GUARDIAN

Ethereal

Orderbook Dex

Security Assessment

May 20th, 2024

Summary

Audit Firm Guardian

Prepared By Owen Thurm, Parker Stevens, Mark Jonathas,
Kiki, Osman Ozdemir, Vladimir Zotov, Izuman

Client Firm Ethereal

Final Report Date May 20, 2025

Audit Summary

Ethereal engaged Guardian to review the security of their Orderbook Perps settlement contracts. From the 12th of February to the 24th of February, a team of 7 auditors reviewed the source code in scope. All findings have been recorded in the following report.

For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

🔗 Blockchain network: **Converge**

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

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/ethereal-fuzzing

Table of Contents

Project Overview

Project Summary

Project Name	Ethereal
Language	Solidity
Codebase	https://github.com/meridianxyz/contracts
Commit(s)	Initial commit: 17f30f9221b0b52f3717adb19419d7e80c9372fe Final commit: ce077be857fb1e259f64d6c16540903a71994ccd

Audit Summary

Delivery Date	May 15, 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	1	0	0	0	0	1
● Medium	3	0	0	0	0	3
● Low	16	0	0	9	0	7
● Info	15	0	0	8	0	7

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	● Critical	● High	● Medium
Likelihood: *Medium*	● High	● Medium	● Low
Likelihood: *Low*	● Medium	● Low	● Low

Impact

High	Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.
Medium	A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.
Low	Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High	The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.
Medium	An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.
Low	Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

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

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

The auditing process pays special attention to the following considerations:

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

Invariants Assessed

During Guardian’s review of Ethereal, fuzz-testing was performed on the protocol’s main functionalities. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 10,000,000+ runs with a prepared fuzzing suite.

ID	Description	Tested	Passed	Remediation	Run Count
MATCH-01	In CLOSE_ONLY mode, position sizes can only decrease or stay the same	✅	✅	✅	10M+
MATCH-02	ProductStatusViolation error should not occur when reducing position in CLOSE_ONLY mode	✅	✅	✅	10M+
MATCH-03	Total position size should be 0	✅	✅	✅	10M+
MATCH-04	No-position trader balance must be freely withdrawable	✅	✅	✅	10M+
MATCH-05	Long OI is always the same as Short OI.	✅	✅	✅	10M+
MATCH-06	Sum of long position sizes is equal to the product tracked OI.	✅	✅	✅	10M+
FEE-01	Protocol should be globally solvent	✅	✅	✅	10M+
LIQUI-01	Liquidations should never unexpectedly revert	✅	✅	✅	10M+
ERR-01	Unexpected Error	✅	✅	✅	10M+

