GUARDIAN AUDITS

SMART CONTRACT SECURITY AUDIT OF

Orderly Network

Summary

Audit Firm Guardian

Prepared By Owen Thurm, Daniel Gelfand, Kiki_dev

Client Firm Orderly Network

Final Report Date October 23, 2023

Audit Summary

Orderly Network engaged Guardian to review the security of its perpetuals protocol utilizing an off-chain order book. From the 1st of October to the 13th of October, a team of 3 auditors reviewed the source code in scope. All findings have been recorded in the following report.

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

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

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/OrderlyEVMContractsSuite

Table of Contents

Project Overview

Project Summary

Project Name	Orderly Network
Language	Solidity
Codebase	https://gitlab.com/orderlynetwork/orderly-v2/contract-evm/ https://gitlab.com/orderlynetwork/orderly-v2/evm-cross-chain/
Commit(s)	contracts-evm: a3fd9ff7b9bdffa5cd21260e95f5cf62d224ae22 evm-cross-chain: 348abd9c77bc624c0d0abbb2d5d5c3f7d8988b9b

Audit Summary

Delivery Date	October 13, 2023
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	3	0	0	1	0	2
High	2	0	0	1	0	1
Medium	17	0	0	12	0	5
Low	19	0	0	8	0	11

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
LDL	LedgerDataLayout.sol	5b1d58f2b4a52c67bb093dab08d0fc4e9e237efa
OMDL	OperatorManagerDataLayout.sol	aeefcecbfa8eda69568575a08a3b5a5a8efca9ce
IOM	IOperatorManager.sol	eb4d6ec8ce1eb59ca0c926a17dc02ac2ba4c2e4a
ILE	ILedgerEvent.sol	eb8bfc14634a6fd83528665cf9021a8b91b39d55
ILCCM	ILedgerCrossChainManager.sol	9aeedb1ddc356352e88a9c4d81567fb7e899c82b
IOMC	IOperatorManagerComponent.sol	d082e422fa1010e030ba519dc881750f3f298ab7
ILC	ILedgerComponent.sol	39349d3bf564e94b38ee52b6a164be1377c2007d
IVLT	IVault.sol	4f4832b8db3869d11c479c044b97ae605fe56c4a
IMM	IMarketManager.sol	883138df555ac68eee89b2bf73de04eb3d5d412a
IFM	IFeeManager.sol	93d5c83a5fc07b440b22b514d5977ac9f08c88e0
ILE	ILedgerError.sol	758e51f7a59a13ac0da19cce1b8e1c032a1df072
IVCCM	IVaultCrossChainManager.sol	36e04b807e14e7b623ae2885b94526fa311acec3
ILGR	ILedger.sol	7799b9cc8207503d4d775abe788cda9aa181201b
IVM	IVaultManager.sol	21ec304215cb55b1841d47d266367565b17c117b
MKTM	MarketManager.sol	99ec0ae78454457483a13155c6b58e91e792c701
OPMAN	OperatorManager.sol	f273278cf952f9b29b863d8b8af7f633ac3f4496
OMC	OperatorManagerComponent.sol	e6972c540a070aea2c4393cf0fc6610178c7ff7e
FM	FeeManager.sol	063dc961bee56f7e952bae673854c65a83f78ac4
VAULT	Vault.sol	bfc2f73a44ae62d1b3731a6455089a78ed23e5f6
TUSDC	tUSDC.sol	8abdd2a89dc51ceed77ce12401d91f05332e0194

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
LC	LedgerComponent.sol	902d2a9a369110d54ba3ecb9db3bc6e8dc629b02
CCMT	CrossChainMessageTypes.sol	1f35b03d05f2bbd9a5c97b7f5e56a401ca71126b
ETYP	EventTypes.sol	02940ca1056077e2f4e49bb975eefc40da1c1193
PRPT	PerpTypes.sol	1dc6549cafab3d58c265ed02d83767eede9e534b
MKTT	MarketTypes.sol	937620243c02f976aa1741cfcc18ea09e8443d05
ACTT	AccountTypes.sol	be4e53fd41c1a0cab6a1329926b379bd9e396f3c
VLTT	VaultTypes.sol	1c5188d9dc8b9add9fd141d85ded86a627fe7d19
MTH	MarketTypeHelper.sol	749dc292d4a0009dbd2cff4cb4a3ffb1adf93506
ATH	AccountTypeHelper.sol	875915ccfad06a820bb1ec926749ed445af18261
ATPH	AccountTypePositionHelper.sol	53a24f76967a2e19d5c900e853c218500a3dd07a
SCH	SafeCastHelper.sol	532d04c29dbdad514b13745d6fe258a053c42f4a
SIG	Signature.sol	8bc8c6290e8a7f259b7570977722938aa916a6c3
UTIL	Utils.sol	9e87d98ac083758d1752170acc4f8782b5cc6a57
LGR	Ledger.sol	275d343f0e64a9e89d848a3da1cef3189565ed5d
VAULTM	VaultManager.sol	b9d907ae831d4bf527638c42ed755d9765311858
IOCC	IOrderlyCrossChain.sol	a916f8a76497976cc2a2f80740b4bee8440bce5e
ICCM	ICrossChainManager.sol	155243ee1b7703c48dfee413143180f1fbdb0836
ILCCM	ILedgerCrossChainManager.sol	fa09dc22df5b67c4022e4a2b3da5c471bf27cf05
IVCCM	IVaultCrossChainManager.sol	f603284419673521d776c6e9de768d11fe2bc6e4
LZM	LZEndpointMock.sol	2e482fae69005feb7c4c6d5539c8aa722eb9d630

