault	✅	✅	✅	10M+
PV-09	User asset balance should increase by unclaimed assets when claiming assets	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
PV-10	Strategy asset balance should increase by unclaimed assets when claiming assets	✅	✅	✅	10M+
PV-11	If asset per share > strategy hwm fundAssetsAfterFee must be less than total pending fund assets when calling updateStrategyFundAssets	✅	✅	✅	10M+
PV-12	If asset per share > strategy hwm pendingStrategyProviderShares must increase when calling updateStrategyFundAssets	✅	✅	✅	10M+
PV-13	If asset per share > strategy hwm pendingTotalShares must be greater than total pending fund assets when calling updateStrategyFundAssets	✅	✅	✅	10M+
PV-14	AccountId PendingShares must be converted to accountId Shares after settleAccounts	✅	✅	✅	10M+
PV-15	latestPeriodId should increment by 1 after updatePeriodId	✅	✅	✅	10M+
PV-16	pendingLpDepositAssets should increment be set to 0 after updatePeriodId	✅	✅	✅	10M+
PV-17	pendingLpWithdrawShares should increment be set to 0 after updatePeriodId	✅	✅	N/A	10M+
PV-17-REM	pendingLpWithdrawAssets should be set to 0 after updatePeriodId	✅	N/A	✅	10M+
PV-18	Protocol Vault token balance should decrease by asset distribution	✅	❌	✅	10M+
PV-19	Dex Vault token balance should increase by asset distribution	✅	❌	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
PV-20	Dex Vault Ledger token balance should increase by asset distribution	✅	❌	✅	10M+
PV-21	User unclaimable assets in ProtocolVault should increase by user claim info assets in ProtocolVaultLedger	✅	✅	✅	10M+
PV-22	On executeWithdrawAction accountId ledger balance should decrease by amount	✅	✅	✅	10M+
PV-23	On executeWithdrawAction receiver token balance should increase	✅	✅	✅	10M+
PV-24	On executeWithdrawAction protocolVault accountId ledger balance should decrease by amount	✅	✅	✅	10M+
PV-25	On executeWithdrawAction protocolVault token balance should increase	✅	✅	✅	10M+
PV-26	feeCollector account balance should increment by fee amount after withdraw2Contract	✅	❌	✅	10M+
PV-27	Protocol Vault balance should increase by amount minus fee after withdraw2Contract	✅	✅	✅	10M+
PV-28	When depositing to the DexVault user token balance should decrease by amount	✅	✅	✅	10M+
PV-29	User account balance for the Ledger should increase by tokenAmount when depositing to the DexVault	✅	✅	✅	10M+
PV-30	globalEventId should increment by 1 after a deposit from the DexVault	✅	✅	✅	10M+
PV-31	globalDepositId should increment by 1 after a deposit from the DexVault	✅	✅	✅	10M+

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	Missing Access Control In sendMessage Function	Configuration	Critical	Resolved
C-02	Some ProtocolVault Functions Do Not Validate Properly The PayloadType	Validation	Critical	Resolved
C-03	handleOpFromVault Function Does Not Scale operationData.amount Decimals	Validation	Critical	Resolved
C-04	State Variables Are Updated After A ProtocolVaultLedger._checkWithdraw Failed Check	Logical Error	Critical	Resolved
C-05	Initial DoS State In ProtocolVaultLedger Contract Multiple Divisions By Zero	Logical Error	Critical	Resolved
C-06	rebalanceMint May Corrupt Ledger State	Logical Error	Critical	Resolved
C-07	Compilation Error Due To Naming Mismatch	Code Best Practices	Critical	Resolved
H-01	withdraw2Contract Crosschain Flow Pays Withdrawal Fee Twice	Configuration	High	Resolved
H-02	depositToStrategy Function Will Always Revert	Logical Error	High	Resolved
H-03	Newly Added Strategy Provider Will Not Receive LP Deposits	Logical Error	High	Acknowledged
H-04	Wrong Cross-chain Manager Usage	Configuration	High	Resolved
H-05	DOS Of updateLPAndStrategyFund	Logical Error	High	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
H-06	Withdraw Message Sent Even If _validReceiver Check Fails	Validation	High	Acknowledged
M-01	SP Deposits Are Not Refunded If The Strategy Is Not Allowed In The ProtocolVaultLedger Contract	Configuration	Medium	Resolved
M-02	Missing Payable Modifier In ProtocolVault.depositToStrategy Function	Configuration	Medium	Resolved
M-03	Operators Could Replay Signatures In The ProtocolVaultLedger Contract	Configuration	Medium	Resolved
M-04	No Enforcement Of Required Bridging Fee In sendMessage	Validation	Medium	Resolved
M-05	Strategy Deposit Might Fail Due To Limit	Logical Error	Medium	Acknowledged
M-06	FeeRate Changes Lead To Loss Of Yield	Logical Error	Medium	Resolved
M-07	Native Funds Locked In Contract	Logical Error	Medium	Resolved
M-08	Event Emitted With Incorrect Values	Validation	Medium	Resolved
M-09	A Portion Of Frozen Fees Will Remain Frozen	Logical Error	Medium	Acknowledged
M-10	Period Update Breaks When Batching	Logical Error	Medium	Acknowledged
M-11	Rounding Up Will DoS When Funds Are Withdrawn	Logical Error	Medium	Acknowledged
M-12	Enabling Tokens Breaks Protocol	Configuration	Medium	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
M-13	Token Losses On Deposit	Validation	Medium	Acknowledged
M-14	Vault LZ Fee Can Be Lost	Validation	Medium	Resolved
M-15	Insufficient msgOptions	Configuration	Medium	Acknowledged
M-16	Insufficient Validation In withdraw2Contract	Validation	Medium	Partially Resolved
L-01	Wrong Check In _convertToShares Function	Configuration	Low	Resolved
L-02	quoteOperation Function Always Assumes a LP_DEPOSIT Payload	Configuration	Low	Resolved
L-03	ProtocolVault Claim Function Can Transfer Any Locked Token	Validation	Low	Resolved
L-04	Possible Inconsistent Decimal Precision Configuration	Validation	Low	Resolved
L-05	Redundant Period ID Parameter	Code Best Practices	Low	Acknowledged
L-06	Proportional Allocation Overfunds Past Performers	Configuration	Low	Acknowledged
L-07	Minor Unallocated Remainders In Proportional LP Allocation	Precision Loss	Low	Acknowledged
L-08	Debugging Checks And Test Code Left In Production Code	Code Best Practices	Low	Acknowledged
L-09	Lack Of A Double Step TransferOwnership Pattern	Code Best Practices	Low	Resolved
L-10	CCTP depositForBurn Has Maximum Burn Per Transaction	Validation	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-11	Floating Pragma	Code Best Practices	Low	Acknowledged
L-12	Incompatibility With CREATE2 On ZkSync	Configuration	Low	Acknowledged
L-13	Griefing Of Vault Deposits	Validation	Low	Acknowledged
L-14	Unfair Performance Fee Distribution	Logical Error	Low	Acknowledged
L-15	No Paused Check For Withdrawals	Configuration	Low	Acknowledged
L-16	Centralization Risks	Configuration	Low	Acknowledged
L-17	Multiple Typos	Code Best Practices	Low	Resolved
L-18	updateUnclaimed Does Not Have Duplicate Check	Validation	Low	Acknowledged
L-19	Salt Is Not Hashed With Deployer Address	Configuration	Low	Acknowledged
L-20	Unused Errors And Events	Code Best Practices	Low	Resolved
L-21	Warning About Paused DexVault	Configuration	Low	Acknowledged
L-22	Unvalidated Params In setAllowedStrategyProvider	Validation	Low	Resolved
L-23	Some Functions Cant Cover Fee Costs	Configuration	Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-24	A Hard-fork Can Disrupt Messaging	Configuration	Low	Acknowledged
L-25	accountToken.assets Is Never Decreased	Logical Error	Low	Resolved
L-26	No Withdrawal Confirmation For Solana	Configuration	Low	Acknowledged
L-27	Market Manager Flag Not Cleared	Validation	Low	Resolved
L-28	Slight withdrawAssets Discrepancy	Precision Loss	Low	Resolved
L-29	safeApprove() Deprecated	Code Best Practices	Low	Acknowledged
L-30	Token Deposits Cannot Be Disabled	Configuration	Low	Acknowledged
L-31	enableDepositFee Can Be Paused	Configuration	Low	Acknowledged
L-32	Mixed Decimals	Configuration	Low	Resolved
L-33	Unable To Reinitialize Contracts	Configuration	Low	Acknowledged
L-34	Lack Of onlyProxy Modifier	Code Best Practices	Low	Acknowledged
L-35	Cross-chain Communication May Be Blocked	Configuration	Low	Acknowledged

