GUARDIAN

Orderly

Updates Review

Security Assessment

November 11th, 2025

Summary

Audit Firm Guardian

Prepared By Zdravko Hristov, Osman Ozdemir, Alex Lazar

Client Firm Orderly

Final Report Date November 11, 2025

Audit Summary

Orderly engaged Guardian to review the security of their updates to Orderly Solana Vault, Sol-CC, and Contract-EVM. From the 25th of August to the 3rd of September, a team of 3 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Given the number of High and Critical issues detected as well as additional code changes made after the main review, Guardian assigns a Confidence Ranking of 2 to the protocol. Guardian recommends that an independent security review of the protocol at a finalized frozen commit is conducted before deployment. Guardian strongly advises that the protocol undergo a full follow-up audit at a finalized and fully remediated commit before any mainnet deployment. 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)	Solana Vault Main Review Commit: 575f56341ded8aa004e68020b01c21c89c4eeda9 Solana Vault Remediation Review Commit: 4582ca7c7df7842bc0bf5805e6ff16930d2ddbcc Sol-CC Main Review Commit: 2106b8a45f78d34c8da4036fccbf7f74be657e36 Sol-CC Remediation Review Commit: e565fcd379b37475cfa7268ca5612ba6f9dbcde8 Contract-EVM Main Review Commit: 2adf4141392c0547ce37361f70eb765160e26421 Contract-EVM Remediation Review Commit: 51a7a28af361865fb2fa7b10833cc9ea5e4c81db

Audit Summary

Delivery Date	November 11, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	1	0	0	0	0	1
● High	6	0	0	2	0	4
● Medium	5	0	0	4	0	1
● Low	22	0	0	13	2	7
● Info	19	0	0	8	0	11

Audit Scope & Methodology

Scope and details:

Solana Vault Repo:
https://gitlab.com/orderlynetwork/orderly-v2/solana-vault/

Solana Vault Commit:
575f56341ded8aa004e68020b01c21c89c4eeda9

Sol-CC Repo:
https://gitlab.com/orderlynetwork/orderly-v2/sol-cc

Sol-CC Commit:
2106b8a45f78d34c8da4036fccbf7f74be657e36

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
C-01	Any Allowed Token Can Be Used	Validation	● Critical	Resolved
H-01	LZ Messaging Channel Can Be Blocked	DoS	● High	Resolved
H-02	EVM/Solana Fee Collectors Must Be Separate	Logical Error	● High	Acknowledged
H-03	Insufficient Manager Role Validation	Access Control	● High	Resolved
H-04	Frozen ATAs Can't Ever Recover Funds	Logical Error	● High	Acknowledged
M-01	Can DOS Oapp_lz_receive.rs	DoS	● Medium	Resolved
M-02	Incorrect EIP-712 Typehash	Validation	● Medium	Acknowledged
M-03	Ordered Execution Option Is Not Added	Unexpected Behavior	● Medium	Acknowledged
M-04	rebalanceBurn() Should Not Revert	Logical Error	● Medium	Acknowledged
L-01	Only Onchain Success Changes Vault Balances	Warning	● Low	Resolved
L-02	Sol_vault Must Be Rent Exempt	DoS	● Low	Acknowledged
L-03	Internal Transfers Warning	Warning	● Low	Acknowledged
L-04	Deposit Fee Is Disabled In Paused State	Unexpected Behavior	● Low	Partially Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-05	Vault ATA Must Be Initialized	DoS	● Low	Acknowledged
L-06	Native Token Swaps Will Always Fail	Unexpected Behavior	● Low	Acknowledged
L-07	Lack Of Event Emissions	Events	● Low	Acknowledged
L-08	Avoiding Precision Loss For Withdrawals	Rounding	● Low	Acknowledged
L-09	Discrepancy About SYMBOL_MANAGER_ROLE	Access Control	● Low	Partially Resolved
L-10	Warning Related To delegateSwap	Warning	● Low	Acknowledged
L-11	0 Deposits Are Allowed	Validation	● Low	Resolved
L-12	Handling Ordered Delivery Transition	Configuration	● Low	Acknowledged
L-13	Duplicate lastEngineEventId Assignment	Logical Error	● Low	Resolved
L-14	Incorrect Receiver Check	Validation	● Low	Resolved
L-15	No ETH Refund When depositFee Is Disabled	Unexpected Behavior	● Low	Acknowledged
L-16	SolConnector Balance Cannot Be Withdrawn	Logical Error	● Low	Resolved
L-17	Mixed Events In LedgerImplD	Events	● Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-18	Weaponizing Swaps To Drain The Ledger	Logical Error	● Low	Acknowledged
I-01	Consider Having Rescue Functionality	Informational	● Info	Acknowledged
I-02	Include System_program During Invoke	Informational	● Info	Resolved
I-03	Incomplete bizType Comment Description	Informational	● Info	Resolved
I-04	Settlement Balance Check Is Commented	Warning	● Info	Acknowledged
I-05	OApps Don't Call Skip(), Burn() Or Clear()	Best Practices	● Info	Acknowledged
I-06	Hardcoded Roles	Best Practices	● Info	Acknowledged
I-07	Deposit Nonce Can Be Changed	Configuration	● Info	Resolved
I-08	Outdated Comment	Informational	● Info	Resolved
I-09	SwapSignature Library Is Not Used	Informational	● Info	Acknowledged
I-10	Can Set Arbitrary Hashes For Roles	Best Practices	● Info	Resolved
I-11	Executor Needs Funds To Initialize ATA	Informational	● Info	Resolved
I-12	Withdrawals May Drain The Connector	Informational	● Info	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
I-13	Technically Possible Underflow	Best Practices	● Info	Resolved
I-14	ATA Check Is Not Used For The Logical Forks	Superfluous Code	● Info	Resolved
I-15	SOL Mint_account Should Stay 0	Informational	● Info	Resolved

C-01 | Any Allowed Token Can Be Used

Category	Severity	Location	Status
Validation	● Critical	deposit_sol.rs: 71-76	Resolved

Description PoC

The deposit_sol file includes a check to ensure that the token account derived from the provided deposit_params.token_hash is an allowed token.

However, it does not verify that the hash specifically corresponds to SOL. As a result, the hash of any other allowed token can be provided when depositing SOL.