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
ESC	ExcessivelySafeCall.sol	27563f19821da6d9e59e31996e33cd0af4e8c2cc
BYTES	BytesLib.sol	90329d1b8d0302c4824da5a42d311e03703a3094
LZLIB	LzLib.sol	87432acd7c23e352f1cb13a96fdff82cc0658873
LZAU	LzAppUpgradeable.sol	8e6ec27ca84ad51b6f64e3a9f07ef581f62f1021
LZA	LzApp.sol	be1d614811a07a63c50a8d100129f4159697b52b
NBLZA	NonblockingLzApp.sol	57afe74643d79539dd4cf82d464d08eb4b9f2a8e
ILZR	ILayerZeroReceiver.sol	40fdfafbbd43411c04fab817e023e5162ab86386
ILZUA	ILayerZeroUserApplicationConfig.sol	619d629e55c336e6297b51444c9751b25d699b7c
CCRP	CrossChainRelayProxy.sol	eebf6f108b3022685df326c030aa6d6fbf475b3c
LCMU	LedgerCrossChainManagerUpgradeable.sol	7b2df71fb191287cacaa43d01e2d0a1e21357f99
OCCM	OrderlyCrossChainMessage.sol	d75a2d7d4cb64f7622d8920b67715d386c256015
CCRU	CrossChainRelayUpgradeable.sol	4a45fd60da745d887cb70cde746c4e349faba630
CCMP	CrossChainManagerProxy.sol	b79b14d267f830b0ae8b51f9c4528233a61767bb
VCCMU	VaultCrossChainManagerUpgradeable.sol	2d2d83a746f49913ddd690f6be67b3d37c6fa5ff

Audit Scope & Methodology

Vulnerability Classifications

Vulnerability Level	Classification
🔴 Critical	Easily exploitable by anyone, causing loss/manipulation of assets or data.
🟠 High	Arduously exploitable by a subset of addresses, causing loss/manipulation of assets or data.
🟡 Medium	Inherent risk of future exploits that may or may not impact the smart contract execution.
🟢 Low	Minor deviation from best practices.

Methodology

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
LGR-1	Duplicated insuranceTransferAmount	Logical Error	Critical	Resolved
LCMU-1	Any User Can Set A Token's Decimals	Access Control	Critical	Resolved
GLOBAL-1	Malicious User Drains CrossChainRelay	DoS	Critical	Acknowledged
LGR-2	Rounded Frozen Balance Bricks Withdrawals	Precision	High	Acknowledged
ATPH-1	Incorrect Decimals	Precision	High	Resolved
GLOBAL-2	LayerZero Message Blocking	DoS	Medium	Acknowledged
GLOBAL-3	Centralization Risk	Centralization Risk	Medium	Acknowledged
VAULT-1	Allowed Token Contract Address Added	Logical Error	Medium	Resolved
ATPH-2	Half Rounding Improperly Handles 0 Quotient	Logical Error	Medium	Resolved
ATPH-3	Average Entry Can Be Rounded In User's Favor	Logical Error	Medium	Resolved
GLOBAL-4	Vault Can Be Drained On Specific Chain	Protocol Manipulation	Medium	Acknowledged
LGR-3	User Can Withdraw Entire Collateral When Open Position	Logical Error	Medium	Acknowledged
VAULTM-1	Funds Will Be Locked On Hard Fork	Hard Fork	Medium	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
LGR-4	Large Withdrawals Can Fail	Logical Error	Medium	Acknowledged
SIG-1	Signature Malleability	Signatures	Medium	Resolved
MKTM-1	Incorrect updatedAt Assigned	Logical Error	Medium	Resolved
LGR-5	Potential For Trapped Deposits	Trapped Funds	Medium	Acknowledged
LGR-6	Insurance Account May Become Insolvent	Logical Error	Medium	Acknowledged
GLOBAL-5	Liquidations When Deposits Paused	Logical Error	Medium	Acknowledged
GLOBAL-6	Lacking Storage Gaps	Storage Gaps	Medium	Acknowledged
OPMAN-1	Risk Of DoS	DoS	Medium	Acknowledged
ATPH-4	DoS On Average Entry Price	DoS	Medium	Acknowledged
GLOBAL-7	Unnecessary Timestamp Emitted	Superfluous Code	Low	Resolved
PRPT-1	Unused Side Attribute	Superfluous Code	Low	Acknowledged
ETYP-1	Unused liquidationTransferId attribute	Superfluous Code	Low	Acknowledged
SFCH-1	Inaccurate Overflow Error	Errors	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
GLOBAL-8	Multiple Sources Of Truth	Warning	Low	Acknowledged
ATPH-5	MaintenanceMargin Does Not Match Documentation	Documentation	Low	Acknowledged
ATPH-6	Unsafe Casting	Casting	Low	Resolved
VAULT-2	Missing Check-Effect-Interact Pattern	Reentrancy	Low	Resolved
LGR-7	Redundant for-loop	Optimization	Low	Resolved
ATPH-7	Typo	Typo	Low	Resolved
LGR-8	Unused Helper Functions	Superfluous Code	Low	Resolved
CCRU-1	Hardcoded Zero Address	Hardcoded Value	Low	Acknowledged
ATPH-8	Multiplication On The Result Of Division	Precision	Low	Resolved
VCCMU-1	Excess Fee Locked in Contract	Trapped Funds	Low	Resolved
LGR-9	Excess Insurance Fund Transfer Protection	Validation	Low	Acknowledged
LGR-10	Position Never Cleared	Optimization	Low	Resolved
CCRU-2	Missing Address 0 Check	Validation	Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
GLOBAL-9	Use LayerZero Package	Maintainability	Low	Acknowledged
UTIL-1	Unnecessary Util Functions	Superfluous Code	Low	Resolved

LGR-1 | Duplicated insuranceTransferAmount

Category	Severity	Location	Status
Logical Error	● Critical	Ledger.sol: 374	Resolved

Description

In the executeSettlement function, if the settlement.insuranceTransferAmount amount is nonzero it is added to both the insuranceFund.balances and the account.balances therefore duplicating the settlement.insuranceTransferAmount across these two accounts.