C-01 | Missing Access Control In sendMessage Function

Category	Severity	Location	Status
Configuration	● Critical	VaultCrossChainManager.sol: 47	Resolved

Description

The `VaultCrossChainManager` contract implements the `sendMessage` which is used to send cross-chain messages.

This function does not implement any type of access control and, therefore, any malicious user can craft arbitrary cross-chain payloads and transmit them causing the receiving chain's contract logic to execute unverified operations.

This could be easily exploited to perform unauthorized withdrawals or deposits.

Recommendation

Introduce strict access control to `sendMessage` so that only trusted contracts, such as whitelisted vaults, can invoke cross-chain operations.

Resolution

Orderly Team: The issue was resolved in commit 8540bc3.

C-02 | Some ProtocolVault Functions Do Not Validate Properly The PayloadType

Category	Severity	Location	Status
Validation	● Critical	ProtocolVault.sol: 130, 157	Resolved

Description

In the `ProtocolVault` contract, both the deposit and withdraw functions accept a `PayloadType` that is never validated against the function's intended usage. As a result, a user could call the withdraw function but submit a payload indicating `LP_DEPOSIT` or `SP_DEPOSIT`.

On the ledger side, this is interpreted as a deposit even though no tokens were transferred to the `ProtocolVault`, artificially increasing the user's balance. This leads to unbacked shares on the ledger and allows users to perform "free" deposits.

Recommendation

Add strict checks in each function to require that deposit only accepts deposit payloads (`LP_DEPOSIT` or `SP_DEPOSIT`) and withdraw only accepts withdrawal payloads (`LP_WITHDRAW` or `SP_WITHDRAW`). Any other `PayloadType` should always revert.

Resolution

Orderly Team: The issue was resolved in commit 3275584.

C-03 | handleOpFromVault Function Does Not Scale Decimals

Category	Severity	Location	Status
Validation	● Critical	ProtocolVaultLedger.sol: 132	Resolved

Description

In the handleOpFromVault function, the ProtocolVaultLedger contract treats operationData.amount as a direct integer without adjusting for the underlying token's decimals.

Assets and shares are supposed to be stored with 6 decimals precision. As per the code comments, this accountToken will be USDC and USDC has 6 decimals in most of the chains. However USDC has 18 decimals instead of 6 on the following chains:
• Oasys
• BNB
• OKX Chain
• Sora
• Kucoin Chain
• Telos
• Conflux
• Bitgert

If accountToken has a different number of decimals than 6 the ProtocolVaultLedger contract calculations end up over-counting or under-counting actual token amounts which would totally break the accounting in the contract.

Recommendation

Normalize operationData.amount according to the token's decimals before updating ProtocolVaultLedger 's state. A robust approach is to store each token's decimal information on-chain and adjust incoming amounts consistently.

Resolution

Orderly Team: The issue was resolved in commit 872f355.

C-04 | State Variables Are Updated After A Failed Check

Category	Severity	Location	Status
Logical Error	● Critical	ProtocolVaultLedger.sol: 704	Resolved

Description

In the `ProtocolVaultLedger` contract, the `_checkWithdraw` function merely emits an event and returns early if a user attempts to withdraw more shares than they hold, instead of reverting the entire transaction. Because control flow proceeds after this early return, the contract continues to update its state variables (e.g., incrementing `frozenShares` or proceeding with other post-withdraw steps) even when the user's withdrawal request is invalid.

Recommendation

In case that the `_checkWithdraw` function require check was not passed, ensure that the `frozenShares` state variable is not updated.

Resolution

Orderly Team: The issue was resolved in commit 3d698ab.

C-05 | Initial DoS State Contract Multiple Divisions By Zero

Category	Severity	Location	Status
Logical Error	● Critical	ProtocolVaultLedger.sol	Resolved

Description

In the `ProtocolVaultLedger` contract, when a strategy fund has not yet minted any shares, `strategyFundTokenInfo[strategyProviderId][USDC_HASH].totalShares` remains zero, leading to a guaranteed division by zero during future certain end-of-period operations.

In `settleMainAndStrategyFunds`, the code invokes `_calculateHWM` to update the High Water Mark, which executes the line `hwm = strategyFundToken.fundAssetsAfterFee * 10 * priceDecimal / totalShares` reverting because `totalShares` is zero.

A similar issue appears in `updateStrategyFundAssets` if `fundShares` (i.e., `strategyFundToken.totalShares`) is zero, again causing a division-by-zero revert.

As a result, the very first call to these functions in the `ProtocolVaultLedger` contract will always fail, blocking any attempt to settle or update strategy fund assets when no shares are yet in circulation.

This breaks the normal lifecycle flow for the initial period, preventing operators from correctly finalizing and advancing the ledger state.

Recommendation

Introduce specialized handling for the zero-shares scenario in `settleMainAndStrategyFunds` and `updateStrategyFundAssets`. Whenever `totalShares = 0`, skip or defer the HWM calculation and other division-based logic until at least one share exists.

Resolution

Orderly Team: The issue was resolved in commit 4d1464d.

C-06 | rebalanceMint May Corrupt Ledger State

Category	Severity	Location	Status
Logical Error	● Critical	Global	Resolved

Description

The ledger can perform `rebalanceBurn` and `rebalanceMint` of tokens. This effectively burns the tokens on one chain and mints them on another one by using Circle's tokenManager.

The flow is as follows:

1. `Ledger.executeRebalanceBurn()`. This will deduct the burnt amount from the chain's balance in `VaultManager` and will add it to the `frozenBalances` in case the burn fails.

2. A cross chain message is sent to `Vault.rebalanceBurn()`.

3. `Vault.rebalanceBurn()` calls `tokenMessengerContract.depositForBurn()`

4. If the call fails, we send a failed rebalanceBurnFinish message to the ledger, to increase the chain's balance back from the frozen tokens.

5. If the call succeeds, the tokens are burnt from the vault and, event is emitted and a successful `rebalanceBurnFinish` message is sent to reduce the frozen tokens.

6. Once enough attestations confirm the message, it can be executed on the destination chain by calling `messageTransmitterContract.receiveMessage()`.

7. If the receive is successful, the tokens are minted and a successful `rebalanceMintFinish` is sent to the Ledger to increase the balance of the destination chain.