When a user invokes deposit_sol with a USDC hash, 1e9 native SOL will be transferred from the user to the vault on Solana, while 1000e6 USDC will be credited to the user on the ledger chain, due to SOL having 9 decimals and USDC having 6 decimals.

This results in an immediate ~5x profit for the user at current market rates.

Recommendation

Only allow the SOL/wSOL hash to be provided in deposit_sol.

Resolution

Orderly Team: The issue was resolved in commit 2beba85.

H-01 | LZ Messaging Channel Can Be Blocked

Category	Severity	Location	Status
DoS	● High	Vault.sol: 395	Resolved

Description

The Vault contract now supports native deposits and withdrawals. During a native withdrawal, the receiver address is called using `payable(receiver).sendValue(amount)`.

The receiver address can intentionally revert to block the messaging channel or consume the entire gas provided by the LZ endpoint.

The `CrossChainRelay` contract uses `_blockingLzReceive`, meaning that subsequent messages cannot be executed until the previous message has been successfully delivered.

When the receiver intentionally reverts during `_ethWithdraw`, the `lzReceive` call reverts, and the message is stored in `storedPayload` on the `LayerZero` endpoint.

As a result, the messaging channel is blocked until `forceResumeReceive` is called.

Recommendation

Wrap the `_ethWithdraw` in a try/catch block and emit an event to allow manual resolution later if the receiver cannot accept native tokens.

Resolution

Orderly Team: The issue was resolved in commit c57806a.

H-02 | EVM/Solana Fee Collectors Must Be Separate

Category	Severity	Location	Status
Logical Error	● High	LedgerImplC.sol: 131	Acknowledged

Description

Both EVM and Solana withdrawals from the Ledger contract attribute any fees earned to the same fee collector - the one that was collecting all the EVM fees until now.

This approach worked until now since EVM accounts can withdraw from every other supported EVM chain, but only accounts that deposited on Solana can withdraw on Solana.

Look at the following example:

• Alice deposits 1000 USDC on Solana

• Bob deposits 1000 USDC on Arbitrum

• Alice withdraws 1000 (900 for her and 100 for fee collector)

• The fee collector claims them on Arbitrum

• Bob cannot access the Solana funds which means he loses them forever, can only withdraw 900

Furthermore, any SOL fees will be lost since they cannot be withdrawn on an EVM chain.

Recommendation

Add a new fee collector which collects only Solana fees in the FeeManager contract.

Resolution

Orderly Team: Acknowledged.

H-03 | Insufficient Manager Role Validation

Category	Severity	Location	Status
Access Control	● High	set_withdraw_broker.rs,22	Resolved

Description PoC

The Solana Vault implements a role based approach to let certain accounts change the configuration. The vault owner can grant and revoke roles by executing the set_manager_role instruction.

A manager_role PDA is derived for the given role and user account. This manager_role stores a few fields, the most important of which is the allowed one - it shows whether the given user account has the role.

When executing state changing instructions, that PDA is derived and the allowed flag is checked to be true. For example, in the set_broker instruction

```
#[account(
seeds = [ACCESS_CONTROL_SEED, params.broker_manager_role.as_ref(),
broker_manager.key().as_ref()],
bump = manager_role.bump,
constraint = manager_role.allowed = true @VaultError:ManagerRoleNotAllowed,
)]
```

There is a problem with the derivation. One of the used seeds - broker_manager_role - is read directly from params. This allows the caller to supply any other role that they have and still execute the current action.

For example, if they don't have the broker_manager_role, but instead the token_manager_role, they can use the token_manager PDA to bypass the check and still change the broker.

Recommendation

Consider hardcoding the role_hash instead of reading it from the user parameters.

Resolution

Orderly Team: The issue was resolved in commit 196e1f2.

H-04 | Frozen ATAs Can't Ever Recover Funds

Category	Severity	Location	Status
Logical Error	● High	packages/solana/contracts/programs/solana-vault /src/instructions/oapp_instr/oapp_lz_receive.rs	Acknowledged

Description

The very first thing that happens in `OAppLzReceive:apply()` is a CPI (cross-program invocation) to the `OAppLzReceive` program to clear the message. Unlike in the EVM LZ implementation, the LZ user is expected to clear the message. If the transaction fails "loudly" later, all changes get reverted.

The problem is if the transaction fails silently, which can happen in the if `ata_account.is_frozen()` {} code block. At the point when this line is executed, the program already validated the LZ message and the withdrawal as a whole. It is effectively ready to transfer the funds.

If the receiver's ATA is frozen, neither a transfer to the user or an escrow occurs, nor does the transaction revert. Instead, the code only emits a `FrozenWithdrawn` event.

There is no instruction available for owners or managers to manually withdraw funds from the vault to recover them for the receiver if the frozen ATA becomes unfrozen at a later date.

Recommendation

Consider leaving the balance on the destination chain if the receiver is frozen, and add a function to allow the owner/manager to manually withdraw it in case the receiver later becomes unfrozen.

Alternatively, consider reverting the transaction and not clearing the LZ message to allow replayability.

However, this may introduce DoS scenarios when message delivery is ordered, requiring manual resolution by the OApp manager/owner.

Resolution

Orderly Team: Acknowledged.

M-01 | Can DOS Oapp_lz_receive.rs

Category	Severity	Location	Status
DoS	● Medium	oapp_lz_receive.rs: 146	Resolved

Description

The `oapp_lz_receive.rs` expects nonces to be ordered if the `order_delivery` is true. (`vault_authority.check_nonce(params.nonce)` on line 70). The function transfers native SOL to an arbitrary address (receiver).

However, native SOL transfers can fail for a number of reasons like rent exemption, the receiver being executable, or a write-demotion happening (Reference article). These are things that happen in practice (Jito had a bug reported for this on their bug bounty, for example).

If the SOL transfer fails for any of the above reasons while `order_delivery` is set to true, it will not only revert the current transfer but also block any future messages because the nonce does not increment.

This can be easily fixed by setting `order_delivery` to false in `set_order_delivery.rs`. However, if it occurs, users may be blocked from receiving their funds until the admins disable `order_delivery` to allow the problematic nonce to pass.

It is also worth noting that an attacker could, at any time while `order_delivery` is true, exploit this to DoS the program by initiating a withdrawal of less than the rent-exempt amount or by setting the receiver address to one of the reserved accounts.