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	Insolvency Due To Trading Fees	Logical Error	● Critical	Resolved
H-01	Traders Credited With Infinite Liquidity	Logical Error	● High	Resolved
M-01	excludedAccounts Can Open Trades	Validation	● Medium	Resolved
M-02	Liquidator Can Open Trades For Any Account	Centralization	● Medium	Resolved
M-03	DoS Of processActions()	DoS	● Medium	Resolved
L-01	Impossible To Remove Some Tokens	Unexpected Behavior	● Low	Acknowledged
L-02	tokensByAddress Written For Virtual Tokens	Logical Error	● Low	Resolved
L-03	Lacking Maker Taker Fee Validation	Validation	● Low	Resolved
L-04	UpdateFunding Risks	Centralization	● Low	Acknowledged
L-05	Lacking Exclusion Validation	Validation	● Low	Resolved
L-06	updateMaxLeverage Can Change Liquidation Status	Validation	● Low	Acknowledged
L-07	Healthy Positions Can Be Liquidated	Logical Error	● Low	Resolved
L-08	Missing On Chain Cancellation Logic	Validation	● Low	Resolved
L-09	Unclaimed Fees Are Locked After Update	Logical Error	● Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-10	Changing Lot Size Can Impact Order Fulfillment	Warning	● Low	Resolved
L-11	Incorrect Typehash With Liquidation Orders	Compatibility	● Low	Acknowledged
L-12	Updating Lockout Affects Pending Withdrawals	Unexpected Behavior	● Low	Acknowledged
L-13	maxLeverage Not Validated	Validation	● Low	Acknowledged
L-14	Funds Permanently Stuck With Failed ERC20 Transfer	DoS	● Low	Resolved
L-15	maxOpenInterest Not Verified	Validation	● Low	Acknowledged
L-16	Market Orders Have No On-Chain Price Protection	Validation	● Low	Acknowledged
I-01	Misleading Error	Validation	● Info	Resolved
I-02	Incorrect Error Used For Excluded Accounts	Errors	● Info	Resolved
I-03	Trade Digest Parameters Should Match	Code Quality	● Info	Acknowledged
I-04	Unused Errors	Errors	● Info	Resolved
I-05	Typos	Typo	● Info	Resolved
I-06	FeeCollector’s Unclaimed Fees Are Counted Toward depositCap	Logical Error	● Info	Acknowledged
I-07	Incorrect Comment For Open Interest Calculation	Logical Error	● Info	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
I-08	Lacking Error Data	Errors	● Info	Acknowledged
I-09	Gas Savings Reading Constant Storage Slots	Optimization	● Info	Acknowledged
I-10	decimals() Not Required For ERC20 Standard	Compatibility	● Info	Acknowledged
I-11	Lacking Event Info	Events	● Info	Resolved
I-12	Incorrect Comment	Documentation	● Info	Resolved
I-13	Protocol Cannot Handle Bad Debt	Logical Error	● Info	Acknowledged
I-14	Users Are Never Charged For Gas	Warning	● Info	Acknowledged
I-15	minDeposit Circumvented	Validation	● Info	Acknowledged

C-01 | Insolvency Due To Trading Fees

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

Description

When matching orders, taker and maker fees are not deducted from traders' balances, yet they are still added to the fee collector's balance.

However, when the fee collector claims these fees, both the global balance and the fee collector's personal balance are reduced. This results in insolvency, preventing depositors from withdrawing their balances.

Recommendation

Deduct maker and taker fees from traders' balances.

Resolution

Ethereal Team: The issue was resolved in PR#94.

H-01 | Traders Credited With Infinite Liquidity

Category	Severity	Location	Status
Logical Error	● High	PerpEngine.sol: 396-405	Resolved

Description

In the protocol, PNL is calculated based on the whole position size regardless of the actual filled amount. This means that even if there aren't enough counterparties to fill the full position at a given price, the system still accounts for the profit as if the whole size was successfully exited, which creates a mismatch between realized profits and what is actually achievable in the market.

This creates an issue especially when the liquidity depth is not enough to close a large position. For example:
• Trader opens a 100-size long at $1000
• The price rises to $1500, but it's a resistance level with strong sell pressure and limited buy liquidity.
• Trader closes only size of 1.
• The protocol realizes profit on the full 100-size position at $1500, even though only 1 unit was actually sold.
• Realized profit: $50,000

Normally, equilibrium would be reached as the price declines, with the trader closing the remaining 99-size position at a lower price and realizing a loss. However, knowing they cannot exit the full size at reasonable prices, the trader can withdraw these realized but unbacked profits and leave the remaining position for liquidation.

For this position above (at 10x leverage):
• Initial position value: $100,000
• Deposit: $10,000
• Realized profit: $50,000
• New balance: $60,000
• New position value: 99 * 1500 = $148,500
• New required margin: $14,850
• Trader can withdraw up to $45,150, and leave the rest of the position for liquidation.

Recommendation

Consider realizing PNL only when decreasing or closing positions and calculating it based on the filled amount rather than the entire position size. This ensures that realized profits accurately reflect actual market execution.

When increasing positions, calculate and adjust the average entry price instead of realizing PNL and setting the latest match price as the new entry. This ensures that unrealized gains or losses remain properly accounted for until an actual position reduction occurs.

Resolution

Ethereal Team: The issue was resolved in PR#114.

M-01 | excludedAccounts Can Open Trades

Category	Severity	Location	Status
Validation	● Medium	PerpEngine: _verifyOrder	Resolved

Description

Inside the the _verifyOrder function the order of operations is incorrect. It checks if the order.sender is an excluded account.