8. Otherwise, if the receive fails, a failed `rebalanceMintFinish` will be sent to the Ledger.

The problem with this flow is that `messageTransmitterContract.receiveMessage()` is permissionless. If anyone calls it before the Vault, the message's nonce will be consumed and even though the mint is successful, the Vault will treat it as failed.

In result, the tokens will be deducted from the source chain, but won't be credited to the destination chain, leading to loss of funds.

Recommendation

If the nonce is already used, `messageTransmitterContract.receiveMessage()` will revert with `Nonce already used`. You can catch that and send a successful `rebalanceMintFinish` to update the state correctly.

Resolution

Orderly Team: The issue was resolved in the merge request 332.

C-07 | Compilation Error Due To Naming Mismatch

Category	Severity	Location	Status
Code Best Practices	● Critical	VaultCrossChainManagerUpgradeable.sol: 153	Resolved

Description

`EventTypes.Withdraw2Contract` struct in the `orderly-contract-evm` repo has a `uint256 clientId` parameter, which is used instead of `periodId`.

However, the `orderly-evm-cross-chain` repo attempts to read `periodId` from this struct in the `VaultCrossChainManagerUpgradeable.receiveMessage` function, causing a `TypeError` compilation error.

Recommendation

Update the `orderly-evm-cross-chain` repo to reflect the changes in the `orderly-contract-evm` repo.

Resolution

Orderly Team: The issue was resolved in commit 7d71522.

H-01 | withdraw2Contract Crosschain Flow Pays Withdrawal Fee Twice

Category	Severity	Location	Status
Configuration	High	Global	Resolved

Description

In the `withdraw2Contract` crosschain flow, the `Ledger` credits the fee collector's account with the withdrawal fee in `executeWithdraw2Contract`, then credits the same fee again in `accountWithDrawFinish`.

As a result, a single user withdrawal leads to a double fee charge in the Ledger's accounting, once when the funds are initially frozen and again when the ledger finalizes the withdrawal. This inflates the fee collector's balance with inexistent funds.

Recommendation

Remove one of the two fee credits so that the fee is only applied once. For example, either credit the fee collector immediately on `executeWithdraw2Contract` and avoid doing so in `accountWithDrawFinish`, or defer the fee credit until final settlement.

Resolution

Orderly Team: The issue was resolved in the merge request 327.

H-02 | depositToStrategy Function Will Always Revert

Category	Severity	Location	Status
Logical Error	High	ProtocolVault.sol: 237	Resolved

Description

The `depositToStrategy` function sends a deposit request to the `dexVault` via `IDexVault(dexVault).depositTo(...)`, transferring USDC (or another token) in the process.

However, there is no approval step for the vault to pull tokens from this contract. Without an ERC-20 approve call, the `dexVault` has no permission to transfer tokens on behalf of the `ProtocolVault`. As a result, the deposit call will always revert.

Recommendation

Approve the `dexVault` before calling the `depositTo` function.

Resolution

Orderly Team: The issue was resolved in commit f6381c3.

H-03 | Newly Added Strategy Provider Will Not Receive LP Deposits

Category	Severity	Location	Status
Logical Error	High	ProtocolVaultLedger.sol: 335	Acknowledged

Description

Under the current proportional deposit logic in `allocatToFunds`, the contract allocates newly deposited LP assets among strategies based solely on the main vault's existing shares (i.e., `mainShares`).

When a new strategy provider is introduced at a later period (for example, in the period 5), it starts with zero main shares. Consequently, the formula:

```
portionForThisStrategy = pendingLpDepositAssets * (mainAssetsInFund / totalMainAssetsInFund)
```

will yield zero for that strategy. The new strategy never accumulates any main vault capital automatically, as it isn't part of the existing distribution ratio (which depends on `mainShares`).

Even if the new strategy invests its own capital (SP deposit), that action mints strategy provider shares, not main shares, thus it does not affect the main vault ratio or future LP deposit splits. As a result, this new strategy remains perpetually excluded from LP inflows.

Recommendation

Incorporate a method for the operator to allocate a desired seed amount of main vault capital to a newly added strategy (e.g., "rebalance from existing strategies" to ensure the newcomer starts with a non-zero main share).

Resolution

Orderly Team: Acknowledged.

H-04 | Wrong Cross-chain Manager Usage

Category	Severity	Location	Status
Configuration	High	LedgerImplC.sol	Resolved

Description

`LedgerImplC` holds the logic for Solana withdrawals and `withdraw2Contract` withdrawals. It correctly uses `crossChainManagerV2Address` inside `executeWithdrawSolAction()` to execute a Solana withdrawal.

However, it uses the same `crossChainManagerV2Address` inside `executeWithdraw2Contract()` while the `withdraw2Contract()` function is implemented in `crossChainManagerAddress`.
In result, `withdraw2Contract()` will always fail.

Recommendation

Replace `crossChainManagerV2Address` with `crossChainManagerAddress` inside `executeWithdraw2Contract()`.

Resolution

Orderly Team: The issue was resolved in the merge request 328.

H-05 | DOS Of updateLPAndStrategyFund

Category	Severity	Location	Status
Logical Error	High	ProtocolVaultLedger.sol	Resolved

Description

`LP_WITHDRAW` and `SP_WITHDRAW` inside `ProtocolVaultLedger.handleOpFromVault()` should only be allowed if the user has enough withdrawable shares.

This is handled by `_checkWithdraw()` - it ensures the amount to be withdrawn added to the current frozen amount doesn't surpass the share balance of the account.

After that, the amount to be withdrawn is added towards `frozenShares` so `checkWithdrawal` will continue to work properly for further withdrawal requests.

The withdrawal request will be handled by `_handleLpWithdraw()` and the amount to be withdrawn will be subtracted by the user's `pendingShares` and `frozenShares`. After some time, `settleAccounts()` will update the user's actual shares by setting them to `pendingShares`.

This creates a window between `_handleLpWithdraw()` and `settleAccounts()` where `frozenShares` is decreased, but `account.shares` is not updated.

Any withdrawal request in that window will successfully performed if it doesn't exceed the user's shares because `_checkWithdraw()` won't stop it. This will result in an increase in `frozenShares`, potentially doubling the current value.

The withdrawal request inside this window will be processed inside `_handleLpWithdraw()` for the next period.

The amount to be withdrawn will be subtracted from `pendingShares` again, however this time `pendingShares` will not cover it causing a revert and DOS of the `updateLPAndStrategyFund` function.

Recommendation

One possible solution may be to disallow withdrawal requests in that window.

Resolution

Orderly Team: The issue was resolved in commit 2ff9be9.

H-06 | Withdraw Message Sent Even If _validReceiver Check Fails

Category	Severity	Location	Status
Validation	High	Vault.sol	Acknowledged

Description

In the `Vault.withdraw` function `IVaultCrossChainManager(crossChainManagerAddress).withdraw(vaultWithdrawData)` is called before verifying that the `receiver` is valid.

Consequently, if `_validReceiver(data.receiver, address(tokenAddress))` returns `false`, the vault still sends a cross-chain message to the ledger acknowledging a "successful" withdrawal.

Meanwhile, the local code emits only a `WithdrawFailed` event and never transfers the tokens to the `ProtocolVault`. This leaves the ledger believing the user's withdrawal went through, while in reality no tokens were actually delivered.

As a result, the user's ledger state and on-chain vault state become out of sync. The ledger sees a final "withdraw finish," but the vault never transferred tokens if the receiver check fails. This scenario could strand the user's funds or require an off-chain correction.