The validation on the insuranceTransferAmount indicates that the settlement.insuranceTransferAmount should be deducted from the insuranceFund.balances and added or “transferred” to the account.balances.

Recommendation

Modify the following lines:

```
insuranceFund.balances[settlement.settledAssetHash] += settlement.insuranceTransferAmount;
account.balances[settlement.settledAssetHash] += settlement.insuranceTransferAmount;
```

Such that the balance is transferred rather than duplicated:

```
insuranceFund.balances[settlement.settledAssetHash] -= settlement.insuranceTransferAmount;
account.balances[settlement.settledAssetHash] += settlement.insuranceTransferAmount;
```

Resolution

Orderly Team: Fixed.

LCMU-1 | Any User Can Set A Token's Decimals

Category	Severity	Location	Status
Access Control	● Critical	LedgerCrossChainManagerUpgradeable.sol: 39	Resolved

Description

There is a lack of access control on the `setTokenDecimal` function, therefore any user can set an arbitrary decimal amount for any `tokenHash` on any `tokenChainId`. This can easily be used to inflate how many tokens a user has on-chain and drain the vault.

For example, assume WETH has 18 decimals on chain A and 18 decimals on chain B. However, Alice sets the decimals for chain A as 18 and the decimals for chain B as 19. Upon converting 1 ETH from Chain A, the amount of WETH on Chain B after conversion is `tokenAmount * uint128(10 ** (dstDecimal - srcDecimal))` = `1e18 * 10**1 = 1e19`.

Alice was just able to turn 1 WETH on chain A into 10 WETH on chain B, and can withdraw those extra funds from the vault. The theft can be even more drastic by increasing the decimal spread between chains.

Recommendation

Ensure only the owner can set the token decimals.

Resolution

Orderly Team: Fixed.

GLOBAL-1 | Malicious User Drains CrossChainRelay

Category	Severity	Location	Status
DoS	● Critical	Global	Acknowledged

Description

There is no minimum deposit amount in the Vault contract, therefore a malicious actor may make many deposits with trivial amounts to drain the `CrossChainRelay` of its Ether and halt execution in the system.

Recommendation

Implement a minimum deposit amount in the `Vault` contract such that DoS attacks like this one become unfeasible.

Resolution

Orderly Team: We have proposed a solution to this issue. As mentioned, we will require users to cover the cross-chain fee, this feature will be released in the next sprint.

LGR-2 | Rounded Frozen Balance Bricks Withdrawals

Category	Severity	Location	Status
Precision	High	Ledger.sol: 319	Acknowledged

Description PoC

When making a withdrawal, the action goes from the ledger chain to the vault chain back to the ledger chain. When sending the message to the vault, `withdraw` is called which converts the fee to the proper amount of decimals on the vault chain using `convertDecimal`. However, truncation may occur due to the collateral on the vault chain having fewer decimals than on the ledger chain.

For example, let's assume the fee is initially 1 wei and the decimal spread between chains is 12. When `executeWithdrawAction` is triggered on the ledger, 1 wei of fees have been accounted for such that the `VaultManager` freezes 1 less wei than the `withdraw.tokenAmount` transmitted: `vaultManager.frozenBalance(tokenHash, withdraw.chainId, withdraw.tokenAmount - withdraw.fee)`.

Once `ILedgerCrossChainManager(crossChainManagerAddress).withdraw(withdraw)` is called, decimal conversion occurs and the new fee to be sent to the vault chain is `1 / 10**12 = 0`. When the message finally arrives back to the ledger to finish withdrawal `vaultManager.finishFrozenBalance(withdraw.tokenHash, withdraw.chainId, withdraw.tokenAmount - withdraw.fee)` is called.

Because the `withdraw.fee` is smaller than accounted for initially (1 wei vs 0 wei), more is unfrozen than was originally frozen upon `executeWithdrawAction`, which will cause an arithmetic underflow in the `VaultManager` and brick all withdrawals until a user deposits to cover the deficit. This can continuously be done maliciously to continuously block withdrawals from occurring.

Recommendation

Round beforehand such that the rounded value when finishing the withdrawal is the same as when starting the withdrawal. Alternatively, use higher precision for token amounts although this may be a considerable refactor.

Resolution

Orderly Team: This is an known issue and wont fix in codes. The key point is that the decimal of Ledger should be less or equal to the decimal of vaults, thus no round occurs.

ATPH-1 | Incorrect Decimals

Category	Severity	Location	Status
Precision	High	AccountTypePositionHelper.sol: 72	Resolved

Description

When the `calAverageEntryPrice` function is called during a liquidation or ADL execution, the `liquidationQuoteDiff` has 6 decimals of precision. Therefore the resulting `quoteDiff` on line 72 has 14 decimals of precision.

This differs from the `quoteDiff` decimals of 16 when uploading a trade.
Therefore the `averageEntryPrice` and `openingCost` values are perturbed when the `quoteDiff` is used to calculate the average entry during a liquidation or ADL.

Recommendation

Adjust the `quoteDiff` during liquidation or ADL such that it has the expected 16 decimals of precision.