However, at this point the order.sender could be the linked signer of the account. In these situations the validation would not be sufficient and an excluded account could open a trade.

Recommendation

Perform the validation with account.sender instead of order.sender.

Resolution

Ethereal Team: The issue was resolved in PR#90.

M-02 | Liquidator Can Open Trades For Any Account

Category	Severity	Location	Status
Centralization	● Medium	ExchangeGateway: _handleLiquidationMatchOrders	Resolved

Description

The liquidator account has the power to open trade positions for any account. The liquidator can increase or decrease another account's position size without their approval or signature.

Recommendation

Similar to how when a market is in a closed status require that `_isIncreasingPosition` returns false. This will limit the liquidator to only decreasing the position's size.

Resolution

Ethereal Team: The issue was resolved in PR#files.

M-03 | DoS Of processActions()

Category	Severity	Location	Status
DoS	● Medium	ExchangeGateway: 404	Resolved

Description

A user can temporarily cause a DoS of the processActions() function by submitting a link signer request and depositing. Doing so will prevent other transactions that are meant to be submitted from being posted on-chain.

In turn, this can delay liquidations and users from closing trades at crucial times. To carry this attack out a malicious user needs to create a link signer action. Then, deposit() can be called by the user directly. This will add them to the excludedSigners mapping.

Inside of _handleLinkSigner(), a revert will occur when it validates that the user is not in the excludedSigners mapping.

Recommendation

Wrap internal calls inside of processAction() in try-catch blocks to prevent a revert from causing a DoS of the entire transaction.

Resolution

Ethereal Team: The issue was resolved in PR#116.

L-01 | Impossible To Remove Some Tokens

Category	Severity	Location	Status
Unexpected Behavior	● Low	ExchangeConfig.sol: 191-195	Acknowledged

Description

The ExchangeConfig contract allows tokens to be removed and specifically contains logic for removing the USD token used in the contract.

However, it is realistically impossible to remove the USD token as the USD token must be used as the quote token in the PerpEngine.sol contract, thus marking it as removeProtected.

```
// Only allow usdToken (i.e. USDe) as the perp settlement currency. if
(quoteToken.tokenAddress = accountGlobal.usdToken)

    {revert InvalidParameter("quoteTokenName", P_ERR_BAD_ADDR);} if
(quoteToken.removeProtected) {quoteToken.removeProtected = true;}
```

Furthermore, protected tokens in general can never be removed as they are required to be removeProtected = false, but there is not way to assign removeProtected back to false for tokens which are only used in deprecated markets.

Recommendation

Be aware of this behavior and if you wish to be able to remove tokens that are only used in deprecated markets then consider refactoring the removeProtected logic.

Resolution

Ethereal Team: This is expected behaviour and documented in the code. removeToken is largely in-place to undo fat finger configuration when an asset is initially added and needs to be updated before enabling deposits.

L-02 | tokensByAddress Written For Virtual Tokens

Category	Severity	Location	Status
Logical Error	● Low	ExchangeConfig.sol: 73	Resolved

Description

In the addToken function the tokensByAddress mapping is updated even in the case where the tokenAddress is address(0). Therefore the entry for address(0) is a valid token and is the last token that was added with the addToken function.

Recommendation

Only write to the tokensByAddress mapping inside of the tokenAddress = address(0) else case.

Resolution

Ethereal Team: The issue was resolved in PR#105.

L-03 | Lacking Maker Taker Fee Validation

Category	Severity	Location	Status
Validation	● Low	PerpEngine.sol	Resolved

Description

In both the `register` and `updateFees` functions there are no validations performed on the maker and taker fees to ensure that the maker fee is lower than the taker fee and also within a reasonable range.

Recommendation

Consider adding validations in both the `register` and `updateFees` functions so that the maker fee cannot be configured higher than the taker fee and that both fees are within a reasonable range.

Resolution

Ethereal Team: The issue was resolved in PR#106.