Recommendation

If the vault does not transfer tokens because `_validReceiver(data.receiver, address(tokenAddress))` returned false, consider sending a "withdraw failure" message back to the ledger or omit sending a "successful" cross-chain call.

This ensures the ledger and vault remain consistent, reflecting that the withdrawal did not finalize.

Resolution

Orderly Team: Acknowledged.

M-01 | SP Deposits Are Not Refunded

Category	Severity	Location	Status
Configuration	Medium	ProtocolVaultLedger.sol: 141	Resolved

Description

In the handleOpFromVault function of ProtocolVaultLedger, whenever a deposit operation targets a strategy provider (spId) that is not marked as allowed (isAllowedStrategyProvider[spId] = false), the contract simply emits an event and returns.

This silent return means that the deposited tokens, already locked on the ProtocolVault side are not refunded to the strategy depositor. As a result, funds end up stuck, creating a loss scenario for the depositor.

Recommendation

Consider incorporating a refund logic for the deposited assets to cover this edge case.

Resolution

Orderly Team: The issue was resolved in commit 63d9a53.

M-02 | Missing Payable Modifier Function

Category	Severity	Location	Status
Configuration	Medium	ProtocolVault.sol: 237, VaultCrossChainManager.sol: 86	Resolved

Description

In the `depositToStrategy` function, the contract attempts to call `IDexVault(dexVault).depositTo{value: fee}(...)` but the function itself is not declared as `payable`. As a result, it cannot receive native assets in the current transaction, causing a revert if no native assets are already in the contract's balance.

This breaks the expected flow of paying a fee at runtime, preventing the contract from funding the `DexVault` deposit call.

Furthermore, in the `VaultCrossChainManager` contract's `_lzReceive` function, the `ASSETS_DISTRIBUTION` case calls `IProtocolVault(vault).depositToStrategy(...)` but does not supply `msg.value` for the fee.

Recommendation

Add the `payable` modifier to `depositToStrategy` so that it can accept native assets in the same transaction.

At the same time, ensure that the cross-chain message in `VaultCrossChainManager` includes the fee in `msg.value` when relaying `ASSETS_DISTRIBUTION`, so the contract handles and transfers the deposit fee to the `DexVault`.

Resolution

Orderly Team: The issue was resolved in commit e311386.

M-03 | Operators Could Replay Signatures In The Contract

Category	Severity	Location	Status
Configuration	Medium	ProtocolVaultLedger.sol	Resolved

Description

The `ProtocolVaultLedger` contract relies on its backend to provide signatures for state-changing functions, such as `updateLPAndStrategyFund`, `allocatToFunds` and similar functions, but it does not seem to enforce strict replay protection.

Once a valid signature has been used, an operator can potentially reuse that same signature multiple times (even in different contexts) to reapply changes or to trigger previously authorized operations again within the same period.

This opens the door for double handling of deposit/withdraw operations or other malicious state transitions. On the other hand, the `vaultId` is derived from the vault address and broker hash.

All protocol vaults are deployed to the same address using `CREATE3` and the `brokerHash` is a hardcoded value. As a result, the `vaultIds` of all protocol vaults across every EVM chain are identical and their accounting is tracked as a single account on the ledger chain.

None of the signature verifications include the chain ID. Since `vaultIds` are identical across all chains, a valid signature on one chain will also be valid on another chain for the same `periodId`.

Recommendation

Implement a nonce or sequential counter mechanism for each signature payload, incrementing a contract-stored counter after each valid call. Moreover, consider adding `chainIds` to engine signatures.

Resolution

Orderly Team: The issue was resolved in commit 3275584.

M-04 | No Enforcement Of Required Bridging Fee In sendMessage

Category	Severity	Location	Status
Validation	Medium	VaultCrossChainManager.sol: 47	Resolved

Description

The `sendMessage` function in `VaultCrossChainManager` calculates a `MessagingFee` via `_quote` but never checks whether the user-supplied `msg.value` actually matches the required bridging fee

Because there is no comparison between `messageFee.nativeFee` and `msg.value`, the function can be underfunded without reverting.

Recommendation

Enforce that `msg.value` is at least equal to the calculated bridging fee. For example, add a check like `require(msg.value = messageFee.nativeFee, "Insufficient bridging fee");` to ensure that the user has paid for the fee.

Resolution

Orderly Team: The issue was resolved in commit b13ebd6.

M-05 | Strategy Deposit Might Fail Due To Limit

Category	Severity	Location	Status
Logical Error	Medium	Vault.sol: 205-210	Acknowledged

Description

Asset distributions to strategies are initiated by the operators on the ledger chain. The message is then transferred to the vault chain, where the `depositToStrategy` function triggers asset movements from the `protocolVault` to the `dexVault`.

There is a time lag between an operator initiating the process on the ledger chain and the actual execution of the transfer on the vault chain.

Even if the operator initiates the process with valid distribution amounts, the `Vault.depositTo` function might revert with a `DepositExceedLimit` error due to ongoing deposits during this time lag.

For example:
• `dexVault` deposit limit: 1000
• Current balance of the `dexVault`: 850
• Operator calls asset distribution with 100 on the ledger chain (valid amount at the time of initiation).
• Regular users deposit 60 more until this message reaches to vault chain.
• New balance of the `dexVault`: 910
• The distribution transaction fails with `DepositExceedLimit`.
• There is still 90 left in the deposit limit that is not filled.

Since asset distributions can only be called once per period, the operator cannot attempt to distribute assets with a lower value. As a result, assets in the `protocolVault` remain unused.

Recommendation

Consider implementing a separate deposit function in the `dexVault` for strategy deposits. This function should not revert if the full amount cannot be deposited; instead, it should deposit the available amount up to the limit.

Resolution

Orderly Team: Acknowledged.

M-06 | FeeRate Changes Lead To Loss Of Yield

Category	Severity	Location	Status
Logical Error	Medium	ProtocolVaultLedger.sol	Resolved

Description

The `setFeeRate` function allows changing performance fee rates during an active period, which can result in users being charged different rates than what they initially agreed to. When users deposit funds, they implicitly agree to the current fee structure.

However, if the fee rate is modified mid-period via `setFeeRate`, users will be charged the new rate when performance fees are calculated in `updateStrategyFundAssets`, even though this wasn't the rate in effect when they deposited.

For example:
1. User deposits when fee rate is 20%
2. Mid-period, owner calls `setFeeRate` to change rate to 30%
3. At period end, performance fees are calculated using 30% rate
4. User pays higher fees than they agreed to when depositing

Recommendation

Only allow fee rate changes to take effect in future periods. Or restrict fee rate changes to only occur after the current period's performance fees have been calculated.

Resolution

Orderly Team: The issue was resolved in commit 26e43f9.

M-07 | Native Funds Locked In Contract

Category	Severity	Location	Status
Logical Error	Medium	OAppSenderUpgradeable.sol	Resolved

Description

In the `_lzSend` function, the `_refundAddress` parameter is set to the contract's address, but the contract lacks functionality to withdraw or rescue these refunded ETH funds. When `LayerZero` returns excess fees to the contract address, they will be not be retrievable

Recommendation

Consider implementing a pull method for users to receive their refund. Or add a rescue function so that admin can recover the locked ETH.

Resolution

Orderly Team: The issue was resolved in commit b13ebd6.

M-08 | Event Emitted With Incorrect Values

Category	Severity	Location	Status
Validation	Medium	ProtocolVaultLedger.sol: 450	Resolved

Description

The `MainAndStrategyFundsSettled` event's first parameter should represent the `mainAssets` amount. However, it is currently emitted with the `periodId`. Since the protocol's backend heavily relies on event emissions, this issue may cause incorrect operations on the backend.