Resolution

Orderly Team: Fixed.

GLOBAL-2 | LayerZero Message Blocking

Category	Severity	Location	Status
DoS	Medium	Global	Acknowledged

Description

LayerZero implements blocking functionality such that when a transaction on the destination chain fails, before any new transactions can be executed, the failed transaction has to be retried until success.

A malicious user can submit a deposit or withdrawal they know will fail, and will prevent all other deposits and withdrawals from occurring, leading to loss of protocol functionality and loss of funds for those who already deposited.

Some methods a malicious user can use to force such a scenario include but are not limited to:

1) Broker is allowed for Vault on `srcChain` but disallowed on the `VaultManager` on `dstChain`. The configuration of these two contracts aren't atomic.
2) `depositTo` for an address that is blacklisted for the collateral token. When the blacklisted address triggers a withdraw action, the Vault withdrawal will revert.

Recommendation

Consider utilizing a non-blocking approach as described in LayerZero documentation.

Resolution

Orderly Team: We can handle blocking scenarios, and we also have a force resume function to force drop a message.

GLOBAL-3 | Centralization Risk

Category	Severity	Location	Status
Centralization Risk	Medium	Global	Acknowledged

Description

Throughout the smart contract system there is a lack of validation to prevent privileged addresses from taking malicious actions or even committing errors that have drastic consequences.

- Executing malicious withdrawals, settlements, ADLs, and liquidations.
- No validation that the `liquidationFee = liquidatorFee + insuranceFee`.
- No validation that the ratio of `positionQtyTransfer` to `costPositionTransfer` is accurate to the `adlPrice` provided.
- No validation that tokens being actively used as collateral cannot be removed from support, causing liquidations and insolvency.
- No validation that `trade.notional = trade.tradeQty * trade.executedPrice` in the `executeProcessValidatedFutures` function.
- No cap on configured values such as the `maxWithdrawalFee` and `liquidationFeeMax`.
- No cap on the `liquidationFee` as the `liquidationFeeMax` is unused at the contract level.

Recommendation

Consider implementing validations to prevent any potential errors privileged addresses may make. And be sure to document the risks of these privileged abilities.

Resolution

Orderly Team: Acknowledged.

VAULT-1 | Allowed Token Contract Address Added

Category	Severity	Location	Status
Logical Error	Medium	Vault.sol: 57	Resolved

Description

The documentation for `setAllowedToken` states that the function is supposed to “Add contract address for an allowed token given the tokenHash.”

However, `setAllowedToken` only adds or removes the `tokenHash` from the `allowedTokenSet`. As a result, the address for the token is never added to the `allowedToken` mapping.

Deposits and withdrawals will revert as the zero address does not have function `safeTransferFrom`, and continue to revert until `changeTokenAddressAndAllow` is called, which according to the documentation is an "unusual case on Mainnet".

Recommendation

Add a parameter for the token address and perform `allowedToken[_tokenHash]=_tokenAddress`.

Resolution

Orderly Team: Fixed.

ATPH-2 | Half Rounding Improperly Handles 0 Quotient

Category	Severity	Location	Status
Logical Error	Medium	AccountTypePositionHelper.sol	Resolved

Description PoC

Orderly utilizes the `halfUp` and `halfDown` methods to round the quotient up or down on the magnitude of the fractional part.
However, the edge case of a 0 quotient will cause the division to return a mathematically incorrect result.
For example, consider the function call `halfUp16_8(10, 11)`

```
int256 quotient = dividend / divisor = 10 / 11 = 0
int256 remainder = dividend % divisor = 10 % 11 = 10
if (10 * 2 >= 11) {
    if (quotient > 0) {
        quotient += 1;
    } else {
        quotient -= 1; <- quotient = 0 - 1 = -1
    }
}
```

The returned quotient is `-1` although `10 / 11 = 0.909` is not supposed to be a negative number and should round to `1`.

Recommendation

Add 1 to the quotient when `quotient >= 0`

Resolution

Orderly Team: Fixed.

ATPH-3 | Average Entry Can Be Rounded In User's Favor

Category	Severity	Location	Status
Logical Error	Medium	AccountTypePositionHelper.sol: 84	Resolved

Description

When rounding operations are performed, it is safer for the protocol to round against the users so that there is a smaller likelihood of needing to use the insurance fund or ADL.

The average entry price is calculated using `position.averageEntryPrice = halfDown16_8(-openingCost, currentHolding).toUint128()` which rounds up if the fractional part is greater than half, and down otherwise.

When a trader is long, it is possible for the entry price to round down which would provide the user a superior entry as they want to buy as low as possible.

When a trader is short, it is possible for the entry price to round up which would provide the user a superior entry as they want to sell as high as possible.

Recommendation

Round against the user depending on their trade direction.

Resolution

Orderly Team: The recommendation was implemented in commit `5dc96c9b`.

GLOBAL-4 | Vault Can Be Drained On Specific Chain

Category	Severity	Location	Status
Protocol Manipulation	Medium	Global	Acknowledged

Description

Because a withdrawal message can be sent to a Vault on any supported chain, a malicious user can deposit in a Vault on one chain and withdraw from a vault on another chain. This can be used to drain the Vault on a chain, forcing other users to withdraw their liquidity on other chains.

Recommendation

Consider restricting what Vault a user can withdraw from, such as only allowing withdrawals from the chain they performed a deposit.

Resolution

Orderly Team: Will fix in the next version. Rebalance (with CCIP) is considered to solve this issue.