L-04 | UpdateFunding Risks

Category	Severity	Location	Status
Centralization	● Low	PerpEngine.sol: 259	Acknowledged

Description

There is no signature validation necessary for the UpdateFunding action from the sequencer. Therefore a compromised or errant sequencer may submit invalid funding updates.

This affords the sequencer the power to apply arbitrarily large funding amounts and subsequently liquidate users or a malicious actor who has compromised the sequencer may assign a high positive funding amount and immediately close their account to withdraw ill gotten gains.

Recommendation

There are already TODO comments which suggest adding caps on the magnitude of the fundingDeltaUsd and limiting how often update funding can be called, which appropriately addresses the risk. Consider implementing these TODO comments.

Additionally, consider adding a specific FundingUpdater signer which must attest to the validity of the funding updates. This way even if the sequencer were compromised they cannot submit any inaccurate funding updates.

Resolution

Ethereal Team: These TODO comments will be implemented at a later time before the initial launch.

L-05 | Lacking Exclusion Validation

Category	Severity	Location	Status
Validation	● Low	ExchangeConfig.sol	Resolved

Description

In the addSequencer, updateFeeCollector, and updateLiquidator functions there is no validation to ensure the added account is not an excludedSigners or excludedAccounts exclusion before adding it.

Recommendation

Consider validating that the newly assigned account is not already set as true in either the excludedSigners or excludedAccounts mappings.

For the updateFeeCollector and updateLiquidator just be sure that the account is not the same as the existing fee collector and liquidator accounts before performing this validation.

Resolution

Ethereal Team: The issue was resolved in PR#files.

L-06 | updateMaxLeverage Can Change Liquidation Status

Category	Severity	Location	Status
Validation	● Low	PerpEngine: updateMaxLeverage	Acknowledged

Description

In the offchain Matching Engine, initialMarginRate is the reciprocal of maxLeverage. maintenanceMarginRate is half of initialMarginRate. If an account fails to satisfy the maintenance margin they will be liquidated. Therefore, if the maxLeverage for a product is decreased the maintenanceMargin will increase. Accounts with open positions may be moved under the maintenanceMargin and into liquidation zone. Consider the following example:

STATE 1
initialMaxLeverage = 10
initialMarginRate = 1/(initialMaxLeverage) = 1/10
maintenanceMarginRate = (1/2)*initialMarginRate = 1/20

STATE 2
newMaxLeverage = 5
newInitialMarginRate = 1/(newMaxLeverage) = 1/5
maintenanceMarginRate = (1/2)*newInitialMarginRate = 1/10
Let's say an account has a positionSize = $10,000 and collateral = $750

In STATE 1
minMaintenanceCollateral = (1/20)*positionSize
minMaintenanceCollateral = (1/20)*$10,000
minMaintenanceCollateral = $500
$750 > $500, SAFE FROM LIQUIDATION

In STATE 2
minMaintenanceCollateral = (1/10)*positionSize
minMaintenanceCollateral = (1/10)*$10,000
minMaintenanceCollateral = $1000
$750 < $1000, LIQUIDATABLE

Therefore, the collateral requirements increased after the maxLeverage was decreased.

Recommendation

First put the product in a closed status. Once all trades are closed, update the max leverage and then change the status of the product to open.

Resolution

Ethereal Team: Acknowledged.

L-07 | Healthy Positions Can Be Liquidated

Category	Severity	Location	Status
Logical Error	● Low	PerpEngine.sol: executeLiquidationMatch	Resolved

Description

Even with sequencer checks if the user makes a deposit right before a batch of orders goes through the sequencer could check and see that the account is underwater.

But by the time liquidations go through the user could have deposited funds and had a healthy account, but still get liquidated since there are no on chain margin checks.

Recommendation

Document to users that if the account is at any point in time liquidatable they are eligible to be liquidated and pay the liquidation fee. Regardless of the accounts health when the actual liquidation occurs.

Resolution

Ethereal Team: The issue was resolved in PR#110.