Recommendation

Update the event.

Resolution

Orderly Team: The issue was resolved in commit d4b54b2.

M-09 | A Portion Of Frozen Fees Will Remain Frozen

Category	Severity	Location	Status
Logical Error	Medium	LedgerImplC.sol	Acknowledged

Description

During the `withdraw2contract` flow the amount of funds and the fee will be frozen. Then at the end of the flow these amounts are intended to be unfrozen. However, there will be small difference between the amount frozen and unfrozen.

This will happen when `convertDecimal` is called and the destination chain decimals are less than the sending chain. In this case there will be some precision loss and while X amount of funds are frozen at the beginning only X - Y will be unfrozen. Where Y equals the precision loss.

Recommendation

Adjust fee amount and withdraw amount so that there is no precision loss prior to freezing the funds. This can be done by truncating the amount to the destination decimals and then expanding it back to the senders decimals. This will ensure that the amount frozen and unfrozen are the same.

Resolution

Orderly Team: Acknowledged.

M-10 | Period Update Breaks When Batching

Category	Severity	Location	Status
Logical Error	Medium	ProtocolVaultLedger.sol 163	Acknowledged

Description

When `updateStrategyFundAssets` is called it will iterate through all funds. However given that there is no hard cap on the number of funds the protocol can have and that fund creation will become permissionless it will eventually require multiple iterations to update all the funds.

The issue with this is that `mainAssetsAfterFees` will become much less than its actual value on the second call since it will not take into account `mainAssetsAfterFees` from the first call. The reduced `mainAssetsAfterFees` will drastically reduce users share value.

Recommendation

Modify the `updateStrategyFundAssets` function so that it can be called multiple times without losing data from previous `updateStrategyFundAssets` calls.

Resolution

Orderly Team: Acknowledged.

M-11 | Rounding Up Will DoS When Funds Are Withdrawn

Category	Severity	Location	Status
Logical Error	Medium	ProtocolVaultLedger.sol 415	Acknowledged

Description

When calculating `distributeWithdrawAssets` the value is rounded up. Because of this the amount of assets being distributed can be larger than the actual amount of funds.

In some cases `distributeWithdrawShares` rounding down will offset this and there won't be excess funds distributed.

But in situations where `distributeWithdrawAssets` does have a remainder causing the value to round up and `distributeWithdrawShares` does not have remainders resulting in no amount being rounded down.

More assets will be distributed then intended. During times where all funds are withdrawn transferring an amount that is greater than what is available will lead to a failed transaction.

Recommendation

Consider rounding down when calculating `distributeWithdrawAssets`.

Resolution

Orderly Team: Acknowledged.

M-12 | Enabling Tokens Breaks Protocol

Category	Severity	Location	Status
Configuration	Medium	ProtocolVault.sol	Acknowledged

Description

`ProtocolVault.setAllowedToken()` lets the owner of the contract enable or disable a new token for deposits. When users perform deposits with this token, they will pay an `amount` of that token.

However, the whole system currently is setup to work with USDC. For example, `_getOperationData` hardcodes the `tokenHash` to `USDC_HASH`. If another token were to be enabled, users will be charged that token, but their balance of the USDC token will be increased on the Ledger side instead.

Recommendation

If you should support multiple tokens, consider not hardcoding the token hashes. Be careful with this approach because some tokens may have different decimals across chains.

Resolution

Orderly Team: Acknowledged.

M-13 | Token Losses On Deposit

Category	Severity	Location	Status
Validation	Medium	Vault.sol	Acknowledged

Description

Users can use `Vault._deposit()` to deposit tokens from the vault side to the ledger side. They will be charged an amount of these tokens on the vault side.

Since this token may have different decimal precision on each chain, the amount added towards the user balance on the ledger side is adjusted by `convertDecimal`

In case `srcDecimals > dstDecimals`, the amount will be divided to convert it to `dstDecimals` and the rest will be lost. This adjusted amount will also be recorded in the `vaultManager` for the given `srcChainId`. In result, users will lose part of their tokens.

Recommendation

Consider adding a `convertDecimal` function to the `Vault` as well and charging the user the newly adjusted amount.

Resolution

Orderly Team: Acknowledged.

M-14 | Vault LZ Fee Can Be Lost

Category	Severity	Location	Status
Validation	Medium	ProtocolVault.sol	Resolved

Description

When `ProtocolVault.depositToStrategy()` is called, the `dexVault.getDepositFee()` will be forwarded to `dexVault. depositTo()`. The vault will then use that value to pay for LZ fees.

However, the vault has a `depositFeeEnabled` boolean. It will use the `msg.value` send to `depositTo()` to pay for the fees only if this flag is set to true. Otherwise, the sent native token will not be used and remain stuck in the contract.

Recommendation

Forward the fee from `ProtocolVault` to `Vault` only if `depositFeeEnabled = true`.

IMPORTANT: If you implement this fix, any value provided by the executor to pay the fees will now be stuck in `ProtocolVault`. You should come up with a solution for these funds. You can:
• transfer the fee to the vault if `depositFeeEnabled = true`
• otherwise transfer it to the `VaultCrossChainManagerUpgradeable`

Resolution

Orderly Team: The issue was resolved in commit f7648f0.

M-15 | Insufficient msgOptions

Category	Severity	Location	Status
Configuration	Medium	VaultCrossChainManager.sol	Acknowledged

Description

`VaultCrossChainManager.sendMessage()` will use the `msgOptions` mapping to determine what gas and value the executor should use for executing `lzReceive` on the destination chain. The values used is chosen based on the message's `payloadType`.

However, they are the same for each chain. Some chains may require different parameters. While it may be fine for most EVM chains, sending messages to Solana is different.

Instead of `gas_limit` and `msg.value`, the values used for Solana will be `compute_units` and lamports which is quite different.

Reference: https://docs.layerzero.network/v2/developers/solana/gas-settings/options

Recommendation

Consider having different `msgOption` values for different chains (or at least Solana).

Resolution

Orderly Team: Acknowledged.

M-16 | Insufficient Validation In withdraw2Contract

Category	Severity	Location	Status
Validation	Medium	LedgerImplC.sol	Partially Resolved

Description

`LedgerImplC.withdraw2Contract()` doesn't implement the withdrawal validation implemented in `LedgerImplA.executeWithdrawAction()`. This poses a significant risk because of the `withdrawNonce`. Since its not validated, a lower nonce than the current last value may be used.

This will then lead to overriding the last withdrawal nonce with the new value (which is way lower) and will enable past withdrawals to be executed again. The fee is also not validated which means it can exceed the maximum configured fee.

Recommendation

Consider implementing validation for the two things mentioned in the report.

Resolution

Orderly Team: The issue was resolved in the merge request 329.

L-01 | Wrong Check In _convertToShares Function

Category	Severity	Location	Status
Configuration	Low	ProtocolVaultLedger.sol: 832	Resolved

Description

In the `_convertToShares` function, the condition currently checks whether (`_toatlShares = 0`) to decide if the vault is in a “first deposit” scenario.

Recommendation

Update the `_convertToShares` function to compare `_totalAssets = 0` rather than `_toatlShares = 0`.

Resolution

Orderly Team: The issue was resolved in commit 5d09f50.