Recommendation

It would be safer (albeit more complex) to allow the user to withdraw funds and handle the transfer in a separate function, rather than directly transferring Sol to the user within `oapp_lz_receive.rs`

Resolution

Orderly Team: The issue was resolved in commit 4582ca7.

M-02 | Incorrect EIP-712 Typehash

Category	Severity	Location	Status
Validation	● Medium	SwapSignature.sol: 13	Acknowledged

Description

The DELEGATE_SWAP_TYPEHASH defined in the SwapSignature contract is keccak256("DelegateSwap(uint256 swapNonce,uint256 chainId,bytes32 inTokenHash,uint256 inTokenAmount,address to,uint256 value,bytes swapCalldata)").

The first parameter here is uint256 swapNonce. However, the first parameter used in the validateSwapSignature function is bytes32 tradeId from the DelegateSwap struct.

This discrepancy causes the signatures to be non-compliant with EIP-712, or it would cause the signature check to fail if the signer correctly signs according to the EIP-712 standard.

Recommendation

Update the DELEGATE_SWAP_TYPEHASH to match the actual struct used in validation.

Resolution

Orderly Team: Acknowledged.

M-03 | Ordered Execution Option Is Not Added

Category	Severity	Location	Status
Unexpected Behavior	● Medium	SolConnector.sol: 158-160	Acknowledged

Description

Like already described in this issue of the Sherlock report, `SolConnector.withdraw()` doesn't call `addExecutorOrderedExecutionOption()` when it builds the message options.

The newly added `withdraw2ContractV2()` function doesn't use this option as well. This means there is no guarantee that the messages will be executed in order.

If they aren't and the Solana vault has turned it's ordered delivery feature on, the messages will fail and will have to be resubmitted manually once the previous ones have been executed.

Recommendation

1. Add new state variable that tracks the ordered delivery status of the Solana Vault.

2. If this state variable is `true`, use `addExecutorOrderedExecutionOption` when sending the crosschain message.

Resolution

Orderly Team: Acknowledged.

M-04 | rebalanceBurn() Should Not Revert

Category	Severity	Location	Status
Logical Error	● Medium	Vault.sol: 562	Acknowledged

Description

A new check has been added to `Vault.rebalanceBurn()`

```
if (_rebalanceEnableTokenSet.contains(data.tokenHash)) revert NotRebalanceEnableToken();
```

This will result in failed attempts to burn disabled tokens, which:

• blocks the communication channel because the crosschain manager defaults to the blocking behavior

• causes the funds to be permanently frozen on the Orderly Network's Ledger because `rebalanceBurnFinish()` is not invoked, resulting in a loss of these funds

Recommendation

Instead of reverting, send back a failure `rebalanceBurnFinish()` message to the Ledger and stop the execution of the function.

Resolution

Orderly Team: Acknowledged.

L-01 | Only Onchain Success Changes Vault Balances

Category	Severity	Location	Status
Warning	● Low	LedgerImplD.sol: 120-123	Resolved

Description

During executeSwapResultUpload, the user's balances are always updated, but the chain ones - only if the result of the swap was onchain success.

This creates a risk of an accounting mismatch for offchain swaps because the Vault contract will hold different funds that what is actually recorded in the manager.

Recommendation

Consider if `offchain success` should also cause update of the `vaultManager` balance

Resolution

Orderly Team: Resolved.

L-02 | Sol_vault Must Be Rent Exempt

Category	Severity	Location	Status
DoS	● Low	deposit_sol.rs: 30-36	Acknowledged

Description PoC

Accounts on Solana must hold lamports value below their rent exemption value. The rent exemption value depends on the data stored. Since empty accounts still store metadata, there is a minimum amount of rent exemption that has to be paid. At the moment of writing that amount is 890880.

The sol_vault account in deposit_sol is a PDA with no data. Because of that, no initialization happened and no lamports were transferred to the account.

During deposits the sol_vault receives token_amount of lamports. If the first deposit's token_amount is less than the rent exemption value, the transaction will fail.

During withdrawals, lamports are withdrawn from the sol_vault. If it's not the full amount being withdrawn and the balance of the sol_vault drops below the rent exemption value, the withdrawal will fail, the channel will be blocked if ordered delivery is enabled and the user won't receive their funds.

Recommendation

To solve the issue, consider sending the minimum rent exemption value to the sol_vault when you deploy the app.

Resolution

Orderly Team: Acknowledged.

L-03 | Internal Transfers Warning

Category	Severity	Location	Status
Warning	● Low	Ledger.sol	Acknowledged

Description

The internal transfer functionality of the `Ledger`, for example `executeBalanceTransfer()` and `executeFeeDistribution()` should be executed with caution between EVM and Solana accounts because the liquidity is not shared, so transferring from one chain to another can result in insolvency.

Recommendation

The internal transfer functionality of the `Ledger`, for example `executeBalanceTransfer()` and `executeFeeDistribution()` should be executed with caution between EVM and Solana accounts because the liquidity is not shared, so transferring from one chain to another can result in insolvency.

Resolution

Orderly Team: Acknowledged.

L-04 | Deposit Fee Is Disabled In Paused State

Category	Severity	Location	Status
Unexpected Behavior	● Low	Vault.sol: 301,312	Partially Resolved

Description

The `enableDepositFee()` `onlyOwner` function and the `getDepositFee()` view functions have the `whenNotPaused` modifier.

This restricts the flexibility the owner of the Vault has during periods of emergencies when the contract is paused. It also causes failures for offchain integrators that call `getDepositFee()`.

Recommendation

Consider removing the modifier from these 2 functions.

Resolution

Orderly Team: The issue was resolved in commit f769643.

L-05 | Vault ATA Must Be Initialized

Category	Severity	Location	Status
DoS	● Low	oapp_lz_receive.rs: 76	Acknowledged

Description

When the lz_receive instruction is executed for withdrawal, vault_token_account is ATA with mint = token_mint. For SOL, the token_mint is WSOL.

If nobody deposited WSOL through the vault before that, the vault_token_account will most likely be uninitialized, which will lead to a failed withdrawal message that has to be retried manually and blocked pathway if ordered delivery is turned on.

Recommendation

Initialize the WSOL vault_token_account right after calling init_oapp().