L-08 | Missing On Chain Cancellation Logic

Category	Severity	Location	Status
Validation	● Low	ExchangeGateway.sol	Resolved

Description

When a cancellation occurs it is intended to be handled off-chain. But the signatures will still be valid on chain. This means that even after a user cancels an order that signature can still be used on chain.

Even if this information is maintained off chain all it would take is the off chain data to be lost, overlooked, or changed, at which point all cancelled orders could be used despite the original sender's request to cancel the order.

The sequencer is trusted to not re-use these signatures/nonces and the off-chain component is trusted to not allow re-submissions of these signatures/nonces.

Recommendation

Ensure that the off-chain systems cannot allow nonces to be re-submitted when they haven't been used on-chain. Optionally, consider adding a deadline parameter to the signature so that naturally after a set amount of time the order cannot be used.

Resolution

Ethereal Team: The issue was resolved in PR#118.

L-09 | Unclaimed Fees Are Locked After Update

Category	Severity	Location	Status
Logical Error	● Low	ExchangeConfig.sol	Acknowledged

Description

Fee collectors are excluded accounts and claim fees using the `claimFees` function. The owner can change the fee collector address, but this change does not perform a balance check.

Setting a new fee collector without claiming the previous balance will lock the previously earned fees. This happens because the previous fee collector can no longer call the `claimFees` function.

Additionally, they cannot use the withdraw function due to their excluded status. The new fee collector also cannot withdraw those fees, as they remain in the previous collector's balance.

Recommendation

Perform a balance check and claim earned fees before updating the fee collector address. Otherwise be aware of this behavior and ensure that fees are always claimed before changing the fee collector or that the fee collector address should be reset to the original address to claim any fees that went unclaimed.

Resolution

Ethereal Team: Acknowledged.

L-10 | Changing Lot Size Can Impact Order Fulfillment

Category	Severity	Location	Status
Warning	● Low	PerpEngine: _verifyOrder	Resolved

Description

Whenever the a trade is opened it checks to make sure the quantity product.lotSize = 0. Thus, if the owner makes updates the product.lotSize, there is a good chance that open positions will not be able to be 100% liquidate-able. For example, the original lotSize = 2.

A trade is made with quantity of 4 and passes all checks. Then the lostSize is changed to 5 and the position cannot be liquidated because 4 * 5 = 0. In reality the amount would be small, but there is a very high chance this will happen if the lotSize is ever updated.

Recommendation

Validate that the existing order fulfillment is not impacted by lot size changes prior to changing the lot size.

Resolution

Ethereal Team: The issue was resolved in PR#104.

L-11 | Incorrect Typehash With Liquidation Orders

Category	Severity	Location	Status
Compatibility	● Low	ExchangeGateway.sol: 134	Acknowledged

Description

The LiquidateTradeOrder struct includes the TraderOrder struct, along with liquidator and liquidatorSubaccount fields. The type hash for this struct is created in the code as follows:

“LiquidateTradeOrder(address sender, bytes32 subaccount, uint128 quantity, uint128 price, uint8 side, uint8, engineType, uint32 productId, uint64 nonce,address liquidator, bytes32 liquidatorSubaccount)".

However, according to EIP-712, when encoding structs that contain nested structs, each struct should be encoded separately. For the LiquidateTradeOrder type above, the correct encoding should be:

"LiquidateTraderOrder(TradeOrder order, address liquidator, bytes32 liquidatorSubaccount)TraderOrder(address sender, bytes32 subaccount, uint128 quantity, uint128 price, uint8 side, uint8 engineType, uint32 productId, uint64 nonce)”.

Recommendation

Update the struct encoding according to the EIP standard.

Resolution

Ethereal Team: Although this does not strictly follow the EIP standard, the current encoding still produces a unique hash for each order, just implemented differently.

L-12 | Updating Lockout Affects Pending Withdrawals

Category	Severity	Location	Status
Unexpected Behavior	● Low	ExchangeGateway.sol: 251-254	Acknowledged

Description

Withdrawals are a two-step process in the protocol. Users initiate withdrawals and finalize them after the lockout period has passed.

The lockout period check is performed during the second step of the process in the `finalizeWithdraw` function:

`validAfter = withdraw.initiatedAt + exchange.withdrawLockout`.