LGR-3 | User Can Withdraw Entire Collateral When Open Position

Category	Severity	Location	Status
Logical Error	Medium	Ledger.sol	Acknowledged

Description PoC

Before creating a position, a user is required to deposit a balance sufficient for their position size. Upon the call to executeWithdrawAction, there is validation that the withdraw amount is not greater than the balance.
However, there is no validation at the contract level in the executeWithdrawAction function that restricts a user from withdrawing their entire balance once they have an open position.

If a user's position is in loss, the user could simply withdraw their entire collateral and not risk losing any of their balance during settlement. This terribly disrupts the operations of the protocol as funds won't be available to pay profitable traders.

Recommendation

Before allowing a withdrawal, consider validating that the position will not be in a liquidatable state on chain. This may require passing a price with the WithdrawData.

Resolution

Orderly Team: Won't fix now. maybe in the future version.

The restrict of withdraw is controlled by engine team, so the contract will not lose balance.

In order to restrict withdrawal by contract, a lot of work need take into consideration with a big update:

1. align price oracle for liquidation, for both contract and engine

2. liquidation should be triggered before withdrawals, for prevent bad debts

3. as the suggestion, add a price with the WithdrawData

VAULTM-1 | Funds Will Be Locked On Hard Fork

Category	Severity	Location	Status
Hard Fork	Medium	VaultManager.sol: 16, 18	Acknowledged

Description

In the `VaultManager` contract, token balances are stored in mappings such as `tokenBalanceOnchain` or `tokenFrozenBalanceOnchain`. These mappings take a token hash and a `chainID` and point to a token balance.

If a chain were to experience a hard fork that `chainID` will change. This will result in the token balance being inaccessible as there will be a difference between what is being stored in state and the actual `chainID` of the chain.

Recommendation

Add an `onlyOwner` restricted function that will allow the protocol to migrate balances from the old `chainID` to the new `chainID`.

Resolution

Orderly Team: Won't fix. Will add migrate method only this really happens.

1. When a hard fork happens, the legitimate chain should remain the same ChainID, and we should follow this chain.

2. Even the ChainID changes in HF, we can upgrade the contract to support migrate.

LGR-4 | Large Withdrawals Can Fail

Category	Severity	Location	Status
Logical Error	Medium	Ledger.sol: 264	Acknowledged

Description

In the executeWithdrawAction function, there is a check to ensure that the withdraw.fee is less than the maxWithdrawFee. However the maxWithdrawFee is a fixed value that will be used for all withdrawals. While the withdraw.fee will be a percentage based on the size of the individual withdraw.

Because the two are inherently misaligned, there is a risk that large withdraws will fail as their withdraw.fee will be greater than the maxWithdrawFee. Users in this scenario will need to break up their withdrawals into multiple smaller withdrawals leading to operational inefficiency.

Recommendation

Make maxFee percentage based so that the fee is never more then X percentage of the amount being withdrawn. This will ensure that all valid withdrawals are still possible.

Resolution

Orderly Team: Won't fix.

1. In design, maxWithdrawFee should be a net value, not a percentage value.

2. maxWithdrawFee will be only used in emergency case.

SIG-1 | Signature Malleability

Category	Severity	Location	Status
Signatures	Medium	Signature.sol: 47	Resolved

Description

The `verify` function uses `ecrecover` without any validation that the `s` value is from one half of the valid `s` range, therefore it is possible for signatures to be maliciously replayed with a different `s`.
At present, the `OperatorManager` is the only address that can perform this signature malleability, however, the opportunity should be removed.

Recommendation

Use the OpenZeppelin ECDSA library, which automatically restricts the valid s range, to verify signatures.

Resolution

Orderly Team: Fixed.

MKTM-1 | Incorrect updatedAt Assigned

Category	Severity	Location	Status
Logical Error	Medium	MarketManager.sol: 26, 38	Resolved

Description

In the `updateMarketUpload` functions for both the perp prices and `sumUnitaryFundings` the `lastMarkPriceUpdated` and `lastFundingUpdated` are set to the `block.timestamp`. However, these attributes ought to be set to the `perpPrice.timestamp` and `sumUnitaryFunding.timestamp` as these are the timestamps from which the data was recorded.

Using the `block.timestamp` for the `lastFundingUpdated` and `lastMarkPriceUpdated` values is not in line with the logic in the `Ledger.executeProcessValidatedFutures` function where the `lastFundingUpdated` is assigned to the `trade.timestamp` rather than the `block.timestamp`.

Recommendation

Replace the `cfg.setLastFundingUpdated(block.timestamp)` lines with `cfg.setLastFundingUpdated(data.timestamp)`.

Resolution

Orderly Team: Fixed.

LGR-5 | Potential For Trapped Deposits

Category	Severity	Location	Status
Trapped Funds	Medium	Ledger.sol: 176	Acknowledged

Description

In the event that the `accountDeposit` function execution reverts, there is no recourse for the user to recover their deposit in the vault.

The `accountDeposit` function may revert if the `AccountDeposit` data carries a `brokerHash` or `tokenHash` & `srcChainId` that does not agree with the configuration in the `vaultManager` contract.

Recommendation

Consider implementing a method for the user to recover their funds in the event that the cross-chain deposit transaction cannot succeed even upon retry.

Resolution

Orderly Team: Won't fix, but should pay attention:

1. Ledger should update whitelist before vault update.

2. Even the situation occurs, the cc tx will be payload-store, and can be retried after updating Ledger's whitelist.

LGR-6 | Insurance Account May Become Insolvent

Category	Severity	Location	Status
Logical Error	Medium	Ledger.sol	Acknowledged

Description

When liquidatable accounts cannot cover the `liquidatorFee` with their remaining margin, all positions in the account and the remaining margin balance are transferred to the insurance fund.

