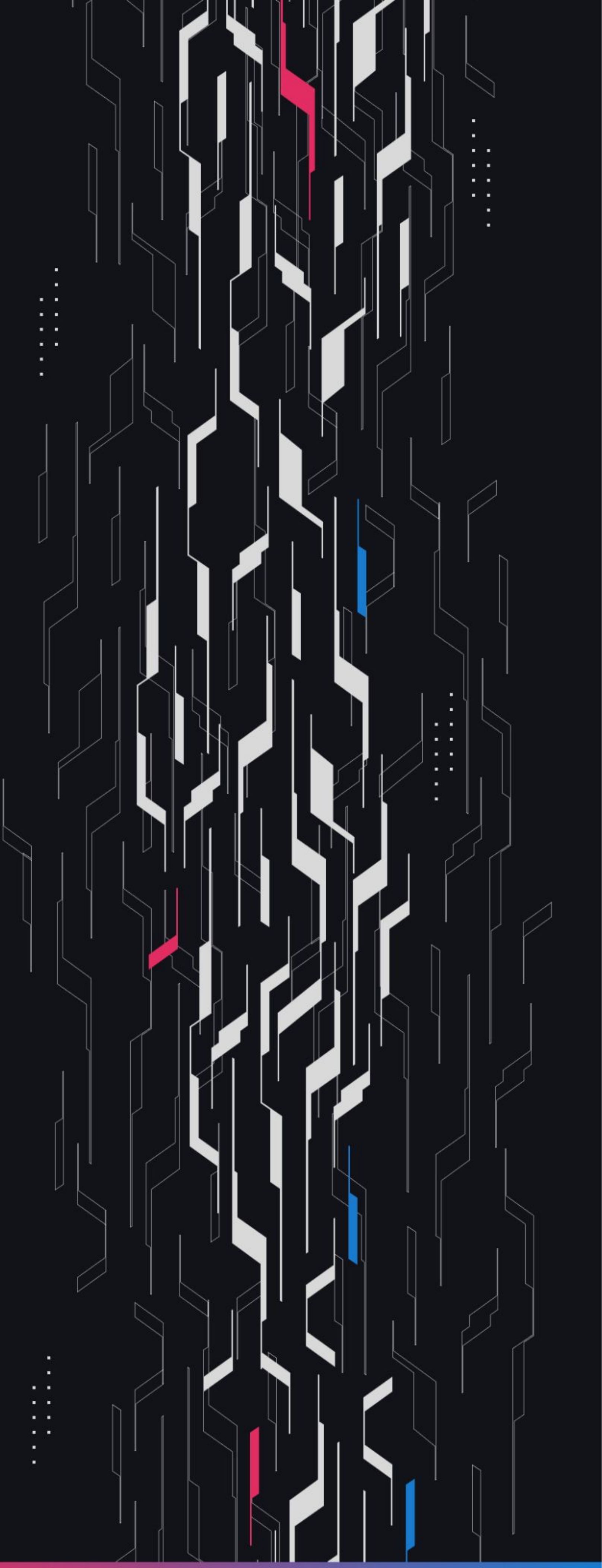
GUARDIAN

Orderly

Solana Vault Review

Security Assessment

November 10th, 2025



Summary

Audit Firm Guardian

Prepared By Robert Regeida, Osman Ozdemir

Client Firm Orderly

Final Report Date November 10, 2025

Audit Summary

Orderly engaged Guardian to review the security of their Orderly Solana Vault Review. From the 10th of October to the 22nd of October, a team of 2 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Given the lack of critical issues detected and minimal code changes following the main review, Guardian assigns a Confidence Ranking of 4 to the protocol. Guardian advises the protocol to consider periodic review with future changes. For detailed understanding of the Guardian Confidence Ranking, please see the rubric on the following page.

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

Guardian Confidence Ranking

Confidence Ranking	Definition and Recommendation	Risk Profile
5: Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. **Recommendation:** Code is highly secure at time of audit. Low risk of latent critical issues.	0 High/Critical findings and few Low/Medium severity findings.
4: High Confidence	Code is clean, well-structured, and adheres to best practices. Only Low or Medium-severity issues were discovered. Design patterns are sound, and test coverage is reasonable. Small changes, such as modifying rounding logic, may introduce new vulnerabilities and should be carefully reviewed. **Recommendation:** Suitable for deployment after remediations; consider periodic review with changes.	0 High/Critical findings. Varied Low/Medium severity findings.
3: Moderate Confidence	Medium-severity and occasional High-severity issues found. Code is functional, but there are concerning areas (e.g., weak modularity, risky patterns). No critical design flaws, though some patterns could lead to issues in edge cases. **Recommendation:** Address issues thoroughly and consider a targeted follow-up audit depending on code changes.	1 High finding and ≥ 3 Medium. Varied Low severity findings.
2: Low Confidence	Code shows frequent emergence of Critical/High vulnerabilities (~2/week). Audit revealed recurring anti-patterns, weak test coverage, or unclear logic. These characteristics suggest a high likelihood of latent issues. **Recommendation:** Post-audit development and a second audit cycle are strongly advised.	2-4 High/Critical findings per engagement week.
1: Very Low Confidence	Code has systemic issues. Multiple High/Critical findings (≥5/week), poor security posture, and design flaws that introduce compounding risks. Safety cannot be assured. **Recommendation:** Halt deployment and seek a comprehensive re-audit after substantial refactoring.	≥5 High/Critical findings and overall systemic flaws.

Table of Contents

Project Overview

Project Summary

Project Name	Orderly
Language	Solidity
Codebase	https://gitlab.com/orderlynetwork/orderly-v2
Commit(s)	**Strategy Vault Main Review commit:** bf11f8eae622315c687ec2bde6e39692c691f129 **Strategy Vault Remediation Review commit:** 883f5d103fa9933327fe2b3cbc95b8ff18499a95 **Omnichain Ledger Main Review commit:** ff0681adfa7a4c937ba8e094a7b5589f78122034 **Omnichain Ledger Remediation Review commit:** d993eec7ad126f67c4056d5358f062ffe92de747 **Contract EVM Main Review commit:** 636b0b733c2cab604b778a07f43b4246977e6166 **Contract EVM Remediation Review commit:** a3ac7aa5b75783b1e92265612f54ce57046c3126

Audit Summary

Delivery Date	November 10, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	0	0	0	0	0	0
● High	1	0	0	0	0	1
● Medium	3	0	0	0	0	3
● Low	18	3	0	3	0	12
● Info	19	10	0	6	0	3

Audit Scope & Methodology

Scope and details:

Repo: `strategy-vault`
New commit: `bf11f8eae622315c687ec2bde6e39692c691f129`
Diff:
https://github.com/GuardianOrg/strategy-vault-team1-1760042507908/compare/fdb4fd2e7284d882a74e93d6248387e0761fa545...bf11f8eae622315c687ec2bde6e39692c691f129

Repo: `omnichain-ledger`
New commit: `ff0681adfa7a4c937ba8e094a7b5589f78122034`
Diff:
https://github.com/GuardianOrg/omnichain-ledger-team1-1760042549133/compare/3345695695bb874f11af2f34b8ea2b820c822dd5...ff0681adfa7a4c937ba8e094a7b5589f78122034

Repo: `contract-evm`
New commit: `636b0b733c2cab604b778a07f43b4246977e6166`
Diff:
https://gitlab.com/orderlynetwork/orderly-v2/contract-evm/-/compare/51a7a28af361865fb2fa7b10833cc9ea5e4c81db..636b0b733c2cab604b778a07f43b4246977e6166?from_project_id=45490567

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.

Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	Broken Multi-Vault Support Due	Logical Error	● High	Resolved
M-01	CCTPv2 Rebalance Credits Ignore Bridge Fees	Logical Error	● Medium	Resolved
M-02	Escrow Bypass In Solana V2 Withdrawal	Validation	● Medium	Resolved
L-01	Native Deposit Overpay Burns Funds	Validation	● Low	Resolved
L-02	Initialize Cannot Be Called	Unexpected Behavior	● Low	Resolved
L-03	Incorrect Timestamp Update	Unexpected Behavior	● Low	Resolved
L-04	CCTP Finality Threshold Misconfiguration	Configuration	● Low	Resolved
L-05	Hardcoded VaultType In Operation Data	Events	● Low	Acknowledged
L-06	CancelAllVestingRequests Messages Are Allowed	Configuration	● Low	Resolved
L-07	UintValue Signatures Lack A Domain Separator	Best Practices	● Low	Acknowledged
L-08	Immutable Withdrawal Address Once Set	Unexpected Behavior	● Low	Resolved
L-09	ClaimEsOrderRevenue Should Check ValorSwitched	Unexpected Behavior	● Low	Resolved
L-10	Backward Message Payloads Are Not Empty	Unexpected Behavior	● Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-11	Missing Validation For Proxy Ledger Destination	Validation	● Low	Resolved
L-12	Vesting Request Lookup Enables User Self-DoS	DoS	● Low	Acknowledged
L-13	Ledger Compose Bypasses Pause Control	Unexpected Behavior	● Low	Resolved
L-14	Vesting Calculation Truncates Twice	Rounding	● Low	Resolved
I-01	Misleading Comment	Informational	● Info	Acknowledged
I-02	Rebalance Slot Reuse Locks Funds	Unexpected Behavior	● Info	Acknowledged
I-03	Delegate Swap Signature Replayable	Signatures	● Info	Acknowledged
I-04	Implementation Setters Don't Validate Code	Best Practices	● Info	Resolved
I-05	Missing Events For Timeline Changes	Events	● Info	Resolved
I-06	Unused Role In Ledger	Superfluous Code	● Info	Resolved
I-07	HWM Decreases When Minting Discounted Shares	Unexpected Behavior	● Info	Acknowledged
I-08	VaultType Mismatch Between Repos	Unexpected Behavior	● Info	Acknowledged
I-09	Redundant setAllowedStrategyProvider Function	Superfluous Code	● Info	Acknowledged

H-01 | Broken Multi-Vault Support Due

Category	Severity	Location	Status
Logical Error	● High	ProtocolVault.sol: 248-254	Resolved

Description PoC

The ProtocolVaultLedger now supports multiple vaults with the latest updates. Each new vault has its own namespaced storage slots, while the protocolVault is accessed through the regular storage. These vaults also have their own brokers, which are stored in the vaultBroker mapping.

Previously, during the distributeAssets flow from the ledger to the vault chains, the StrategyVaultCCMessage did not include the vault address. As a result, the depositToStrategy function on the receiving vault chain was always being called on the protocolVault.

After the multi-vault updates, the StrategyVaultCCMessage now includes the [vault address]((https://github.com/GuardianOrg/strategy-vault-team1-1760042507908/blob/bf11f8eae622315c687ec2bde6e39692c691f129/contracts/Ledger/LedgerCoreImpl.sol#L368)), and that specific address is invoked during _lzReceive:

ttps://github.com/GuardianOrg/strategy-vault-team1-1760042507908/blob/bf11f8eae622315c687ec2bde6e39692c691f129/contracts/VaultCrossChainManager.sol#L145

However, the depositToStrategy function in the ProtocolVault contract still uses the hardcoded ORDERLY_BROKER when retrieving the vaultId and preparing the VaultDepositFE payload. Since these other vaults will not always have ORDERLY_BROKER as their broker, messages sent from the ledger will be misinterpreted on the vault chains.

The same issue is present in the updateUnClaimed and depositFromStrategy functions as well. However, the impact of this issue is limited, as the vaultId is only used for event emission.

Recommendation

To support multiple strategy vaults with different brokers, these broker hashes should also be included in the cross-chain message, and the ProtocolVault should fetch vaultIds using the vault-specific brokers rather than hardcoded ORDERLY_BROKER.

Resolution

Orderly Team: The issue was resolved in commit 1aa2a4b.

M-01 | CCTPv2 Rebalance Credits Ignore Bridge Fees

Category	Severity	Location	Status
Logical Error	● Medium	Global	Resolved

Description

The CCTPv2 migration assumes that the bridge mints exactly the quantity that was burned, but Circle's v2 flow now deducts a fee before minting on the destination chain.

During the burn path the vault approves the full data.amount and when depositForBurn succeeds, forwards that same value back to the ledger

CCTPv2 documentation clarifies that only amount - fee is minted on the destination chain, yet once the mint completes the vault still reports the original data.amount

The Ledger books this figure by calling finishMintToken, which blindly increments tokenBalanceOnchain with the reported amount

As a result, every rebalance inflates the on-chain balance tracker even though the vault's actual holdings lag by the CCTP fee. Over time this mismatch causes withdrawals to revert (vault balance lower than expected) or leaves liabilities unbacked.

The LayerZero message emitted by burnFinish is still necessary: CCTP itself only moves USDC, while the LayerZero channel carries the authoritative status update back to the ledger so it can mark the burn as complete, update accounting and coordinate the matching mint.

Because the payload currently carries the unadjusted burn amount, the ledger adopts the wrong balance.

Recommendation

Capture the net quantity that actually arrives on the destination chain—either by measuring the vault's token balance before and after receiveMessage, or by decoding the CCTPv2 payload to read the minted amount—then pass that net value (and optionally the fee for auditability) into mintFinish and finishMintToken instead of the original data.amount.

Resolution

Orderly Team: The issue was resolved in commit 0290854.

M-02 | Escrow Bypass In Solana V2 Withdrawal

Category	Severity	Location	Status
Validation	● Medium	LedgerImplD.sol	Resolved

Description

The new Solana withdrawal pipeline lives in LedgerImplD._executeWithdraw2SOL. Inside the state block that decides whether a withdrawal can proceed we now only guard on the raw ledger balance.