Resolution

Orderly Team: Acknowledged.

L-06 | Native Token Swaps Will Always Fail

Category	Severity	Location	Status
Unexpected Behavior	● Low	Vault.sol	Acknowledged

Description

Vault.delegateSwap() swaps the desired vault token for another one and the result is being uploaded by calling Ledger.executeSwapResultUpload().

The Vault supports native tokens as well, but any swaps where a native token is the output token will be failing because the Vault doesn't have a payable receive() or fallback() function.

This causes DOS for such on chain swaps and is even more dangerous for offchain swaps, since it can cause accounting mismatch.

Recommendation

Add a payable receive() function to the Vault.

Resolution

Orderly Team: Acknowledged.

L-07 | Lack Of Event Emissions

Category	Severity	Location	Status
Events	● Low	set_ordered_delivery.rs; set_vault.rs; oapp_instr/	Acknowledged

Description

The set_ordered_delivery and set_vault instructions change important configuration of the Vault, but no events are emitted. The same is true for the instructions in the oapp_instr/ directory.

Recommendation

Consider adding event emissions for at least the two vault instructions.

Resolution

Orderly Team: Acknowledged.

L-08 | Avoiding Precision Loss For Withdrawals

Category	Severity	Location	Status
Rounding	● Low	Global	Acknowledged

Description

When withdrawals are uploaded on the Ledger side, the amounts used are in Ledger decimals and will be converted to destination decimals before the crosschain message is sent. For example, look at the SolConnector.

```
convertWithdrawDecimals(tokenHash, _withdrawData.tokenAmount),
convertWithdrawDecimals(tokenHash, _withdrawData.fee),
```

In case that the destination chain's decimals are lower than the ledger ones, both the `tokenAmount` and the fee will experience precision loss.

Whenever the `tokenAmount` loses precision, the withdrawing user receives less funds and whenever the fee loses precision, the available liquidity for the fee collector decreases. However, on the ledger sides these values are stored in ledger decimals which will lead to accounting mismatch overtime.

For example, if the fee collector accrues fees from several withdrawals, the final sum it tries to withdraw may be greater than what's available on that chain.

This problem is also present for the rest of the EVM chains, but for Solana it's more serious since a withdrawal failure may block future requests if ordered delivery is turned ON.

Recommendation

The most simple fix to avoid the precision loss would be to make the `BE` adjust the withdrawal amount depending on the desired destination chain.

For example, if ledger token has 3 decimals, but destination token has 2 decimals and the user wants to withdraw 123 tokens, the BE (and the FE) will instead give the user to withdraw only 12 tokens to avoid the precision loss. Same thing should be applied to the fee

Resolution

Orderly Team: Acknowledged.

L-09 | Discrepancy About SYMBOL_MANAGER_ROLE

Category	Severity	Location	Status
Access Control	● Low	Vault.sol: 187-199	Partially Resolved

Description

SYMBOL_MANAGER_ROLE has access to most functions related to deposit tokens and allowed tokens, such as setDepositLimit, setNativeTokenHash, setNativeTokenDepositLimit, and disableDepositToken.

However, the enableDepositToken, setRebalanceEnableToken, and changeTokenAddressAndAllow functions remain restricted to onlyOwner.

Recommendation

Consider whether this is the expected behavior. If not, change onlyOwner to onlyRoleOrOwner(SYMBOL_MANAGER_ROLE) for these functions as well.

Resolution

Orderly Team: The issue was resolved in commit e45a1bf.

L-10 | Warning Related To delegateSwap

Category	Severity	Location	Status
Warning	● Low	Vault.sol: 685	Acknowledged

Description

The `delegateSwap` function in the Vault contract allows the `swapOperator` to transfer the entire `ERC20` token balance and native balance of the contract to an arbitrary `to` address.

Additionally, the function is not payable, which means native swaps will always use the contract's own balance.

While the `swapOperator` is a trusted role, this issue serves to inform users about the capabilities of the role.

Recommendation

No fix is required if the `swapOperator` is expected to use the entire funds in the Vault contract.

Resolution

Orderly Team: Acknowledged.

L-11 | 0 Deposits Are Allowed

Category	Severity	Location	Status
Validation	● Low		Resolved

Description

The deposit and deposit_sol instructions don't enforce positive amount when depositing which allows anyone, including accounts with 0 balance, to invoke the instructions and send meaningless messages to the Ledger side that won't update any amounts, but will update the account's metadata, like lastDepositEventId and etc.

This is not possible on the EVM chains because of the following validation

```
if (data.tokenAmount = 0) revert ZeroDeposit();
```

Recommendation

Don't allow deposits with 0 amounts.

Resolution

Orderly Team: The issue was resolved in commit a81bc89.

L-12 | Handling Ordered Delivery Transition

Category	Severity	Location	Status
Configuration	● Low	Global	Acknowledged

Description

Both the Solana Vault and SolConnector are OApps that can toggle the ordered delivery feature. This is a guide to how to transition from unordered delivery to ordered delivery should be handled.

Let's look at an example where ordered delivery was turned ON until nonce 15. Then it was switched OFF and messages until nonce 20 have been verified, but not executed. Because the order is not enforced, nonce 20 can be executed before all the others. If the Orderly team decides to go back to ordered delivery, nonces 16-19 will become invalid, which will result in loss of funds for users since they cannot finish their deposit/withdrawal.

To solve this problem, it's best to first wait for all of the unexecuted messages to be delivered and only then switching to ordered delivery. However, there is no guarantee that new messages won't appear while waiting for the old ones to be executed, which will extend the waiting time. This can continue forever and lead to inability to go back to ordered delivery.

For this reason, there is a need of a way to pause source messages from being initiated
• The SolConnector can be paused for that goal, but that would not only prevent it from sending messages, but also from receiving, which may not be ideal. It's best to be able to pause receiving and sending separately.
• Technically deposits in the Solana Vault can be paused by setting allowed_token.allowed = false for each of the supported tokens, but you should add a global deposit flag that you can toggle for both deposit and deposit_sol.

Even if all of the steps above are taken, the inbound nonce on both chains is updated unconditionally whenever a message is received. Let's take a look at our example again and assume the five messages were executed while we are waiting in a paused state, but the last one had nonce = 15. Because inboundNonce is unconditionally updated to 15, when we switch back to ordered delivery, the app will expect nonce 16 to be executed, but the real one will be 21 and the Orderly team will need to manually update the inboundNonce.