In times of volatility, several accounts may become insolvent and all have their positions transferred to the insurance account. The insurance account may then find itself to be insolvent, in which case ADL will not be sufficient to remedy the situation.

The insurance account is also intended to cover insolvent accounts where the settled PnL is more negative than the account margin. This behavior can also be a pathway for the insurance account to become insolvent, especially when combined with receiving positions from insolvent accounts.

Recommendation

Though this scenario may be rare, it is a distinct possibility. Have a contingency plan in the event that this scenario ever plays out, for example a deposit can automatically be made for the insurance fund when it nears insolvency.

Resolution

Orderly Team: Yes this is a risk; but this risk is also present in other CEXes Reply from our product.

GLOBAL-5 | Liquidations When Deposits Paused

Category	Severity	Location	Status
Logical Error	Medium	Global	Acknowledged

Description

When the Vault is paused, functions `deposit` and `depositTo` are prevented from being processed. As a result, the protocol can reach a state where liquidations can occur while deposits are paused, preventing users from keeping their positions solvent.

Recommendation

Consider pausing liquidations when users are unable to increase their collateral and/or clearly document this scenario for users.

Resolution

Orderly Team: Won't fix.

1. Should only pause Vault in emergency case, that is, Vault is hacked, or under HF, or unstable state.

2. Even one chain may be paused, other vault is still working. 3. So no need to pause liquidation in unstable situation. The engine team will take more actions.

GLOBAL-6 | Lacking Storage Gaps

Category	Severity	Location	Status
Storage Gaps	Medium	Global	Acknowledged

Description

There are several Component abstract contracts that intend to be parents of upgradeable contracts, however they lack an appropriate `_gap` storage variable.

For example, the `LedgerComponent` contract is an abstract contract that is meant to be inherited by upgradeable contracts, however there is a `ledgerAddress` storage variable defined in the `LedgerComponent` followed by no gap variable in the event that more variables would be added to the `LedgerComponent` contract.

Recommendation

Add a storage `_gap` variable so that storage variables may be added to the `LedgerComponent` contract without causing storage collisions.

For more information on the `_gap` variable refer to the `OpenZeppelin` documentation.

Resolution

Orderly Team: Wont fix, but will continously pay attention to this issue.

1. LedgerComponent is an abstract contract, very simple, and will not add any new fields.

2. Contract Ledger and OperatorManager is already use a DataLayout contract with _gap

3. Namespaced Storage is better than _gap method, we are still investigating https://blog.openzeppelin.com/introducing-openzeppelin-contracts-5.0#Namespaced

OPMAN-1 | Risk Of DoS

Category	Severity	Location	Status
DoS	Medium	OperatorManager.sol: 133, 153	Acknowledged

Description

In the `_futuresTradeUploadData` function, a batch of trades are uploaded in a single transaction which allows a single malicious trade to DoS the entire batch if it reverts.

For example, one trade in the batch could have an invalid `symbolHash`, which would cause the `executeProcessValidatedFutures` execution to revert. Similarly, in the `_eventUploadData` function, a batch of events are uploaded to be processed in a single tx which allows a single invalid event to DoS the entire batch.

Recommendation

Be aware of this DoS risk and be sure to simulate batches to verify that they contain no invalid trades before sending a batch upload transaction.
Additionally, consider implementing logic at the contract to handle invalid events or trades.

Resolution

Orderly Team: Won't fix, acknowledged. These two function are triggered by engine team, with onlyOperator modifier. In design, there should be no invalid call. We also add many checks to avoid malicious call(signature check, whitelist symbolHash check, etc).

ATPH-4 | DoS On Average Entry Price

Category	Severity	Location	Status
DoS	Medium	AccountTypePositionHelper.sol: 57	Acknowledged

Description PoC

When a position is updated after a trade, the average entry price is calculated using `calAverageEntryPrice`.

However, if the `openingCost` and `currentHolding` are of the same sign, the calculation of `position.averageEntryPrice = halfDown16_8(-openingCost, currentHolding).toUint128()` will revert with a `SafeCastOverflow`. This is because `halfDown16_8(-openingCost, currentHolding)` will return a negative `int` which cannot be cast to a `uint`.

This can occur if the `currentHolding` exceeds the `openingCost` for a short position, because `halfDown16_8` will round the quotient to -1 on `else { quotient -= 1; };` Note that if `halfDown16_8` is modified to round up on a 0 quotient, the issue can still occur if `currentHolding` exceeds the `openingCost` for a long position, because `halfDown16_8` will round the quotient to 1 and the `currentHolding` is also positive.

Recommendation

Carefully consider which markets are supported, as markets with small prices are more susceptible to this issue. Furthermore, consider implementing a minimum trade size since the attack is more susceptible to smaller quantities causing the `openingCost` to round down to -1.

Resolution

Orderly Team: Acknowledged. In design, this will not happen. halfDown16_8 is only a stateless function, and in this situation, the return value should never be negative.

GLOBAL-7 | Unnecessary Timestamp Emitted

Category	Severity	Location	Status
Superfluous Code	Low	Global	Resolved

Description

Throughout the codebase the `block.timestamp` is often emitted with events, however this is unnecessary as the timestamp of the event can be retrieved from the block in which it was emitted. Therefore gas does not need to be expended to emit the `block.timestamp` as a part of event data.

Recommendation

Remove the `block.timestamp` from each event and retrieve the timestamp from the block in which the event was emitted.

Resolution

Orderly Team: The recommendation was implemented in commit `31d480d2`.