Compare that to the legacy paths:
• LedgerImplA.executeWithdrawAction (the main EVM route) and
• LedgerImplC.executeWithdrawSolAction (the Solana V1 bridge) both subtract the escrow balance before allowing a payout

That subtraction is critical because internal transfers first credit a user by bumping both account.balances and the escrow bucket:

Only when the matching debit arrives (or the transfer is otherwise finalized) do we release the escrowed units.

Because _executeWithdraw2SOL never subtracts escrowBalances, the Solana V2 route treats those in-flight credits as immediately spendable. A malicious account can:

1. Arrange for an internal transfer (or any engine action that posts a credit) so that account.balances[tokenHash] increases while escrowBalances[accountId][tokenHash] carries the same amount.
2. Call Solana V2 withdrawal for that amount. The existing guard account.balances[tokenHash] < withdrawV2.tokenAmount passes, the withdrawal is frozen/finished, and ILedgerCrossChainManagerV2.withdraw2ContractV2 is invoked to deliver real USDC to Solana.
3. Later, when the original transfer fails or is rolled back, _finalizeTransfer simply clears the escrow entry; there is no clawback because the withdrawal already drained the funds. This way, the attacker double-spent the escrowed amount.

Since the Solana V2 driver also calls finishFrozenBalance immediately (no async result), there is no later chance to detect the deficit.

Similarly, the _executeWithdraw2EVM flow does not check escrow balances, allowing them to be withdrawn.

Recommendation

Keep both the _executeWithdraw2EVM and _executeWithdraw2SOL paths consistent with the other withdrawal implementations by deducting escrow before approving the withdrawal.

Run this check in the same state block that handles nonce/fee validation, reuse the existing error code (state = 9) or call the dedicated revert helper and maintain feature parity across all withdrawal flows. That way, users cannot cash out credits that are still unresolved elsewhere in the engine.

Resolution

Orderly Team: The issue was resolved in commit f493ebf.

L-01 | Native Deposit Overpay Burns Funds

Category	Severity	Location	Status
Validation	● Low	Vault.sol: 399-406	Resolved

Description

The native deposit path trusts whatever msg.value the caller forwards and only checks that the amount covers the requested deposit. Inside _ethDeposit the code derives a crossChainFee as msg.value - data.tokenAmount.

When depositFeeEnabled is false the branch simply calls IVaultCrossChainManager(crossChainManagerAddress).deposit(depositData) without forwarding the fee or returning it:

```
uint128 nativeDepositAmount = msg.value.toUint128();
if (nativeDepositAmount < data.tokenAmount) revert NativeTokenDepositAmountMismatch();
uint256 crossChainFee = nativeDepositAmount - data.tokenAmount;
if (depositFeeEnabled) {
if (crossChainFee == 0) revert ZeroDepositFee();
IVaultCrossChainManager(crossChainManagerAddress).depositWithFeeRefund{value: crossChainFee}(
msg.sender,
depositData
);
} else {
IVaultCrossChainManager(crossChainManagerAddress).deposit(depositData);
}
```

Because the else branch neither refunds crossChainFee nor forwards it, any overpayment remains trapped on the vault contract. Integrators who include a buffer to cover LayerZero costs, mis-estimate the fee, or intentionally round up the value will lose the difference with no indication.

This can lead to user fund loss and undermines the expectation that excess native value is either used for fees or returned.

Recommendation

Type-check native deposits so that excess funds do not accumulate on the contract. Either reject any msg.value above data.tokenAmount when depositFeeEnabled is false, or proactively send the surplus back to the sender after _ethDeposit completes.

Resolution

Orderly Team: The issue was resolved in commit b320fde.

L-02 | Initialize Cannot Be Called

Category	Severity	Location	Status
Unexpected Behavior	● Low	OmnichainLedgerV2.sol: 102-120	Resolved

Description

The `initialize` function in the `OmnichainLedgerV2` contract performs `delegatecalls` to multiple implementation addresses. However, these implementation addresses are only set during `initializeV2`.

Since `initializeV2` is restricted by the `DEFAULT_ADMIN_ROLE`, which is granted only during `initialize`, a fresh deployment cannot set the implementation addresses before initialize runs.

While the admin is already set in the previous version and `initializeV2` can be called during an upgrade, initialize cannot be invoked as part of the upgrade process, rendering that function redundant in this contract.

Recommendation

Consider removing the initialize function from `OmnichainLedgerV2` if only `initializeV2` is intended to be called as part of the upgrade process.

If this contract is intended for use in a fresh deployment setup, ensure that both initialize and `initializeV2` can be called independently and do not rely on values set by the other.

Resolution

Orderly Team: The issue was resolved in commit 16e0881.

L-03 | Incorrect Timestamp Update

Category	Severity	Location	Status
Unexpected Behavior	● Low	MarketManager.sol: 49	Resolved

Description

The updateMarketUpload function in the MarketManager contract can be used to update mark and index prices or the sum of unitary funding.

When it is called with UploadSumUnitaryFundings to update the unitary funding value, the function incorrectly updates the setLastMarkPriceUpdated value instead of setLastFundingUpdated, thereby updating lastMarkPriceUpdated incorrectly.

```
cfg.setSumUnitaryFundings(sumUnitaryFunding.sumUnitaryFunding);
cfg.setLastMarkPriceUpdated(sumUnitaryFunding.timestamp); //@audit-issue should be
setLastFundingUpdated
```

Recommendation

Use the setLastFundingUpdated function in the case of a funding update.

Resolution

Orderly Team: The issue was resolved in commit 5fcc28b.

L-04 | CCTP Finality Threshold Misconfiguration

Category	Severity	Location	Status
Configuration	● Low	Vault.sol	Resolved

Description

Circle's CCTP V2 technical guide (section "Defined Finality Thresholds" - https://developers.circle.com/cctp/technical-guide#defined-finality-thresholds) states that only two values are honored on-chain:

1000 for confirmed/fast transfers and 2000 for finalized/standard transfers.

Any value below 1000 is coerced upward to 1000 and any value above 1000 is coerced to 2000. In our deployment the owner-facing configurator does not enforce this contract; `setCCTPConfig` accepts an arbitrary `uint32` and persists it directly:

```
function setCCTPConfig(uint256 _maxFee, uint32 _finalityThreshold) public onlyOwner {
cctpMaxFee = _maxFee;
cctpFinalityThreshold = _finalityThreshold;
}
```

In production an operator could select 2, 500 or 1234 thinking they were tightening or relaxing finality, but Circle's contract silently maps everything below 1000 to 1000 and everything else to 2000.

Operators therefore believe they have requested "500 confirmations" while the bridge actually runs at the fast tier, exposing rebalances to reorg risk; alternatively, they might expect an intermediate value yet the bridge enforces the slower finalized tier and higher fee.