NOTE: Other changes of configuration should be handled like that as well, for example changing token indexes.

Recommendation

1. Make the pause feature in SolConnector more granular
2. Add is_deposit_paused flag to SolanaVault and use it in deposit and deposit_sol.
3. Update the inboundNonce only if the new one is greater than the old one (issue exists in EVM and Solana contracts)
4. When transitioning from unordered to ordered delivery:
• pause the sending functionality of the other chain
• wait until all messages are executed
• unpause the sending functionality of the other chain

Resolution

Orderly Team: Acknowledged.

L-13 | Duplicate lastEngineEventId Assignment

Category	Severity	Location	Status
Logical Error	● Low	LedgerImplD.sol: 266	Resolved

Description

In the `executeWithdraw2ContractV2` function, `account.lastEngineEventId` is updated at line 225 for both the EVM and SOL chain types, and then updated again at line 266 within the `else if` block for the SOL chain type.

Recommendation

Remove the redundant assignment.

Resolution

Orderly Team: The issue was resolved in LedgerImplD.sol#L59.

L-14 | Incorrect Receiver Check

Category	Severity	Location	Status
Validation	● Low	LedgerImplD.sol: 176	Resolved

Description

In the `executeWithdraw2ContractV2` function, the zero-address check for the receiver is performed on `withdraw.sender` instead of `withdraw.receiver`.

Recommendation

Use `withdraw.receiver` in the check if the sender and receiver are not enforced to be the same at all times.

Resolution

Orderly Team: The issue was resolved in LedgerImplD.sol#L188.

L-15 | No ETH Refund When depositFee Is Disabled

Category	Severity	Location	Status
Unexpected Behavior	● Low	Vault.sol: 338-345,368-375	Acknowledged

Description

`Vault._deposit()` and `Vault._ethDeposit()` don't refund the excess `msg.value` sent if `depositFeeEnabled = false`.

While it's not expected for users to send additional funds in that case, such transaction may be executed because of a configuration change.

For example:

• User deposits when fee is enabled

• Owner disables the fee

• User's transaction is executed and they don't get refund

Recommendation

Consider refunding the user if deposit fee is not enabled.

Resolution

Orderly Team: Acknowledged.

L-16 | SolConnector Balance Cannot Be Withdrawn

Category	Severity	Location	Status
Logical Error	● Low	SolConnector.sol	Resolved

Description

The withdraw and withdraw2ContractV2 functions in SolConnector are not payable. However, these functions need to send LayerZero messaging fees during _lzSend.

Therefore, the contract must hold a native token balance to cover the required fees.

However, the contract does not provide a function that enables the owner or admin to withdraw its balance if necessary.

Recommendation

Consider adding a withdraw function that allows a trusted role to withdraw the balance.

Resolution

Orderly Team: The issue was resolved in commit d3897a2.

L-17 | Mixed Events In LedgerImplD

Category	Severity	Location	Status
Events	● Low	LedgerImplD.sol: 207-219,228-239	Resolved

Description

LedgerImplD.executeWithdraw2ContractV2() emits wrong events:

1. if state = 0, it emits AccountWithdrawSolFail without checking whether the destination chain type is EVM or SOL

2. for EVM withdrawals it emits AccountWithdrawSolApprove which is the wrong event.

Recommendation

To fix 1) manually check the chain type and emit the correct event - either AccountWithdrawApprove or AccountWithdrawSolApprove.

To fix 2) change the event emission to AccountWithdrawApprove

Resolution

Orderly Team: The issue was resolved in LedgerImplD.sol#L59.

L-18 | Weaponizing Swaps To Drain The Ledger

Category	Severity	Location	Status
Logical Error	● Low	evm-contracts	Acknowledged

Description

The Vault.delegateSwap() function opens up a window between executing the swap and and uploading it which can be leveraged by an attacker to drain the Ledger.

Example:
Initial state:
Alice deposits 100k USDC + 10 WETH
Bob deposits 3 WETH
Vault USDC = 100k; Vault WETH = 13

A swap is initiated via Vault.delegateSwap(). This swaps Bob's 3 WETH for 9000 USDC
• The vault now holds 109k USDC + 10 WETH (ledger not updated)
• The moment the swap was signed, Bob submitted a withdrawal request of their 3 WETH. Because the result of the swap is still not uploaded, Bob's WETH balance on the Ledger side is still 3, the balance check passes and a withdrawal message is successfully sent to the Vault
• The withdrawal is executed, Bob receives 3 WETH on the EVM chain and the Ledger updates their WETH balance to 0, as well as the chain's WETH balance to 10 WETH in accountWithdrawFinish
• The swap event is finally uploaded, executeSwapResultUpload() is called and Bob's and the chain's WETH balances are reduced by 3. Bob now has -3 WETH and the chain has 7 WETH.
• However, the USDC balances will also be updated. Bob's new USDC balance will become 9000 USDC and the chain's one - 109k USDC
• Now Bob can submit an USDC withdrawal request and withdraw 9000 USDC from the VaultFinal state:

Bob withdraws 3 WETH + 9000 USDC

Vault USDC = 100k USDC; Vault WETH = 7 WETHIn result, Bob stole 3 WETH from Alice.

Furthermore, Alice's balance on the Ledger is still 10 WETH so an attempt to withdraw it will revert on the EVM chain and block the messaging channel.

Recommendation

Consider reworking the swap mechanism to be initiated from the Ledger side - just like a withdrawal, by deducting the amount to be swapped from the account and the chain and then uploading a final result.

Resolution

Orderly Team: Acknowledged.

I-01 | Consider Having Rescue Functionality

Category	Severity	Location	Status
Informational	● Info	solana-vault	Acknowledged

Description

User funds are transferred from the user to the Vault on Solana during deposits, and the user's ledger balance increases after the cross-chain message is delivered.

However, there is no mechanism to recover funds if the cross-chain message fails.

Recommendation

Consider implementing an admin-only functionality to recover funds in cases where the source chain execution succeeds but the message delivery fails on the destination chain.

Resolution

Orderly Team: Acknowledged.