However, the lockout period can be changed at any time, and this change not only affects future withdrawals but also pending withdrawals. A withdrawal request should have the `validAfter` time based on the lockout period at the time of the request's creation.

Recommendation

Determine the `validAfter` parameter during the first step of the withdrawal process.

Resolution

Ethereal Team: Acknowledged.

L-13 | maxLeverage Not Validated

Category	Severity	Location	Status
Validation	● Low	Lack Of Present Code	Acknowledged

Description

The maxLeverage value is configured and updatable. However, a trader’s leverage is never validated. This allows traders to open positions that vastly exceed the leverage tolerance for a product.

Recommendation

Validate the trader is not using more leverage than is allowed for the product.

Resolution

Ethereal Team: maxLeverage and other attributes mentioned here, such as maxOpenInterest although not validated onchain, is validated offchain. The reason we don't validate maxLeverage during trades is it can significantly lower settlement throughput and without formal coordination could lead to temporary settlement halts.

L-14 | Funds Permanently Stuck With Failed ERC20 Transfer

Category	Severity	Location	Status
DoS	● Low	ExchangeGateway.sol: 267	Resolved

Description

In the case where there is a withdraw fee, the fee is subtracted from the action.message.amount. It is checked that the message amount is at least as large as the withdrawfee → if (withdrawFee > action.message.amount) { revert…}. However, this presents an edge case where action.message.amount = withdrawFee. In this case, the amount written for the withdraw request will be 0.

However, some tokens revert upon transferring 0-value, causing the call to finalizeTransfer() to revert. Since a user can only have one pending withdrawal at a time, it is imperative that the user can finalize each withdrawal.

Recommendation

The recommended mitigation is two-fold. Firstly, ensure that the withdrawal amount is strictly greater than the withdraw fee action.message.amount > withdrawFee. Second, add some logic to mitigate general ERC20 transfer failures.

While these tokens will be on their own L3, there may be tokens with transfer restrictions like USDC's blacklist. You can potentially check if the transfer fails and add back the token balance to the user and delete the pending withdrawal so that the user can queue another token to be withdrawn.

Resolution

Ethereal Team: The issue was resolved in PR#100.

L-15 | maxOpenInterest Not Verified

Category	Severity	Location	Status
Validation	● Low	PerpEngine: _executeMatch	Acknowledged

Description

The maxOpenInterest is not verified when opening a new position. Unless the trade is verified by the Sequencer the maxOpenInterest can easily be surpassed since there are no checks inside the contract.

Since many orders will occur in a single batch it may be possible that open interest will exceed the max if a single batch is large enough.

Recommendation

Inside _verifyOrder verify the maxOpenInterest has not been reached similar to how they are checking the quantity, lotSize etc.

Resolution

Ethereal Team: Similar to other configuration variables, this is verified offchain and largely in place to provide a central point of exchange configuration. We've provided more docs around config management in the README.

L-16 | Market Orders Have No On-Chain Price Protection

Category	Severity	Location	Status
Validation	● Low	PerpEngine: _executeMatch	Acknowledged

Description

Off-chain market orders will be signed with a `price = 0` and will offer no on-chain price protection for a user. The fulfilled price could be far from what they originally expected.

Recommendation

Consider implementing either on chain or off chain slippage where the user can ensure they are being matched with what they anticipate the market price to be, and if it is not, don't match the order.

Resolution

Ethereal Team: The matching engine has price slippage protection for market orders such that if the slippage exceeds 5% then the order will not be filled.

I-01 | Misleading Error

Category	Severity	Location	Status
Validation	● Info	Errors.sol: 130	Resolved

Description

In the _verifyProductSizes function the maxQuantity lotSize = 0 validation provides the P_ERR_AMT_GT_MAX error. However this error code is not the most accurate for this validation since it is not comparing against a max, but instead a modulus.

Recommendation

Consider using the P_ERR_BAD_VAL error code or introducing a new error code for the maxQuantity lotSize = 0 validation.

Resolution

Ethereal Team: The issue was resolved in PR#108.