Recommendation

Guard the setter so only the two valid thresholds are accepted. Expose them as named constants (for example `FINALITY_CONFIRMED = 1000` and `FINALITY_FINALIZED = 2000`) and revert on any other input, or offer an enum that maps to those constants.

Populate sensible defaults per chain and surface the choice explicitly in operator tooling, keeping runtime configuration aligned with the actual CCTP behavior.

Resolution

Orderly Team: The issue was resolved in commit 423b704.

L-05 | Hardcoded VaultType In Operation Data

Category	Severity	Location	Status
Events	● Low	ProtocolVault.sol: 471	Acknowledged

Description

The `vaultType` is hardcoded to `PROTOCOL` in the `_getOperationData` function, even though the codebase now supports multiple vault types after recent updates.

This operation data is used during vault deposits, withdrawals, and the cross-chain messages associated with these actions.

Although `vaultType` is not used for validation on the ledger chain, it results in misleading event emissions for off-chain listeners or engines.

Recommendation

Update the contract to support multiple vault types, or consider using separate contracts for different types.

Resolution

Orderly Team: Acknowledged.

L-06 | CancelAllVestingRequests Messages Are Allowed

Category	Severity	Location	Status
Configuration	● Low	ProxyLedger.sol: 172	Resolved

Description

`CancelAllVestingRequests` message types are still supported on the vault side in the `ProxyLedger.buildEvmVaultMessage` function.

These messages will successfully be sent from the vault side, but they can no longer be processed on the ledger side because the `ledgerRecvFromVault` function does not support them anymore. As a result, users will pay LZ fees for unexecutable actions.

Recommendation

Check the `payloadType` in the `buildEvmVaultMessage` function and disallow type 7.

Resolution

Orderly Team: The issue was resolved in commit 9ec0bbc.

L-07 | UintValue Signatures Lack A Domain Separator

Category	Severity	Location	Status
Best Practices	● Low	Signature.sol	Acknowledged

Description

Signature.verifyUintValueSignature signs only the tuple (value, timestamp) before applying the Ethereum personal-message prefix, so the message that is actually authorized off-chain is.

No deployment-specific data: no chainId, no address(this), no contract-specific salt... appears in the digest. ValorImpl._dailyUsdcNetFeeRevenue and _esOrderRevenueUpdate consume these signatures to authorize revenue uploads.

That signature therefore grants CeFi the ability to mint protocol revenue: every verified message increases totalUsdcInTreasure/totalEsOrderInTreasure, refreshes conversion rates and directly shifts redemption accounting.

The deployment config shows the same ledger stack running on many networks. config/ledger.json enumerates contract/proxy addresses for environments like dev, qa, staging and play across Arbitrum Sepolia, Optimism Sepolia, Polygon Amoy, Base Sepolia, Avalanche Fuji, Ethereum Sepolia, and the Orderly L2. config/oft.json lists the corresponding OFT endpoints.

Because the signer key (usdcUpdaterAddress) is shared across those deployments, any authentically signed revenue update observed on one chain can be replayed on every other chain, minting treasury balances and distorting rates without the signer's consent.

Recommendation

Domain-separate the signed payload so signatures become deployment-specific. Either adopt a full EIP-712 domain separator or augment the current hash.

Including chainId and address(this) (and ideally a type hash or domain salt) prevents the same signature from minting funds on multiple chains.

Resolution

Orderly Team: Acknowledged.

L-08 | Immutable Withdrawal Address Once Set

Category	Severity	Location	Status
Unexpected Behavior	● Low	LedgerOCCManager.sol	Resolved

Description

LedgerOCCManager.setUser2WithdrawAddr lets each user set a custom withdrawal target but forbids any later change:

```
function setUser2WithdrawAddr(address _withdrawAddr) external {
if (user2WithdrawAddr[msg.sender] = address(0)) {
user2WithdrawAddr[msg.sender] = _withdrawAddr;
emit SetUser2WithdrawAddr(msg.sender, _withdrawAddr);
} else {
revert("LedgerOCCManager: withdraw address already set");
}
}
```

Subsequent cross-chain payouts (WithdrawOrderBackward, ClaimUsdcRevenueBackward, etc.) automatically forward tokens to user2WithdrawAddr[user] when it is non-zero.

If the user loses access to that address or it becomes compromised, every future withdrawal is irreversibly redirected there and the contract offers no mechanism, neither for the user nor an administrator, to reset or rotate the mapping. Funds can therefore be permanently lost.

Recommendation

Permit updates to the mapping. Options include allowing the user to supply a signed message authorizing a new address, introducing a timelocked rotation, or enabling an authorized operator to reset entries.

The design must provide a safe path to recover from a compromised or outdated withdrawal address.

Resolution

Orderly Team: The issue was resolved in commit 5eaff7f.

L-09 | ClaimEsOrderRevenue Should Check ValorSwitched

Category	Severity	Location	Status
Unexpected Behavior	● Low	StakingValorRevenueImpl.sol: 38-43	Resolved

Description

The `claimEsOrderRevenue` function collects revenue from `Valor2` earnings and stakes it. It uses the internal function `_collectUserRevenueForClaimableBatch2` to collect the revenue and utilizes `userRevenue2`.

The `userRevenue2` can only increase when a user redeems `Valor2`, and there is no way to earn `\$esORDER` revenue before the protocol transitions to `Valor2`. However, the `claimEsOrderRevenue` function does not verify whether this transition has occurred.

While the function ultimately reverts later in the transaction flow, it is recommended to check the transition status and revert early for better efficiency and clarity.

Recommendation

Check whether the protocol has switched to `Valor2`, and allow the `claimEsOrderRevenue` function to execute only after the switch.

Resolution

Orderly Team: The issue was resolved in commit b49fe08.

L-10 | Backward Message Payloads Are Not Empty

Category	Severity	Location	Status
Unexpected Behavior	● Low	OmnichainLedgerV2.sol: 395, 419, 445, 467, 490	Resolved

Description

When composing EvmLedgerMessage for backward settlement, payload is set to the string literal "0x0". The payload field is bytes, so this produces a non-empty bytes value containing the ASCII characters of 0x0, not a zero-length payload.

```
EvmLedgerMessage memory message = EvmLedgerMessage({
dstChainId: _chainId,
token: LedgerToken.ORDER,
tokenAmount: orderAmountForWithdraw,
receiver: _user,
payloadType: uint8(PayloadDataType.WithdrawOrderBackward),
payload: "0x0" // @audit string literal, not empty bytes
});
```

This payload is not used on the vault side and can be empty bytes.

Recommendation

Consider using empty bytes (bytes("") or new bytes(0)) as in LedgerOCCManager:L198

Resolution

Orderly Team: The issue was resolved in commit 540b07c.