I-02 | Include System_program During Invoke

Category	Severity	Location	Status
Informational	● Info	oapp_lz_receive.rs: 155-156	Resolved

Description

The account_infos field in invoke and invoke_signed does not include the system_program in the codebase, only includes the from and to accounts.

However, according to the Solana and Rust documentation, this field should also include the system_program.

Example 2 from the [Solana documentation](https://solana.com/docs/core/cpi#anchor-framework)

A similar pattern can be seen for invoke_signed as well, in Example 2 [here](https://solana.com/docs/core/cpi#anchor-framework-1)

Lastly, this can also be seen in the [Rust documentation](https://docs.rs/solana-cpi/latest/solana_cpi/fn.invoke.html#examples).

Recommendation

Include system_program in account_infos when using invoke and invoke_signed.

Resolution

Orderly Team: The issue was resolved in commit 2e851ec.

I-03 | Incomplete bizType Comment Description

Category	Severity	Location	Status
Informational	● Info	EventTypes.sol: 18	Resolved

Description

The comment next to the the bizType field of the EventUploadData struct describes the event from 1 to 13.

```
struct EventUploadData {
uint8 bizType; // 1 - withdraw, 2 - settlement, 3 - adl, 4 - liquidation, 5 - fee
distribution, 6 - delegate signer, 7 - delegate withdraw, 12 - balance transfer, 13 - swap
result upload
uint64 eventId;
bytes data;
}
```

As we can see events 8-12 are not described and event 14 - withdraw2contractV2 is not present as well.

Recommendation

Consider adding the missing bizTypes in the comment

Resolution

Orderly Team: The issue was resolved in commit b97cb8e.

I-04 | Settlement Balance Check Is Commented

Category	Severity	Location	Status
Warning	● Info	LedgerImplA.sol: 274-278	Acknowledged

Description

The code from LedgerImplA.executeSettlement() changes the user's balance according to the settled amount.

Before, there was a check that reverts the transaction if the new user balance is negative, but now it's commented out. At the same time, the comment explaining the check is not removed.

```
// check balance + settledAmount > 0, where balance should cast to int128 first
int128 balance = account.balances[settlement.settledAssetHash];
// if (balance + ledgerExecution.settledAmount < 0) {
//     revert BalanceNotEnough(balance, ledgerExecution.settledAmount);
// }
account.balances[settlement.settledAssetHash] = balance + ledgerExecution.settledAmount;
```

If insuranceTransferAmount = 0, the check from the above if statement will be sufficient, but otherwise this will allow users' balances to go below 0.

Recommendation

Revisit if the check should stay commented out.

Resolution

Orderly Team: Acknowledged.

I-05 | OApps Don't Call Skip(), Burn() Or Clear()

Category	Severity	Location	Status
Best Practices	● Info	Global	Acknowledged

Description

The SolConnector and the Solana Vault are OApps that can opt-in for ordered delivery. Neither of the two has a way to call skip(), burn() or clear(), they should instead be called by the delegate and the nonce should be updated after that.

LayerZero recommends as a best practice to have the calls to these functions and the nonce update inside the contract to minimize the chances of failed synchronization.

Recommendation

Consider adding a way to call these functions and update the nonce afterwards.

Resolution

Orderly Team: Acknowledged.

I-06 | Hardcoded Roles

Category	Severity	Location	Status
Best Practices	● Info	FeeManager.sol: 21,49-51,83-85	Acknowledged

Description

The newly added SYMBOL_MANAGER_ROLE and BROKER_MANAGER_ROLE are used in FeeManager, VaultManager and Vault.

They are hardcoded in each contract instead of having one central place to fetch them from. This increases the risk of misspelling a role and makes updating the code harder.

Recommendation

Use constants instead and import them in the appropriate files.

Resolution

Orderly Team: Acknowledged.

I-07 | Deposit Nonce Can Be Changed

Category	Severity	Location	Status
Configuration	● Info	set_vault.rs: 36	Resolved

Description

The deposit_nonce field can be changed by calling the set_vault instruction. If the new value set is less than the current one, nonces will start repeating, defeating its purpose and causing confusion to offchain listeners.

Recommendation

Either remove the ability to modify the deposit_nonce entirely or revert if the new value is below the current one.

Resolution

Orderly Team: The issue was resolved in commit e040137.

I-08 | Outdated Comment

Category	Severity	Location	Status
Informational	● Info	oapp_lz_receive_types.rs: 19	Resolved

Description

The comment "Msg.Type: Withdraw (currently at most 13 accounts, otherwise tx oversize)" in oapp_lz_receive_types.rs is outdated and misleading.

Recommendation

Consider removing the comment, as it pertains to a previous version of the codebase.

Resolution

Orderly Team: The issue was resolved in commit 23778dc.

I-09 | SwapSignature Library Is Not Used

Category	Severity	Location	Status
Informational	● Info	contract-evm	Acknowledged

Description

The `contract-evm` repository contains the `SwapSignature` library contract. However, this library is not used in the codebase; instead, `DelegateSwapSignature` is utilized.

Recommendation

Consider whether two different libraries are required for the signature verification.

Resolution

Orderly Team: Acknowledged.

I-10 | Can Set Arbitrary Hashes For Roles

Category	Severity	Location	Status
Best Practices	● Info	set_manager_role.rs and set_broker.rs	Resolved

Description

Both set_manager_role.rs and set_broker.rs take in hashes with type [u8; 32] for roles. These roles are used for functions like set_token() to validate the caller has the right role.

There is no validation on that hash when assigning someone a role so it would be very easy to have a typo.

Recommendation

Either add on-chain validation by defining potential role hashes in constants or an enum, or make sure to use off-chain validation (like a frontend that only allows you to set certain roles) to avoid typos.

Resolution

Orderly Team: The issue was resolved in commit 196e1f2.

I-11 | Executor Needs Funds To Initialize ATA

Category	Severity	Location	Status
Informational	● Info	SolConnector.sol: 186-188	Resolved

Description

In case the receiver_token_account in the lz_receive instruction is empty, it will be initialized and the Executor will have to pay for that. This should be considered when setting the msgOptions.value in the SolConnector.

LayerZero advices to have extraOptions passed by the user if the recipient account will require initialization. This prevents overcharging on each withdrawal (in case the account is not empty).

However, letting the user provide value is not a good idea because they can send a reverting transaction on purpose to DOS the app if it is using ordered delivery.