L-02 | quoteOperation Function Always Assumes a LP_DEPOSIT Payload

Category	Severity	Location	Status
Configuration	Low	ProtocolVault.sol: 326	Resolved

Description

Within `ProtocolVault` contract, the `quoteOperation` function hardcodes `LP_DEPOSIT` as the `PayloadType` to calculate bridging fees.

Recommendation

Update the `quoteOperation` function to accept a `payloadType` parameter or determine it dynamically if needed, thereby ensuring the calculated bridging fee matches the actual operation type.

Resolution

Orderly Team: The issue was resolved in commit b61d5e4.

L-03 | ProtocolVault Claim Function Can Transfer Any Locked Token

Category	Severity	Location	Status
Validation	Low	ProtocolVault.sol: 206	Resolved

Description

In the `claim` function, the user provides a token address in `claimParams.token` without any validation that it matches the asset they previously deposited.

Because the contract simply performs `SafeTransferLib.safeTransfer(ERC20(claimParams.token), msg.sender, amount)`, a malicious user can specify any token owned by the `ProtocolVault`, claiming funds that do not necessarily belong to them.

Recommendation

Restrict the token being claimed to the actual asset recorded for the user in the internal ledger (e.g., by storing the token in `userClaimedById[id]` and only transferring that one).

Resolution

Orderly Team: The issue was resolved in commit 8ef737e.

L-04 | Possible Inconsistent Decimal Precision Configuration

Category	Severity	Location	Status
Validation	Low	ProtocolVaultLedger.sol	Resolved

Description

The `setDecimal` function allows updating three separate decimal values—`priceDecimal`, `shareDecimal`, and `assetsDecimal`—independently. If these values are set to different scales, the accounting logic will be broken.

Moreover, `accountToken` decimals should be always the same as `priceDecimal`. Finally, `assetsDecimal` state variable is not really used across the contract's logic so it can simply be removed.

Recommendation

Enforce that `_priceDecimal`, `_shareDecimal`, and `_assetsDecimal` remain the same, or remove the function altogether if dynamic decimal reconfiguration is not a valid operational case.

Ensure that `accountToken` decimals is equal to `priceDecimal`. Consider removing the `assetsDecimal` state variable.

Resolution

Orderly Team: The issue was resolved in commit 263831c.

L-05 | Redundant Period ID Parameter

Category	Severity	Location	Status
Code Best Practices	Low	ProtocolVaultLedger.sol	Acknowledged

Description

The `_check(uint256 periodId)` function in the `ProtocolVaultLedger` contract compares `periodId` against `latestPeriodId`, but this extra parameter is superfluous.

Since the contract already tracks the currently active period in `latestPeriodId`, requiring an extra parameter in many functions that is later on validated through the `_check` function introduces unnecessary complexity.

Recommendation

Remove the `_check` function and the redundant `periodId` parameter for all the functions. Instead, directly reference `latestPeriodId` wherever period alignment is needed.

Resolution

Orderly Team: Acknowledged.

L-06 | Proportional Allocation Overfunds Past Performers

Category	Severity	Location	Status
Configuration	Low	ProtocolVaultLedger.sol: 357	Acknowledged

Description

In the ProtocolVaultLedger's current design, new LP deposits are allocated proportionally to each strategy's existing "main vault" share, rewarding historically successful strategies with a continually larger share of new deposits.

This works well if a strategy's outperformance persists, but it can backfire when a once-top performer's yield dwindles or fails.

For example, if Strategy1 significantly outperforms Strategy2 over the first 20 periods, it ends up with a much larger share of the main vault's capital.

Consequently, even if Strategy1's yield drops to near zero afterward, it continues to receive a high fraction of new LP deposits for many subsequent periods—because the contract only looks at the legacy ratio of main shares in each strategy.

This can cause a suboptimal capital deployment where fresh user funds flow into a no-longer-productive strategy.

Recommendation

Consider implementing an operator function to realign capital if a strategy's yield clearly stagnates, preventing capital from staying locked in a once-top performer.

Resolution

Orderly Team: Acknowledged.

L-07 | Minor Unallocated Remainders In Proportional LP Allocation

Category	Severity	Location	Status
Precision Loss	Low	ProtocolVaultLedger.sol: 357	Acknowledged

Description

When splitting `pendingLpDepositAssets` among multiple strategies using `mulDiv(..., Math.Rounding.Floor)`, each proportional slice may be truncated downward.

Summing these truncated allocations for all strategies often leaves a small leftover in `pendingLpDepositAssets` that never gets allocated.

Over many periods or multiple strategies, these tiny unallocated remainders can accumulate, causing a minimal mismatch between the total deposit intended and the amounts actually distributed.

Therefore a very small amount of user-deposited capital remains undistributed.
This issue also applies to withdrawals(`pendingLpWithdrawShares`).

Recommendation

After allocating to all but one strategy, assign the final strategy whatever remains of `pendingLpDepositAssets` to ensure there is no remaining dust.

Resolution

Orderly Team: Acknowledged.

L-08 | Debugging Checks And Test Code Left In Production Code

Category	Severity	Location	Status
Code Best Practices	Low	Global	Acknowledged

Description

In multiple contracts, there are code snippets which are presumably for debugging or testing. Such debug checks should not remain in the live, production version of the contract.

Recommendation

Remove all these code snippets. If they are valuable for testing, maintain them in a separate test-only version of the contracts.

Resolution

Orderly Team: Acknowledged.

L-09 | Lack Of A Double Step TransferOwnership Pattern

Category	Severity	Location	Status
Code Best Practices	Low	Global	Resolved

Description

The current ownership transfer process for all the contracts inheriting from the `Ownable` or `OwnableUpgradeable` contracts involves the current owner calling the `transferOwnership` function.

If the nominated EOA account is not a valid account, it is entirely possible that the owner may accidentally transfer ownership to an uncontrolled account, losing the access to all functions with the `onlyOwner` modifier.

Recommendation

It is recommended to implement a two-step process transfer ownership process where the owner nominates an account and the nominated account needs to call an `acceptOwnership` function for the transfer of the ownership to fully succeed.

This ensures the nominated EOA account is a valid and active account. This can be easily achieved by using OpenZeppelin's Ownable2Step contract.

Resolution

Orderly Team: Resolved.

L-10 | CCTP depositForBurn Has Maximum Burn Per Transaction

Category	Severity	Location	Status
Validation	Low	Vault.sol: 387	Resolved

Description

In the `rebalanceBurn` flow, the contract relies on Circle's CCTP method `depositForBurn` for transferring tokens from one chain to another.

However, CCTP enforces a per-transaction burn limit (a maximum USDC amount that can be burned at once) to mitigate risk and manage capacity on the Circle side.

If the protocol attempts to deposit and burn an amount exceeding that limit, the call to `ITokenMessenger(tokenMessengerContract).depositForBurn()` will revert, preventing the rebalancing from succeeding.

Recommendation

Make clear in the protocol's user interface that a single rebalancing transaction is constrained. Operators should plan rebalancing flows accordingly.

Resolution

Orderly Team: Resolved.

L-11 | Floating Pragma

Category	Severity	Location	Status
Code Best Practices	Low	Global	Acknowledged

Description

Contracts should be deployed with the same compiler version and flags used during development and testing. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma.

For example, an outdated pragma version might introduce bugs that affect the protocol negatively. All the contracts in scope are using the following floating pragma: `pragma solidity ^0.8.18;`

Recommendation

Consider locking the pragma version in all the smart contracts. It is not recommended to use a floating pragma in production. For example: `pragma solidity 0.8.28`.

Resolution

Orderly Team: Acknowledged.