L-11 | Missing Validation For Proxy Ledger Destination

Category	Severity	Location	Status
Validation	● Low	LedgerOCCManager.sol	Resolved

Description

When the ledger sends tokens back to an EVM vault, buildOCCLedgerMsg routes the OFT transfer to chainId2ProxyLedgerAddr[message.dstChainId]:

```
sendParam = SendParam({
dstEid: dstEid,
to: bytes32(uint256(uint160(chainId2ProxyLedgerAddr[message.dstChainId]))),
amountLD: amount,
...
});
```

If that mapping entry is unset, chainId2ProxyLedgerAddr[dst] returns address(0), so the OFT send call delivers the ORDER tokens to the zero address and the compose message also targets zero.

There is no fallback or refund; every withdrawal routed through that path is irrecoverably lost or stuck until the proxy address is configured.

The same risk applies for USDC revenue messages that use the same mapping. Any misconfiguration or forgetting to populate the mapping bricks user withdrawals.

Recommendation

Before building the SendParam, require that the destination proxy is known, for example:

```
address proxy = chainId2ProxyLedgerAddr[message.dstChainId];
require(proxy = address(0), "LedgerOCCManager: proxy not set");
```

Only proceed with the cross-chain send once the mapping entry is present; otherwise revert to prevent funds from being sent to zero.

Resolution

Orderly Team: The issue was resolved in commit 47af923.

L-12 | Vesting Request Lookup Enables User Self-DoS

Category	Severity	Location	Status
DoS	● Low	Vesting.sol	Acknowledged

Description

Every vesting operation relies on _findVestingRequest to locate the user's request by ID:

```
function _findVestingRequest(address _user, uint256 _requestId) internal view returns
(VestingRequest storage) {
for (uint256 i = 0; i < userVestingInfos[_user].requests.length; i++) {
if (userVestingInfos[_user].requests[i].requestId = _requestId) {
return userVestingInfos[_user].requests[i];
}
}
revert UserDontHaveVestingRequest(_user, _requestId);
}
```

There is no limit on how many entries a user can accumulate, createVestingRequest just pushes a new struct onto userVestingInfos[_user].requests. A malicious or inattentive account can create thousands of tiny vesting requests.

Later, when they attempt to call claimVestingRequest or cancelVestingRequest, _findVestingRequest must iterate over the entire array. Once the array grows large enough, that linear scan exceeds the block gas limit and the function reverts.

The user can no longer manage their vesting requests and all pending rewards become inaccessible. This is a self-inflicted DoS stemming from the unbounded linear search.

Recommendation

Avoid O(n) lookups. Track request positions with an additional mapping, for example mapping each (user, requestId) to its index in the array, or switch to a mapping-based storage structure that gives O(1) access.

Alternatively, enforce a sensible upper bound on the number of concurrent vesting requests per user so the loop cannot grow beyond a safe size.

Resolution

Orderly Team: Acknowledged.

L-13 | Ledger Compose Bypasses Pause Control

Category	Severity	Location	Status
Unexpected Behavior	● Low	LedgerOCCManager.sol	Resolved

Description

LedgerOCCManager inherits PausableUpgradeable through LedgerAccessControl, so protocol operators can pause the system. However, the LayerZero inbound handler remains active while paused:

```
function lzCompose(
address _from,
bytes32 /*_guid*/,
bytes calldata _message,
address /*executor*/,
bytes calldata /*_extraData*/
) external payable {
...
ILedgerReceiver(ledgerAddr).ledgerRecvFromVault(evmVaultMessage);
}
```

The function is missing whenNotPaused, so even in a paused state it accepts cross-chain messages, decodes them and forwards them to ledgerRecvFromVault.

This defeats the pause mechanism: an operator cannot truly halt operations during an incident because remote messages continue to be processed. Any safety assumptions about pausing the ledger are therefore invalid.

Recommendation

Guard lzCompose with a pause check, e.g. add the whenNotPaused modifier or call _requireNotPaused() at the start of the function. That way, once pause() is invoked, all inbound LayerZero messages are rejected until the system is explicitly unpaused.

Resolution

Orderly Team: The issue was resolved in commit 23687d4.

L-14 | Vesting Calculation Truncates Twice

Category	Severity	Location	Status
Rounding	● Low	VestingView.sol; Vesting.sol	Resolved

Description

_calculateVestingOrderAmount determines how much $ORDER a user can claim after the lock period. The current implementation breaks the expression into two integer divisions:

```
return _vestingRequest.esOrderAmount / 2 + (_vestingRequest.esOrderAmount * vestedTime) /
vestingLinearPeriod / 2;
```

This corresponds to floor(es/2) + floor((es * vestedTime / vestingLinearPeriod) / 2). Because both terms are truncated separately, users lose up to 0.5 token (in accordance with the fixed-point scale) compared to the ideal floor(es/2 + es * vestedTime / (2 * vestingLinearPeriod)).

The bias is small but systematic: every request with non-zero fractional parts suffers an extra rounding loss during the linear vesting phase, so users consistently receive slightly less than intended.

Recommendation

Compute the amount with a single division to minimize truncation:

```
return (_vestingRequest.esOrderAmount * (vestingLinearPeriod + vestedTime)) / (2 *
vestingLinearPeriod);
```

Resolution

Orderly Team: The issue was resolved in commit a66632a.

I-01 | Misleading Comment

Category	Severity	Location	Status
Informational	● Info	LedgerBaseLegacy.sol: 32	Acknowledged

Description

The comment above the `feeRateOfFund` mapping reads, “fee rate of each strategy fund by vault ID.” However, this rate is actually stored by strategy provider ID, not vault ID.

Recommendation

Update the comment.

Resolution

Orderly Team: The issue was resolved in commit 1e47f60.

I-02 | Rebalance Slot Reuse Locks Funds

Category	Severity	Location	Status
Unexpected Behavior	● Info	VaultManager.sol	Acknowledged

Description

The VaultManager contract keeps only 100 rebalance slots via rebalanceStatus[rebalanceId % MAX_REBALACE_SLOT]. When executeRebalanceBurn records a new rebalance, it overwrites the slot unconditionally:

```
rebalanceStatus[data.rebalanceId % MAX_REBALACE_SLOT] =
RebalanceTypes.RebalanceStatus({
rebalanceId: data.rebalanceId,
burnStatus: Pending,
mintStatus: None
});
```

The following check only fires when the same rebalanceId is seen again:

```
RebalanceTypes.RebalanceStatus storage status = rebalanceStatus[data.rebalanceId %
MAX_REBALACE_SLOT];
if (status.rebalanceId = data.rebalanceId) {
if (status.burnStatus = Pending) revert RebalanceStillPending();
else if (status.burnStatus = Succ) revert RebalanceAlreadySucc();
}
```

it does nothing when a different ID collides modulo 100. As soon as 100 rebalances are outstanding, the 101st request overwrites the slot for the oldest in-flight transfer even if the burn or mint hasn't completed.