Recommendation

Account for this possible ATA initialization when setting msgOptions.value.

Resolution

Orderly Team: Resolved.

I-12 | Withdrawals May Drain The Connector

Category	Severity	Location	Status
Informational	● Info	SolConnector.sol: 130,162	Acknowledged

Description

Since SolConnector pays for each relayed message to Solana Vault, frequent withdrawals may cause it to run out of funds.

The existing fee mechanism will probably disincentivize users from spamming such requests, but only if the withdrawn amount is high enough.

Recommendation

You can introduce minimum amount to be withdrawn and control the fee accordingly so users don't spam requests.

In addition, you can consider adding a mechanism which makes the user pay for their withdrawal fee.

Resolution

Orderly Team: Acknowledged.

I-13 | Technically Possible Underflow

Category	Severity	Location	Status
Best Practices	● Info	oapp_lz_receive.rs: 141	Resolved

Description

The let amount_to_transfer = withdraw_params.token_amount - withdraw_params.fee; on line 141 of oapp_lz_receive.rs can theoretically underflow.

It won't underflow because of this check in the source chain:

https://github.com/GuardianOrg/orderly-v2-contract-evm-orderlysolvault-team1/blob/2adf4141392c0547ce37361f70eb765160e26421/src/LedgerImplC.sol#L86

But it would still be a best practice, to use checked_sub here. Specifically, not saturating_sub but checked_sub (read here).

Recommendation

It would be a best practice, to use checked_sub here. Specifically, not saturating_sub but checked_sub (read here).

Resolution

Orderly Team: The issue was resolved in commit 47ffbf4.

I-14 | ATA Check Is Not Used For The Logical Forks

Category	Severity	Location	Status
Superfluous Code	● Info	oapp_lz_receive.rs: 130-138	Resolved

Description

Look for check if the `receiver_token_account` is the correct associated token account in `oapp_lz_receive.rs` (lines of code).

That block of code where the program gets the receiver ATA is not needed when transferring SOL. It is not used in the if `token_index = TOKEN_INDEX_SOL` { logic fork.

Recommendation

Move the `get_associated_token_address` check into the `else` logical fork of the `if token_index = TOKEN_INDEX_SOL` .

Resolution

Orderly Team: The issue was resolved in commit 6c6fa50.

I-15 | SOL Mint_account Should Stay 0

Category	Severity	Location	Status
Informational	● Info	deposit.rs: 49	Resolved

Description

In order to use a deposit_token in the deposit.rs instruction the following condition must be true:

deposit_token.key() = allowed_token.mint_account allowed_token.allowed = true.

The allowed_token is a PDA which will have allowed = true for all the supported SPLs and the native SOL token.

The SOL deposits are handled in a different instruction named deposit_sol.rs, where the mint_account field is not used.

If mint_account were ever set to a valid mint_account it would enable executing the deposit instruction with that token.

Recommendation

You should never set the mint_account field to a value different than 0.

Resolution

Orderly Team: The issue was resolved in commit f8cd1b9.

Remediation Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	Delegated Withdrawals Cannot Be Executed	DoS	● High	Resolved
H-02	Withdrawals Can Block The Channel	DoS	● High	Resolved
M-01	Escrowed Balance Can Be Withdrawn	Validation	● Medium	Acknowledged
L-01	Changed Schema For Event Uploading	Validation	● Low	Acknowledged
L-02	Duplicate lastEngineEventId Assignment	Superfluous Code	● Low	Resolved
L-03	batchGetUserLedger Doesn't Include Escrow	Best Practices	● Low	Acknowledged
L-04	Missing Funds Because Of Balance Transfers	Unexpected Behavior	● Low	Acknowledged
I-01	Redundant Comment	Best Practices	● Info	Acknowledged
I-02	Deposit Transaction Sizes Can Be Decreased	Best Practices	● Info	Acknowledged
I-03	Typo	Informational	● Info	Resolved
I-04	Unused _checkAccount() Function	Best Practices	● Info	Resolved

H-01 | Delegated Withdrawals Cannot Be Executed

Category	Severity	Location	Status
DoS	● High	OperatorManagerImplB.sol: 47-49	Resolved

Description

OperatorManagerImplB._processEventUpload() was modified to fetch the needed function selector for the current operation depending on the bizType.

After that dataOffset = 32 is executed if the function to be executed includes a dynamic struct in its parameters. The actual call is then performed and the calldata construction differs between dataOffset = 0 and dataOffset = 32.

The functions related to bizType 1 and bizType 7 have different names, but both of them accept the EventTypes.WithdrawData dynamic struct. However, the dataOffset is assigned only for bizType 1.

Because of this, the calldata will be incorrectly encoded and any attempts to execute executeDelegateWithdrawAction() will either revert, or worse, execute with unexpected inputs if the calldata can be decoded into the needed fields.

If executeDelegateWithdrawAction() revert not only users won't receive their funds, but the whole batch upload will be blocked.

Recommendation

Include bizType = 7 in the if statement

```
•        if (data.bizType = 1 || data.bizType = 2 || data.bizType = 4 || data.bizType = 9 ||
data.bizType = 10) {  // dynamic event types: withdraw, settlement, liquidation,
liquidationV2, withdrawSol
+        if (data.bizType = 1 || data.bizType = 2 || data.bizType = 4 || data.bizType = 7 ||
data.bizType = 9 || data.bizType = 10) {  // dynamic event types: withdraw, settlement,
liquidation, delegateWithdraw, liquidationV2, withdrawSol
dataOffset = 32;    // 0x20 for dynamic event types
}
```

Resolution

Orderly Team: The issue was resolved in commit 6c1db46.

H-02 | Withdrawals Can Block The Channel

Category	Severity	Location	Status
DoS	● High	oapp_lz_receive.rs	Resolved

Description PoC

The M-01 issue of the main review has to be fixed because if not, anyone can block the EVM > Solana pathway, especially when `inbound_nonce` cannot be modified anymore because of the change in the `set_vault` instruction.

As discussed in the previous issue, if the receiving account is executable or its balance after the transfer will be below what's required to keep that account rent exempt, the transaction will revert.

In addition, if the receiver is a reserved account, for example the System program or the `SysVar`, Solana will demote that to read-only even if it was passed as writable: true in the transaction.