L-12 | Incompatibility With CREATE2 On ZkSync

Category	Severity	Location	Status
Configuration	Low	VaultFactory.sol	Acknowledged

Description

The VaultFactory contract relies on CREATE2 deterministic deployments (or CREATE3 via solady library) to produce predictable addresses.

However, zkSync has its own nuances for CREATE2 instruction usage, documented at zkSync's "Differences in EVM instructions". Therefore, this version of VaultFactory should not be used in ZkSync.

Recommendation

If planning to deploy in ZkSync consult the official zkSync docs to adapt or replace the current VaultFactory logic with a mechanism that is officially supported and yields consistent results.

Resolution

Orderly Team: Acknowledged.

L-13 | Griefing Of Vault Deposits

Category	Severity	Location	Status
Validation	Low	Vault.sol	Acknowledged

Description

The `_deposit()` function in `Vault.sol` will revert if the balance of the contract after the deposit would exceed the `tokenAddress2DepositLimit` set by the owner. This can be manipulated by external party by sending tokens directly to the vault and DOS-ing deposits.

Recommendation

Introduce internal token deposits tracking instead of using `balanceOf`.

Resolution

Orderly Team: The issue was resolved in the merge request 330.

L-14 | Unfair Performance Fee Distribution

Category	Severity	Location	Status
Logical Error	Low	ProtocolVaultLedger.sol	Acknowledged

Description

In situations where a SP underperforms while there is an increase in shares it will take a weighted average of the previous HWM and the HWM of the incoming increase. Although this does lower the HWM it will still be greater than the share price that the incoming depositors are entering at.

Because of this incoming depositors can experience an increase in share price (profit) without paying any performance fee if the higher HWM is not exceeded. This essentially gives any user the opportunity to participate in the protocol profit off the SP's strategy without paying any fees.

Recommendation

Document that the performance fee burden is not always fair amongst LP’s when the fund moves from underperforming to not underperforming. Additionally monitor activity if the attack becomes an issue consider implementing incentives for LP’s that start and stay with underperforming funds.

Resolution

Orderly Team: Acknowledged.

L-15 | No Paused Check For Withdrawals

Category	Severity	Location	Status
Configuration	Low	Vault.sol	Acknowledged

Description

Withdrawals in the `Vault` contract check if the receiver of the token is blacklisted and if they are, the `WithdrawFailed` event will be emitted to credit the sender back their tokens on the Ledger side.

However, there is no check if the token contract is currently paused. If it is, the sender will have to wait until the contract gets unpaused even though the token may not be paused on other Orderly supported chains.

Recommendation

Check if the contract is paused, just like you are checking if the receiver is blacklisted.

Resolution

Orderly Team: Acknowledged.

L-16 | Centralization Risks

Category	Severity	Location	Status
Configuration	Low	Global	Acknowledged

Description

The share price is calculated based on the total assets value provided by the backend. Operators can set the share price to arbitrary values by providing incorrect total asset amounts.

Additionally, the `setFeeRate` function does not have an upper limit for fee rates, allowing them to be set even above 100%. Users also cannot access their funds that were deposited into the `protocolVault` until the deposits are handled in the period logic.

Because of this the protocol can delay or not perform period updates and cause the users funds to be stuck in the `protocolVault`.

Recommendation

Users of the protocol should be aware of these centralization risks.

Resolution

Orderly Team: Acknowledged.

L-17 | Multiple Typos

Category	Severity	Location	Status
Code Best Practices	Low	Global	Resolved

Description

`ProtocolVault` contract L235: “cal dex” should be “call dex”. `ProtocolVaultLedger` contract lines 685, 826 and 834: “`_toatlShares`” should be “`_totalShares`”. `VaultFactory` contract line 38: "with keythe deployer" should be "with the deployer".

Recommendation

Fix typos.

Resolution

Orderly Team: The issue was resolved in commit a528c91.

L-18 | updateUnclaimed Does Not Have Duplicate Check

Category	Severity	Location	Status
Validation	Low	ProtocolVaultLedger.sol: 541	Acknowledged

Description

The `updateUnclaimed` function checks unhandled `requestIds` and creates `userClaimInfos` array based on their length. However, it does not perform a duplicate check for the `requestIds`.

In the case of a duplicate entry, an unhandled `requestId` will be counted twice when calculating the array length but will only be added once to the array, as the first entry will mark that `requestId` as handled. This will result in the `userClaimInfos` array containing empty elements.

Recommendation

Consider implementing a check to prevent duplicate `requestId` entries.

Resolution

Orderly Team: Acknowledged.

L-19 | Salt Is Not Hashed With Deployer Address

Category	Severity	Location	Status
Configuration	Low	VaultFactory.sol	Acknowledged

Description

According to the comments in the code, each deployer should have its own namespace, which is obtained by hashing the salt with the deployer's address. However, this is not the case in the actual code, where the salt is hashed by itself.

Recommendation

Consider hashing the salt with the deployer's address, or update the comments to reflect the current implementation.

Resolution

Orderly Team: The issue was resolved in commit a528c91.

L-20 | Unused Errors And Events

Category	Severity	Location	Status
Code Best Practices	Low	Global	Resolved

Description

• `NotAllowedToken` and `NotEnoughFee` errors in `IProtocolVault`,

• `InsufficientBalance`, `AlreadyAllocatedShare`, `InvalidTotalAssets` errors and `StrategyExecuted` event in `IProtocolVaultLedger` are not used in the codebase and can be removed.

Recommendation

Consider removing unused errors.

Resolution

Orderly Team: The issue was resolved in commit 8b6b665.

L-21 | Warning About Paused DexVault

Category	Severity	Location	Status
Configuration	Low	ProtocolVault.sol: 236-237	Acknowledged

Description

The `distributeAssets` flow invokes `ProtocolVault.depositToStrategy`, which subsequently calls the `DexVault.getDepositFee` and `DexVault.depositTo` functions. The `ProtocolVaultLedger` contract on the Ledger chain does not implement `Pausable`, whereas the `DexVault` on the EVM chain is pausable.

If the broker initiates the `distributeAssets` flow while the `DexVault` is paused, the Ledger chain transaction will succeed, but the EVM part of the transaction will revert.

Consequently, `isAssetDistributed` will be set to true on the Ledger chain, even though the assets remain undistributed.

Recommendation

Be aware of this situation and avoid initiating a transaction on the Ledger chain when the receiver on the EVM chain is paused.

Resolution

Orderly Team: Acknowledged.

L-22 | Unvalidated Params In setAllowedStrategyProvider

Category	Severity	Location	Status
Validation	Low	ProtocolVaultLedger.sol: 606-616	Resolved

Description

The `setAllowedStrategyProvider` function accepts multiple parameters, including `spId`, `vaultId`, the `vault` address, and `brokerHash`. It sets the `spId` and emits the `AllowedStrategyProviderSet` event with these parameters.

Normally, `spId` is derived from these parameters. However, there is no check to ensure that the provided spId matches the ID derived from these parameters.

If the provided values and the `spId` do not match, the function will still execute and emit an event containing misleading values for the backend.

Recommendation

Consider adding a check to ensure that the provided values match the `spId`.

Resolution

Orderly Team: The issue was resolved in commit 27dc0db.

L-23 | Some Functions Can't Cover Fee Costs

Category	Severity	Location	Status
Configuration	Low	ProtocolVaultLedger.sol	Acknowledged

Description

`distributeAssets` and `updateUnclaimed` both will send a message which requires a fee amount. But The functions depend on there being a existing amount in the cross chain contract. This means that both `distributeAssets` and `updateUnclaimed` cant send its own `msg.value` to over the fee.