Later, when the remote chain sends the acknowledgement for that old transfer, rebalanceBurnFinish / rebalanceMintFinish look up the slot, see a different rebalanceId, and revert with RebalanceIdNotMatch.

The pending entry can never be cleared, so tokenBurnFrozenBalanceOnchain or tokenBalanceOnchain remains locked forever and the rebalance is effectively bricked.

Recommendation

Track outstanding operations with unique identifiers rather than a fixed-size modulo array. This guarantees that pending rebalances cannot be displaced before their callbacks arrive.

Resolution

Orderly Team: Acknowledged.

I-03 | Delegate Swap Signature Replayable

Category	Severity	Location	Status
Signatures	● Info	DelegateSwapSignature.sol	Acknowledged

Description

DelegateSwapSignature.validateDelegateSwapSignature hashes only abi.encode(data.tradeId, block.chainid, data.inTokenHash, data.inTokenAmount, data.to, data.value, data.swapCalldata) and runs it through ECDSA.toEthSignedMessageHash.

The contract address (or any vault-unique data) is missing, and the caller-supplied data.chainId is ignored, so the signed digest is identical on every vault that shares the same swapSigner.

Each vault tracks its own _submittedSwapSet; when vault A executes a swap, it records tradeId so the request cannot be replayed on that contract. But vault B's _submittedSwapSet is empty.

If the swap operator submits the exact same (tradeId, payload, signature) to vault B, _verifySwapSignature recomputes the same hash, recovers the same signer and accepts the swap because tradeId has not yet been seen on B.

The swap executes again, draining vault B's tokens even though the signer only meant to approve it for vault A.

Recommendation

Bind the signature to the specific vault instance—e.g., include address(this) in the abi.encode payload, or move to an EIP-712 domain that embeds the verifying contract address—so a signature approved for one vault cannot be replayed on another.

Resolution

Orderly Team: Acknowledged.

I-04 | Implementation Setters Don't Validate Code

Category	Severity	Location	Status
Best Practices	● Info	OmnichainLedgerV2.sol: 142-171	Resolved

Description

The implMerkleDistributor, implVesting, and implStakingValorRevenue addresses can be set to EOAs or non-contract addresses, as the setter functions do not verify that the provided addresses contain code.

Similarly, occAdapter can also be set to an EOA. Although these are admin-only functions, it is recommended to include code size checks as a best practice.

Recommendation

Check that the provided addresses have a non-zero code size in these functions.

Resolution

Orderly Team: The issue was resolved in commit 5e8b6b0, 334ad98.

I-05 | Missing Events For Timeline Changes

Category	Severity	Location	Status
Events	● Info	ValorImpl.sol: 85-89	Resolved

Description

Changing `valorEmissionStartTimestamp` and `valorSwitchTimestamp` materially affects emission and redemption logic, yet no events are emitted.

This impairs on-chain observability and can cause off-chain indexers or UIs to miss these state changes.

Recommendation

Emit events on changes and include old and new values.

Resolution

Orderly Team: The issue was resolved in commit bedcb99.

I-06 | Unused Role In Ledger

Category	Severity	Location	Status
Superfluous Code	● Info	Ledger.sol: 46	Resolved

Description

The SYMBOL_MANAGER_ROLE and BROKER_MANAGER_ROLE are introduced in the Ledger contract.

While the BROKER_MANAGER_ROLE is utilized in the setBrokerFromLedger function, the SYMBOL_MANAGER_ROLE is not used anywhere in the contract and can be removed.

Recommendation

Remove the unused role.

Resolution

Orderly Team: The issue was resolved in commit 3bc472c.

I-07 | HWM Decreases When Minting Discounted Shares

Category	Severity	Location	Status
Unexpected Behavior	● Info	ProtocolVaultLedger.sol	Acknowledged

Description

`ProtocolVaultLedger._calculateHWM` derives the next period's high-water mark by averaging the existing HWM with the current share price of newly issued shares.

The numerator is `totalShares * Math.max(hwm, sharePriceAfterFee) + newIssueShares * sharePriceAfterFee` and the denominator sums the two share counts.

Whenever the newly issued shares are priced below the previous HWM, that weighted average falls strictly below the old HWM.

For example, 100 legacy shares at HWM = 1 combined with 50 new shares at 0.8 yields a blended HWM of 140/150 = 0.93.

Because `settleMainAndStrategyFunds` assigns this value back to `strategyFundToken.hwm`, the protocol can charge performance fees even though existing LPs have not recovered their losses.

This violates the monotonic "high-water" guarantee and lets the strategy accrue fees prematurely.

Recommendation

After computing the blended value, clamp it so that the returned HWM is never lower than the previous one: e.g., `return Math.max(hwm, blendedHwm);`.

Alternatively, maintain separate baselines for new capital instead of blending discounted shares into the global high-water mark.

Resolution

Orderly Team: Acknowledged.

I-08 | VaultType Mismatch Between Repos

Category	Severity	Location	Status
Unexpected Behavior	● Info	LedgerImplD.sol: 106-118	Acknowledged

Description

The strategy-vault repository now supports COMMUNITY vault types. However, in the contract-evm repository, only the PROTOCOL and CEFFU types are supported.

All new COMMUNITY vaults are treated as PROTOCOL type from the contract-evm perspective.

Recommendation

Update the contract-evm repository to also support the COMMUNITY type. If this behavior is expected and all COMMUNITY vaults are indeed PROTOCOL vaults, ensure this is clearly documented.

Resolution

Orderly Team: Acknowledged.

I-09 | Redundant setAllowedStrategyProvider Function

Category	Severity	Location	Status
Superfluous Code	● Info	ProtocolVaultLedger.sol: 304	Acknowledged

Description

The `isAllowedStrategyProvider` mapping has been removed, but the `setAllowedStrategyProvider` function remains in the `ProtocolVaultLedger` contract.

This function accepts a `bool knob` parameter but does nothing other than emitting an event, which could be misleading.

Recommendation

Remove the redundant function.

Resolution

Orderly Team: Acknowledged.