PRPT-1 | Unused Side Attribute

Category	Severity	Location	Status
Superfluous Code	Low	PerpTypes.sol: 29	Acknowledged

Description

The `side` boolean on the `FuturesTradeUpload` struct is unused in the contracts.

Recommendation

Implement a use case for the side attribute or consider removing it.

Resolution

Orderly Team: Won't fix now, the usage of this is for signature verification.

ETYP-1 | Unused liquidationTransferId attribute

Category	Severity	Location	Status
Superfluous Code	Low	EventTypes.sol: 87	Acknowledged

Description

The `liquidationTransferId` is never accessed from the `liquidationTransfer` object.

Recommendation

Implement a use-case for the `liquidationTransferId` or remove it from the `LiquidationTransfer` struct.

Resolution

Orderly Team: Won't fix now, the usage of this is for signature verification.

SFCH-1 | Inaccurate Overflow Error

Category	Severity	Location	Status
Errors	Low	SafeCastHelper.sol: 18	Resolved

Description

In the `toUint128` function if the provided `int128` value is negative, the function reverts with a `SafeCastOverflow` error. However the function should revert with a `SafeCastUnderflow` error as the attempted casting would underflow rather than overflow.

Recommendation

Create a `SafeCastUnderflow` error and use this error to revert the `toUint128` function in the event that a negative number is provided.

Resolution

Orderly Team: Fixed.

GLOBAL-8 | Multiple Sources Of Truth

Category	Severity	Location	Status
Warning	Low	Global	Acknowledged

Description

The `Vault` contract and the `VaultManager` contract both track which tokens are valid for the vault, therefore there are two sources of truth for which tokens are supported by any given vault.

However there is no guarantee that the `allowedTokenSet` in the Vault contract and the `allowedChainToken` mapping in the `VaultManager` contract are in sync.

If the `allowedTokenSet` and `allowedChainToken` mapping are ever in disagreement the system accounting is perturbed and the system is may become insolvent.

Recommendation

Be aware of this design flaw and risk when updating configuration on a vault chain or the ledger chain.

Resolution

Orderly Team: Acknowledged

1. Team should keep consistent of vualt and ledger chain's allowedList.

2. And Ledger should update whitelist before vault update as this issue commented: https://www.notion.so/guardianaudits/aa917beacb674d2c9dd7cb0978c2da15?v=5653ae2c360b4404971a6bd0f4ba0f08&p=50b425edf5af4caba8ffce2e0771c6ca&pm=s

ATPH-5 | MaintenanceMargin Does Not Match Documentation

Category	Severity	Location	Status
Documentation	Low	AccountTypePositionHelper.sol: 35	Acknowledged

Description

The unused `maintenanceMargin` function in the `AccountTypePositionHelper` calculates the `positionQty * markPrice * Base MMR`.

`MMR i = Max(Base MMR i, Base MMR i / Base IMR i * IMR Factor i * Abs(Position Notional i)^(4/5))`

Recommendation

Either update the documentation or update the implementation of the `maintenanceMargin` function.

Resolution

Orderly Team: Acknowledged.

ATPH-6 | Unsafe Casting

Category	Severity	Location	Status
Casting	Low	AccountTypePositionHelper.sol.sol: 138	Resolved

Description

The quotient returned in `halfUp16_8_i256` is cast to an `int128` from an `int256`. This casting operation is unsafe as the result can silently overflow.

For example, when the `quotient = type(int256).max`, casting to `int128` will not revert and return -1 which is an unexpected result.

Recommendation

Use the OpenZeppelin `SafeCast` library or implement your own checks to validate the range of a type is not exceeded prior to casting.

Resolution

Orderly Team: Fixed.

VAULT-2 | Missing Check-Effect-Interact Pattern

Category	Severity	Location	Status
Reentrancy	Low	Vault.sol: 114,132,149	Resolved

Description

In the `withdraw` function the token amount is transferred to the user before calling `withdraw` on the `crossChainManagerAddress`.

This could result in reentrancy opportunities, currently there are no immediate risks with the `withdraw` function. However, best practice is to follow the Check-Effects-Interactions pattern when transferring out tokens to protect against reentrancy attacks.

Recommendation

Use the Check-Effects-Interactions pattern by transferring tokens out of the vault after state changes occur.

Resolution

Orderly Team: Fixed.

LGR-7 | Redundant for-loop

Category	Severity	Location	Status
Optimization	Low	Ledger.sol: 354-356	Resolved

Description

In the `executeSettlement` function, the first for-loop is redundant as the `totalSettleAmount` can be computed inside the second for-loop and validated at the end.

Recommendation

Combine the validation logic into a single for-loop in the `executeSettlement` function.

Resolution

Orderly Team: The recommendation was implemented.

ATPH-7 | Typo

Category	Severity	Location	Status
Typo	Low	AccountTypePositionHelper.sol: 19	Resolved

Description

The `accruedFeeUncoverted` variable misspells unconverted as uncoverted.

Recommendation

Replace `accruedFeeUncoverted` with `accruedFeeUnconverted`.

Resolution

Orderly Team: The recommendation was implemented.

LGR-8 | Unused Helper Functions

Category	Severity	Location	Status
Superfluous Code	Low	Ledger.sol: 374, 375	Resolved

Description

The balances adjustment in the `executeSettlement` function can use `subBalance` and the `addBalance` helper functions rather than adjusting the balances mapping directly.

Recommendation

Use the `subBalance` and `addBalance` functions to adjust the balances mapping in the `executeSettlement` function.

Resolution

Orderly Team: The recommendation was implemented.

CCRU-1 | Hardcoded Zero Address

Category	Severity	Location	Status
Hardcoded Value	Low	CrossChainRelayUpgradeable.sol: 203	Acknowledged

Description

In the `sendMessage` function `_lzSend` is called. One of the parameters in the `_lzSend` function is `_zroPaymentAddress`.