This will cause a failure when Anchor applies the accounts constraints, before any of the program code is executed, because of the mut constraint of the receiver.

Recommendation

To fix the problem with the executable accounts and also the rent exemption issue, execute the lamports transfer only if the account will be exempt and is not executable. However, the write demotion issue is not easily fixable.

Consider adding a list of reserved accounts to the BE and perform checks before submitting a Solana withdrawal on the Ledger side. This will reduce the possibility of the `receiver` being demoted.

The list of reserved accounts has to be updated as Solana updates these accounts. In addition, be ready to skip a given nonce if such account slipped through.

Resolution

Orderly Team: The issue was resolved in commit 4582ca7.

M-01 | Escrowed Balance Can Be Withdrawn

Category	Severity	Location	Status
Validation	● Medium	LedgerImplD.sol	Acknowledged

Description

LedgerImplD.executeWithdraw2ContractV2() allows users to withdraw their whole balance, including the escrowed one which has still not been fully transferred. There is a TODO comment in the _executeWithdraw2SOL that talks about this check.

```
else if (account.balances[tokenHash] < withdrawV2.tokenAmount.toInt128()) { // TODO: check
escrowBalance
revert WithdrawBalanceNotEnough(account.balances[tokenHash], withdrawV2.tokenAmount);
}
Deploying without this check will allow anyone to withdraw their escrowed balance which
doesn't align with the behavior for normal withdrawals.
```

Recommendation

Consider implementing the check in both _executeWithdraw2SOL() and _executeWithdraw2EVM()

Resolution

Orderly Team: Acknowledged.

L-01 | Changed Schema For Event Uploading

Category	Severity	Location	Status
Validation	● Low	OperatorManagerImplB.sol: 52	Acknowledged

Description

Before the fix review, the `EventUploadData.data` field was the abi encoded struct for the given function.

However, the new code in `OperatorManagerImplB._processEventUpload()` cuts the first `dataOffset` bytes out if the `data` field

```
bytes memory dataWithoutOffset = abi.encodePacked(data.data[dataOffset:]);
```

This means the schema for the `data` field has to be also changed for `bizType` where `dataOffset = 0` in such a way that the encoded struct starts from a `dataOffset` position.

Recommendation

Consider if this change in the schema is desired. If not, you can use the raw `data` field without slicing.

Resolution

Orderly Team: Acknowledged.

L-02 | Duplicate lastEngineEventId Assignment

Category	Severity	Location	Status
Superfluous Code	● Low	LedgerImplD.sol: 250	Resolved

Description

In the _executeWithdraw2SOL function of the LedgerImplD contract, lastEngineEventId is assigned twice, once at line 236 and again at line 250.

Recommendation

Remove one of the assignments.

Resolution

Orderly Team: The issue was resolved in commit 51a7a28.

L-03 | batchGetUserLedger Doesn't Include Escrow

Category	Severity	Location	Status
Best Practices	● Low	AccountTypes.sol: 82-88	Acknowledged

Description

The Ledger.batchGetUserLedger returns a snapshot of the user balances, one of which fields is AccountTokenBalances tokenBalances.

This struct includes the balance and frozenBalance of the user, but not their escrowBalance. This missing piece of information may result in the BE making wrong decisions, depending on how it's used.

```
struct AccountTokenBalances {
// token hash
bytes32 tokenHash;
// balance & frozenBalance
int128 balance;
uint128 frozenBalance;
}
```

Recommendation

Consider adding escrowBalances in the AccountTokenBalances struct

Resolution

Orderly Team: Acknowledged.

L-04 | Missing Funds Because Of Balance Transfers

Category	Severity	Location	Status
Unexpected Behavior	● Low	LedgerImplC.sol	Acknowledged

Description

LedgerImplC.executeBalanceTransfer() executes transfers between users, but it allows the receiver to use the funds in the Orderly system before they have been removed from the sender account.

This approach is not efficient because it allows the sender of the funds to also use these funds, or even withdraw them after they have been credited, but yet not debited.

Then, when the transfer is being finalized, _applyDebit() will reduce the sender balance and it will drop to a negative value if it cannot cover the initially transferred amount.

```
fromAccount.subBalance(tokenHash, amount);
```

Funds have been stolen from the system in result of using the balance transfer feature.

Recommendation

Reconsider the addition of this feature.

Resolution

Orderly Team: Acknowledged.

I-01 | Redundant Comment

Category	Severity	Location	Status
Best Practices	● Info	deposit_sol.rs: 121	Acknowledged

Description

When vault_deposit_params are declared in deposit_sol:apply(), there is a redundant empty comment after user_address.

```
let vault_deposit_params = VaultDepositParams {
account_id: deposit_params.account_id,
broker_hash: deposit_params.broker_hash,
user_address: deposit_params.user_address, //
token_hash: SOL_TOKEN_HASH,
src_chain_id: ctx.accounts.vault_authority.sol_chain_id,
token_amount: deposit_params.token_amount as u128,
src_chain_deposit_nonce: ctx.accounts.vault_authority.deposit_nonce,
};
```

Recommendation

Remove the comment

Resolution

Orderly Team: Acknowledged.

I-02 | Deposit Transaction Sizes Can Be Decreased

Category	Severity	Location	Status
Best Practices	● Info	deposit_sol.rs: 86	Acknowledged

Description

The `deposit_sol` instruction doesn't read `DepositParams.token_hash` anymore, but the field is still needed to initiate the transaction which causes a larger overall transaction size.

Recommendation

You can use different struct for `deposit_sol` which doesn't include `token_hash` in order to reduce the tx size.

Resolution

Orderly Team: Acknowledged.

I-03 | Typo

Category	Severity	Location	Status
Informational	● Info	Vault.sol: 681	Resolved

Description

The name of the `Vault.attempTransferETH()` function is spelled incorrectly. It should be `attemptTransferETH()`.

Recommendation

Fix the typo.

Resolution

Orderly Team: The issue was resolved in commit fca2f00.

I-04 | Unused _checkAccount() Function

Category	Severity	Location	Status
Best Practices	● Info	LedgerImplD.sol: 268-270	Resolved

Description

The _checkAccount() function in LedgerImplD.sol is not used.

Recommendation

Remove the function.

Resolution

Orderly Team: The issue was resolved in commit 5303ac3.

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