Remediation Findings & Resolutions

ID	Title	Category	Severity	Status
M-01	halfUp16_8_i256 Function Misrounding	Rounding	● Medium	Resolved
L-01	Redundant Check In RedeemValor2 Flow	Superfluous Code	● Low	Resolved
L-02	Raw Approve Breaks Non-Standard Tokens	Best Practices	● Low	Pending
L-03	getData Blocks Final Dictionary Slot	Unexpected Behavior	● Low	Pending
L-04	Missing onlyOperator In UpdateUnclaimed	Access Control	● Low	Pending
I-01	Users May Pay More ccFee	Logical Error	● Info	Pending
I-02	Unused IAccessControl Import	Best Practices	● Info	Pending
I-03	Duplicate VaultEnum Definitions	Warning	● Info	Pending
I-04	Ledger Carries Unused Helper Imports	Best Practices	● Info	Pending
I-05	Dead VaultEnum.UserVault Entry	Best Practices	● Info	Pending
I-06	Unused Base Margin Setters	Warning	● Info	Pending
I-07	Unused Constants In OmnichainLedgerV2	Best Practices	● Info	Pending
I-08	VaultFactory Imports Unused ERC1967Proxy	Best Practices	● Info	Pending

Remediation Findings & Resolutions

ID	Title	Category	Severity	Status
I-09	TODO Comment Can Be Removed	Informational	● Info	Pending
I-10	StakingValorRevenueImpl Fuzzing Suite	Warning	● Info	Pending

M-01 | halfUp16_8_i256 Function Misrounding

Category	Severity	Location	Status
Rounding	● Medium	AccountTypePositionHelper.sol	Resolved

Description PoC

AccountTypePositionHelper.halfUp16_8_i256 uses if (quotient > 0) when deciding whether to increment its integer division result.

With a negative dividend whose magnitude is at least half the divisor, Solidity division produces quotient = 0 and remainder < 0; the > 0 guard increments that zero to +1. The adjacent int128 implementation correctly checks quotient > 0, then handles the zero case by inspecting the dividend sign.

A direct example is halfUp16_8_i256(-60, 100), which returns +1 instead of the required -1 even though the docstring promises half-up rounding (-1.6 -> -2 in the comment above). This helper feeds halfUp24_8_i256(-qty * price * currentHolding, qty) when a trader closes almost their entire position.

The incorrect +1 injects a small positive term into openingCost, flipping what should remain a negative long-cost into a positive value. The next line negates that number and passes it into halfDown16_8(...).toUint128(), so the cast reverts with SafeCastUnderflow.

Every trade, ADL transfer and liquidation calls calAverageEntryPrice first, therefore any trader can lock their own account (and any forced unwinds against it) by closing down to a 1e8 remainder where |openingCostOld * qty| crosses the half-divisor threshold.

Because of this affected accounts can no longer trade or be liquidated, leaving the protocol with unserviceable bad debt and service downtime for that user.

Recommendation

Match the int128 logic: change the first branch in halfUp16_8_i256 to if (quotient > 0) and keep the explicit else if (quotient < 0) plus the zero-quotient dividend-sign case so negative dividends round to -1.

Resolution

Orderly Team: The issue was resolved in commit afda2fb.

L-01 | Redundant Check In RedeemValor2 Flow

Category	Severity	Location	Status
Superfluous Code	● Low	RevenueImpl.sol: 180	Resolved

Description

The valor switch status is checked in the `OmnichainLedgerV2` contract when the payload is `RedeemValor2`.

It is then checked again inside the `_redeemValor2` internal function during the `delegatecall` to the implementation, which is redundant.

Recommendation

Remove one of the checks.

Resolution

Orderly Team: The issue was resolved in commit 03d50f5.

L-02 | Raw Approve Breaks Non-Standard Tokens

Category	Severity	Location	Status
Best Practices	● Low	OCCManager.sol	Pending

Description

OCCManager.vaultSendToLedger increases the OFT allowance via IERC20(erc20TokenAddr).approve(address(orderTokenOft), message.tokenAmount); even though the file already imports SafeERC20.

The approval function of some stablecoins like USDT do not return a boolean reverting whenever used with the IERC20 interface as the IERC20 interface always expects a boolean as a return.

When approvalRequired() is true the bridge can therefore get stuck: the allowance never updates, transfers revert on non-standard tokens and the function provides no revert reason because the return value is unchecked.

This makes vault -> ledger bridging brittle and incompatible with some specific tokens.

Recommendation

Replace the bare approve with SafeERC20 helpers that handle zero-first semantics and return-value checking, e.g. ERC20(token).safeApprove(orderTokenOft, 0); ERC20(token).safeApprove(orderTokenOft, message.tokenAmount); or safeIncreaseAllowance when appropriate.

This keeps allowances consistent for non-standard ERC20s and surfaces failures through SafeERC20's unified revert path.

Resolution

Orderly Team: Pending.

L-03 | getData Blocks Final Dictionary Slot

Category	Severity	Location	Status
Unexpected Behavior	● Low	DecompressorExtension.sol	Pending

Description

DecompressorExtension applies validDictAccess(end) to the exclusive upper bound of getData. That modifier requires end < MAX_DICT_LEN, yet the loop reads [begin, end):

```
for (uint256 i = begin; i < end; i++) {
res[i - begin] = _dict[i];
}
```

To fetch slot MAX_DICT_LEN - 1, callers must pass end = MAX_DICT_LEN, but the modifier reverts before the loop runs.

The last dictionary entry is therefore unreachable via this accessor, shrinking usable storage and breaking decompressions that rely on the final slot.

OperatorManagerZip inherits this contract but never calls getData, so the impact is limited to external dictionary readers rather than the existing operator flows.

Recommendation

Validate end locally with require(end < MAX_DICT_LEN end > begin) (or early-return when begin = end) instead of reusing validDictAccess so the exclusive bound may equal MAX_DICT_LEN without opening the reserved slots.

Resolution

Orderly Team: Pending.

L-04 | Missing onlyOperator In UpdateUnclaimed

Category	Severity	Location	Status
Access Control	● Low	ProtocolVaultLedger.sol: 246	Pending

Description

The onlyOperator modifier in the updateUnclaimed function of the ProtocolVaultLedger contract was removed after recent fixes.

However, this function should only be called by the operator, along with other functions, in a specific order.

Although it still requires the engine signature, retaining this modifier is important to prevent unintended or unauthorized use.

Recommendation

Add onlyOperator modifier.

Resolution

Orderly Team: Pending.

I-01 | Users May Pay More ccFee

Category	Severity	Location	Status
Logical Error	● Info	LedgerCoreImpl.sol: 433C6-435C19	Pending

Description

The ccFee can now be provided during the updateUnclaimed flow, and this provided ccFee is the per user amount that will be paid by each user. The sum of total fee paid by users should ideally be equal to the actual crosschain fee required.

However, there is no cross-check between the provided ccFee at the time of engine signature and the actual required fee at the time of message execution.

Even if the engine performs a quote and provides the correct fee, it is highly likely that the required fee at execution time will differ.

To ensure successful execution, the engine must include a buffer and provide more fee than required.

Since the cross-chain message in this flow is sent without a fee refund mechanism, any excess amount will not be refunded.

Recommendation

Be aware that discrepancies may occur due to the time gap between signature and execution, and document this behavior for users since they might have to pay slightly more than required.

Resolution

Orderly Team: Pending.