According to Layer Zero integration recommendations the `_zroPaymentAddress` should not be hardcoded. Instead it should be passed as a parameter instead.

Recommendation

Pass the `_zroPaymentAddress` as a parameter instead of hardcoding it.

Resolution

Orderly Team: Acknowledged.

ATPH-8 | Multiplication On The Result Of Division

Category	Severity	Location	Status
Precision	Low	AccountTypePositionHelper.sol: 35	Resolved

Description

The `maintenanceMargin` function performs a multiplication on the result of a division, leading to precision loss in the final maintenance margin requirements.

The `position.positionQty.abs().toInt128() * markPrice` calculation will return a 16 decimal result because both quantity and price are 8 decimal precision values.

Further multiplying by the `baseMaintenanceMargin` in the numerator has virtually no overflow risk as the max value of the `baseMaintenanceMargin` is 10,000.

Recommendation

Perform the multiplication before the division:

`position.positionQty.abs().toInt128() * markPrice * baseMaintenanceMargin / (int128(MARGIN_100PERCENT) * PRICE_QTY_MOVE_RIGHT_PRECISIONS)`

Resolution

Orderly Team: The recommendation was implemented.

VCCMU-1 | Excess Fee Locked in Contract

Category	Severity	Location	Status
Trapped Funds	Low	VaultCrossChainManagerUpgradeable.sol: 164	Resolved

Description

In the `depositWithFee` function a deposit can be made with a fee attached. The fee is determined by the `amount` parameter, and `depositWithFee` checks that the `msg.value` is greater than or equal to the `amount` being passed in.

When the fee is sent in `sendMessageWithFee` the value is the original `amount` that was passed into the function. The issue is that if `msg.value` is greater then `amount` the excess `msg.value` will be left in the contract with no way of retrieving it.

Currently the function is not accessible so this possesses no immediate risk. But the issue should be fixed if there are intentions of using this function.

Recommendation

Either send the excess `msg.value` back to `msg.sender` or to an address that can handle the funds so that they are not stuck.

Resolution

Orderly Team: Fixed, amount is removed and msg.value is taken as a fee.

LGR-9 | Excess Insurance Fund Transfer Protection

Category	Severity	Location	Status
Validation	Low	Leger.sol	Acknowledged

Description

The following validation ensures that any transferred insurance amount is sufficient to cover any negative collateral for an account:

```
if (
    balance.toInt128() + settlement.insuranceTransferAmount.toInt128() + settlement.settledAmount < 0
        || settlement.insuranceTransferAmount > settlement.settledAmount.abs()
) {
    revert InsuranceTransferAmountInvalid(
        balance, settlement.insuranceTransferAmount, settlement.settledAmount
    );
}
```

However the validation allows for a potentially significant amount of extra funds from the insurance account to be transferred to the user's account.

Recommendation

Consider altering the validation such that if an `insuranceTransferAmount` is specified, it must be exactly the amount necessary to make the account solvent, or within a smaller range of an amount that would make the account solvent.

Resolution

Orderly Team: Acknowledged.

LGR-10 | Position Never Cleared

Category	Severity	Location	Status
Optimization	Low	Ledger.sol: 104	Resolved

Description

Upon liquidating a position, `liquidatedPosition.isFullSettled()` is called to check whether the cost and quantity of the position are both 0, and clears the position if so.

However, `isFullSettled` is never checked upon `executeSettlement` nor `executeAdl`.

Recommendation

When positions are cleared in the `executeSettlement` or `executeAdl` functions, clear them when `isFullSettled()` is `true`.

Resolution

Orderly Team: The recommendation was implemented for liquidations, however ADL should rarely need to clear positions.

CCRU-2 | Missing Address 0 Check

Category	Severity	Location	Status
Validation	Low	CrossChainRelayUpgradeable.sol: 65	Acknowledged

Description

In the `CrossChainRelayUpgradeable` contract, the `initialize` function accepts an `_endpoint` address, yet fails to validate that it is not `address(0)`.

Recommendation

Validate that the `_endpoint` address is not `address(0)` in the `initialize` function to avoid improper deployments.

Resolution

Orderly Team: Endpoint can be updated later using function updateEndpoint(address _endpoint) external onlyOwner. Even if endpoint address is not address(0) , it could be another wrong address. Both situations are handled by later calling updateEndpoint.

GLOBAL-9 | Use LayerZero Package

Category	Severity	Location	Status
Maintainability	Low	Global	Acknowledged

Description

The Layer Zero documentation recommends that projects use the latest version of the `solidity-examples` package, rather than directly copying example contracts.

The `solidity-examples` package is not used in the `evm-cross-chain` repository, and therefore if a patch is ever issued it would not be present in `evm-cross-chain`.

Recommendation

Use the solidity-examples package as suggested in the LayerZero docs.

Resolution

Orderly Team: In the future we may have custom requirements, so we will choose to maintain this code ourselves.

UTIL-1 | Unnecessary Util Functions

Category	Severity	Location	Status
Superfluous Code	Low	Utils.sol: 11-17	Resolved

Description

There is no need to have both the `getBrokerHash` and `getTokenHash` functions since their logic is exactly the same, only the naming of the parameters differ.

The `getBrokerHash` and `getTokenHash` functions simply return the result of `calculateStringHash`.

Recommendation

Use the `calculateStringHash` function directly.

Resolution

Orderly Team: The recommendation was implemented.

Disclaimer

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

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

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

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

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

About Guardian Audits

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

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

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

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