Recommendation

Consider making these functions payable and give the operator the option to supply some `msg.value`.

Resolution

Orderly Team: Acknowledged.

L-24 | A Hard-fork Can Disrupt Messaging

Category	Severity	Location	Status
Configuration	Low	Global	Acknowledged

Description

If a hard fork occurs while a message is being sent it is possible that the chainId will change. If this were to happen there would be a mismatch in the chainID's impacting the messaging.

Recommendation

Monitor chain upgrades and in the rare cases that the chainId is going to change notify users or pause the protocol for a few blocks prior to the hard fork.

Resolution

Orderly Team: Acknowledged.

L-25 | accountToken.assets Is Never Decreased

Category	Severity	Location	Status
Logical Error	Low	ProtocolVaultLedger.sol	Resolved

Description

`accountToken.assets` is increased in the `handleOpFromVault` function when users deposit. But there is no way for this value to decrease. So regardless if there are withdrawals or not `accountToken.assets` will continue to grow with each deposit.

Recommendation

As `accountToken` is withdrawn consider decreasing the `assets` amount.

Resolution

Orderly Team: The issue was resolved in commit 474e339.

L-26 | No Withdrawal Confirmation For Solana

Category	Severity	Location	Status
Configuration	Low	LedgerImplC.sol	Acknowledged

Description

When withdrawal requests are processed from the Ledger side to the Vault side, the balance of the user is decreased and the amount is frozen. Upon successful confirmation, the frozen value is being zeroed out.

By tracking users' frozen balances, Orderly can increase their real balance back if the withdrawal action failed. This 2-step process is not happening for Solana withdrawals - everything is processed at once in `LedgerImplC.executeWithdrawSolAction()`.

If the withdrawal fails, the frozen balance will be 0 and the user can't get their funds back. In addition, the fee collector is rewarded `fee` amount with the decimal precision of the Ledger side. If there is a difference between these decimals on Solana, further problems may arise.

Recommendation

Be aware of the potential risks

Resolution

Orderly Team: Acknowledged.

L-27 | Market Manager Flag Not Cleared

Category	Severity	Location	Status
Validation	Low	LedgerImplB.sol	Resolved

Description

At the end, `LedgerB.executeProcessValidatedFuturesBatch()` loops over each trade and calls `_writeBackLastFundingUpdatedTimestamp()`.

This function updates the last funding timestamp of the manager and sets the `TSMarketManagerFlag()` to true, which means no more updates for that `tradeHash`.

This value is not cleared after the for loop ends. If multiple calls to `executeProcessValidatedFuturesBatch()` are made in the same transaction, only the first call will update the timestamp, since the transient storage flag will be left as `true`.

Recommendation

Be sure to use the function correctly.

Resolution

Orderly Team: The issue was resolved in the merge request 333.

L-28 | Slight withdrawAssets Discrepancy

Category	Severity	Location	Status
Precision Loss	Low	ProtocolVaultLedger.sol	Resolved

Description

`ProtocolVaultLedger._handleLpWithdraw()` converts the current withdrawal shares to assets and adds them to the appropriate `userClaimInfo`.

Later in the flow, inside the `allocatToFunds()` function, the sum of all withdrawal shares (`pendingLpWithdrawShares`) is converted the same way to assets and the result is subtracted from `pendingState.pendingTotalAssets`.

Because Solidity truncates on division, `convertToAssets(pendingLpWithdrawShares)` may not be equal to `convertToAssets(withdrawShares1) + convertToAssets(withdrawShares2) + ....`

This can lead to a slight discrepancy between the recorded assets to be withdrawn and the actual amount, potentially corrupting the flow because of a wrong result returned by `checkMainAndStrategyFund()`.

Recommendation

Be aware of this behavior.

Resolution

Orderly Team: The issue was resolved in commit ff1ab3.

L-29 | safeApprove() Deprecated

Category	Severity	Location	Status
Code Best Practices	Low	Vault.sol: 305	Acknowledged

Description

SafeERC20::safeApprove() has been Deprecated. The developer note in the function discourages using this function, and instead recommends using safeDecreaseAllowance() and safeIncreaseAllowance().

Recommendation

Use safeIncreaseAllowance() instead of safeApprove().

Resolution

Orderly Team: Acknowledged.

L-30 | Token Deposits Cannot Be Disabled

Category	Severity	Location	Status
Configuration	Low	Vault.sol: 205	Acknowledged

Description

`Vault.deposit()` has a validation that checks if `tokenAddress2DepositLimit` for a token is not zero, before seeing if the deposit limit has been exceeded. This prevents disabling deposits for a specific token, since a deposit limit of zero will bypass the second condition.

Recommendation

Consider adding a flag that will revert if the token is currently disabled for deposits.

Resolution

Orderly Team: Acknowledged.

L-31 | enableDepositFee Can Be Paused

Category	Severity	Location	Status
Configuration	Low	Vault.sol	Acknowledged

Description

`Vault.enableDepositFee()` is a function which changes configuration, but it has the `whenNotPaused` modifier. This will stop the owner of updating the flag when the contract is paused.

Recommendation

Consider removing the modifier.

Resolution

Orderly Team: Acknowledged.

L-32 | Mixed Decimals

Category	Severity	Location	Status
Configuration	Low	ProtocolVaultLedger.sol	Resolved

Description

It's expected that `assetPerShare` and `hwm` in `ProtocolVaultLedger.sol` will be with `priceDecimals`, but currently they will be with `assetsDecimals + priceDecimals - shareDecimals`.

Recommendation

Be aware of that.

Resolution

Orderly Team: The issue was resolved in commit 263831c.

L-33 | Unable To Reinitialize Contracts

Category	Severity	Location	Status
Configuration	Low	Global	Acknowledged

Description

The `initialize` functions of the already deployed contracts won't be executed successfully because they are already initialized. For example, `CrossChainRelayUpgradeable.sol` has added logic in its `initialize()` function.

Recommendation

Remove the `initializer` modifier from the `initialize` function and add the `reinitialize` and `onlyOwner` modifiers.

Resolution

Orderly Team: Acknowledged.

L-34 | Lack Of onlyProxy Modifier

Category	Severity	Location	Status
Code Best Practices	Low	LedgerCrossChainManagerUpgradeable.sol, VaultCrossChainManagerUpgradeable.sol	Acknowledged

Description

`LedgerCrossChainManagerUpgradeable` and `VaultCrossChainManagerUpgradeable` are `UUPSUpgradeable` contracts with `upgradeTo()` functions.

In these contract the `upgradeTo()` function is overridden, but there is no `onlyProxy` modifier to it. This allows direct upgrades to the implementation.

Recommendation

Consider adding the `onlyProxy` modifier.

Resolution

Orderly Team: Acknowledged.

L-35 | Cross-chain Communication May Be Blocked

Category	Severity	Location	Status
Configuration	Low	CrossChainRelayUpgradeable.sol	Acknowledged

Description

The `CrossChainRelayUpgradeable` is a blocking OApp which means that once initiated, a message has to be successfully executed on the destination chain in order for any subsequent message to be received.

If the receiving transaction reverts, the communication channel between the two chains will be blocked until the owner call `forceResumeReceive()`.

A transaction can revert if one of the `require` checks which confirms the correct data is sent reverts, the contract the vault interacts with become paused and etc...

Recommendation

Consider switching to a non-blocking `Oapp`.

Resolution

Orderly Team: Acknowledged.

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