I-02 | Unused IAccessControl Import

Category	Severity	Location	Status
Best Practices	● Info	AccessControlRevised.sol	Pending

Description

`AccessControlRevised.sol` imports `IAccessControl` but the contract never references that interface anywhere in the file, so the compiler strips it while still emitting metadata about it:

```
import {IAccessControl} from "@openzeppelin/contracts/access/IAccessControl.sol";
```

Leaving unused imports behind clutters the source, confuses reviewers about dependencies, and slightly inflates bytecode metadata for no benefit. Keeping the file lean avoids that noise.

Recommendation

Remove the `IAccessControl` import entirely so the file only declares dependencies that are actually used.

Resolution

Orderly Team: Pending.

I-03 | Duplicate VaultEnum Definitions

Category	Severity	Location	Status
Warning	● Info	VaultTypes.sol; EventTypes.sol	Pending

Description

Both EventTypes.sol and VaultTypes.sol declare their own VaultEnum, yet the two enums drive different halves of the withdraw pipeline: ledger code checks EventTypes.VaultEnum (LedgerImplD.sol) while the on-chain vault consumes VaultTypes.VaultEnum (vaultSide/Vault.sol).

Any future reordering or addition must be mirrored in both files manually; a mismatch would silently corrupt cross-chain withdraw routing because signed payloads would deserialize to the wrong enum branch without compiler errors.

Recommendation

Keep a single definition (e.g., declare the enum in VaultTypes.sol and import/use VaultTypes.VaultEnum everywhere, or move it into a shared base) so both ledger signatures and vault execution reference identical ordinals.

Resolution

Orderly Team: Pending.

I-04 | Ledger Carries Unused Helper Imports

Category	Severity	Location	Status
Best Practices	● Info	Ledger.sol	Pending

Description

`Ledger.sol` imports `Utils.sol` and `Signature.sol` and applies `using AccountTypePositionHelper for AccountTypes.PerpPosition;`, yet the contract never calls `Utils.*`, `Signature.*`, or any extension methods defined in `AccountTypePositionHelper`.

These declarations therefore drag in unused dependencies.

Recommendation

Drop the unused imports and `using` directive so the `Ledger` contract only depends on the libraries it truly needs.

Resolution

Orderly Team: Pending.

I-05 | Dead VaultEnum.UserVault Entry

Category	Severity	Location	Status
Best Practices	● Info	VaultTypes.sol; EventTypes.sol	Pending

Description

VaultEnum.UserVault is declared in both VaultTypes.sol and EventTypes.sol, yet no contract ever checks that value—withdraw flows in vaultSide/Vault.sol and LedgerImplD.sol only handle ProtocolVault and Ceffu, falling through to revert NotImplemented() otherwise.

As a result, any signature that encodes UserVault will always revert, and the enum member serves purely as dead code that suggests a non-existent path.

Recommendation

Remove the UserVault member (and its mirror in EventTypes) unless the user path is fully implemented end-to-end; otherwise callers may believe a vault type exists when it cannot execute.

Resolution

Orderly Team: Pending.

I-06 | Unused Base Margin Setters

Category	Severity	Location	Status
Warning	● Info	MarketTypeHelper.sol	Pending

Description

MarketTypeHelper defines setBaseMaintenanceMargin, setBaseInitialMargin, and setLiquidationFeeMax, yet there are no call sites for any of the three functions elsewhere in the repo.

Unlike the other setters in the helper, these add no behavior—just extra bytecode and cognitive overhead—so the library exports dead helpers that are never run.

Recommendation

Remove the unused setters (or actually integrate them wherever base maintenance/initial margins and max liquidation fees should be updated) so the helper only exposes live functionality.

Resolution

Orderly Team: Pending.

I-07 | Unused Constants In OmnichainLedgerV2

Category	Severity	Location	Status
Best Practices	● Info	OmnichainLedgerV2.sol	Pending

Description

`OmnichainLedgerV2` imports `DEFAULT_UNSTAKE_LOCK_PERIOD`, `DEFAULT_BATCH_DURATION`, `VESTING_LOCK_PERIOD` and `VESTING_LINEAR_PERIOD`, but the contract never references those constants anywhere else in the file (they are only used inside tests such as `contracts/test/LedgerTestV2.sol`).

Keeping these unused imports triggers compiler warnings and marginally inflates the artifact without adding functionality.

Recommendation

Remove the unused constant imports (or actually apply them during initialization) so the production contract only depends on values it consumes.

Resolution

Orderly Team: Pending.

I-08 | VaultFactory Imports Unused ERC1967Proxy

Category	Severity	Location	Status
Best Practices	● Info	VaultFactory.sol	Pending

Description

VaultFactory imports ERC1967Proxy, yet the factory never instantiates proxies. It deploys bytecode exclusively through CREATE3.deployDeterministic. Leaving the proxy import in place adds an unused dependency.

Recommendation

Remove the ERC1967Proxy import so the factory's dependencies reflect the actual implementation (CREATE3 + Ownable2Step) and the compiled artifact stays minimal.

Resolution

Orderly Team: Pending.

I-09 | TODO Comment Can Be Removed

Category	Severity	Location	Status
Informational	● Info	LedgerImplD.sol: 209	Pending

Description

The `// TODO: check escrowBalance` comment at line 209 of the `LedgerImpD` contract can be removed, as the function now includes an escrow balance check after the updates.

Recommendation

Remove the comment.

Resolution

Orderly Team: Pending.

I-10 | StakingValorRevenueImpl Fuzzing Suite

Category	Severity	Location	Status
Warning	● Info	StakingValorRevenueImpl.sol	Pending

Description

We designed test/forge/StakingValorRevenueFuzz.t.sol to act as a state-machine fuzz harness around StakingValorRevenueImplTest.

The harness bootstraps the full staking + Valor + revenue stack with production-like parameters, exposes private accounting getters via StakingValorRevenueFuzzHarness and then drives 13 pseudo-random operations (stakes, immediate/delayed unstakes, Valor1 and Valor2 redemptions, USDC/esORDER claims, revenue updates, batch preparations, operator sweeps, and esORDER unstake) across three chain IDs.

After each of the 20–60 actions per run, the suite enforces conservation invariants including total stake equality, pending-unstake mirrors, treasury coverage, user reward fairness, batch totals per chain, emission caps, and Valor1 freeze semantics post-switch.

We executed the suite for more than one million iterations (combining multiple seeds and high FOUNDRY_FUZZ_RUNS configurations) and observed zero invariants reverting or unexpected harness reverts.

This demonstrates strong confidence that cross-chain accounting, revenue batching and the Valor1 -> Valor2 transition logic are internally consistent under randomized stress.

Recommendation

Adopt this fuzz harness in CI with a mid-sized iteration budget (e.g., 25k–50k runs) so regressions in accounting logic are caught automatically, and continue scheduling periodic > 1M-run campaigns before major releases to preserve the observed invariant stability.

Resolution

Orderly Team: Pending.

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

Founded in 2022 by DeFi experts, Guardian is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian 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