I-02 | Incorrect Error Used For Excluded Accounts

Category	Severity	Location	Status
Errors	● Info	ExchangeGateway.sol: 194	Resolved

Description

The code mistakenly uses the UnauthorizedAccount error instead of ExcludedAccount error when checking if the sender is an excluded account.

```
(accountGlobal.excludedAccounts[msg.sender]) {revert
UnauthorizedAccount(msg.sender);}
```

Compare this to a similar check elsewhere in the code:

```
(accountGlobal.excludedAccounts[action.message.account]) {revert
ExcludedAccount(idx, action.message.account);}
```

Recommendation

Use the ExcludedAccount error for both checks.

Resolution

Ethereal Team: The issue was resolved in PR#107.

I-03 | Trade Digest Parameters Should Match

Category	Severity	Location	Status
Code Quality	● Info	Global	Acknowledged

Description

The function _getTradeOrderDigest() orders the parameters for the struct digest slightly differently than the TradeOrder struct.

```
function _getTradeOrderDigest(TradeOrder memory order) private view returns
(bytes32) {bytes32 structDigest =
keccak256(abi.encode(TRADE_ORDER_ABI_PARAM_SIG, order.sender, order.quantity,
order.price, order.side, order.productId, order.engineType, order.subaccount,
order.nonce));
```

vs:

```
struct TradeOrder {address sender; bytes32 subaccount; uint128 quantity;
uint128 price; OrderSide side; EngineType engineType; uint32 productId; uint64
nonce;}
```

Recommendation

Consider ordering the digest entries the same as the TradeOrder struct entries.

Resolution

Ethereal Team: Acknowledged.

I-04 | Unused Errors

Category	Severity	Location	Status
Errors	● Info	ExchangeGateway.sol	Resolved

Description

The ExchangeGateway contract holds several unused errors:

- InvalidParameter
- P_ERR_BAD_ADDR
- SignerNotFound

Recommendation

Consider implementing the use of these errors or removing them.

Resolution

Ethereal Team: The issue was resolved in PR#94.

I-05 | Typos

Category	Severity	Location	Status
Typo	● Info	Global	Resolved

Description

Throughout the contract there are several typos:

- ExchangeGateway.sol:292 Accured → Accrued
- PerpEngine.sol:349 Accured → Accrued
- Architecture Diagram: Sotre -> Store
- ExchangeGateway.sol:244: withou -> without
- ProcessActions.InitiateWithdraw.t.sol:37: Inititate -> Initiate
- ProcessActions.LinkSigner.t.sol:91-93 Lined -> Linked

Recommendation

Consider correcting these typos.

Resolution

Ethereal Team: The issue was resolved in PR#94.

I-06 | FeeCollector’s Unclaimed Fees Are Counted Toward depositCap

Category	Severity	Location	Status
Logical Error	● Info	ExchangeGateway.sol: 176-183	Acknowledged

Description

The depositCap is strictly checked against the global tokenBalance. However, the FeeCollector’s unclaimed fees are contained in the global balance until they claim them.

This is somewhat misleading as the FeeCollector’s fees are already earmarked for them and are not truly a deposit.

Recommendation

When checking the global balance against the deposit cap, deduct the FeeCollector’s balance.

Resolution

Ethereal Team: Expected behaviour. The team will ensure deposit capacity is sufficiently large enough to account for unclaimed fees. Additionally, we will have processes to periodically claim fees on a regular basis.

I-07 | Incorrect Comment For Open Interest Calculation

Category	Severity	Location	Status
Logical Error	● Info	PerpProduct.sol: 31-32	Resolved

Description

The comment about the OI calculation states: Total product OI in native units (e.g. 5 short, 5 long = 10 OI). But the implementation only counts long open interest.

Recommendation

Correct the comment to correctly reflect the OI calculation.

Resolution

Ethereal Team: The issue was resolved in PR#94.

I-08 | Lacking Error Data

Category	Severity	Location	Status
Errors	● Info	PerpEngine.sol: 300	Acknowledged

Description

The _verifyOrder function includes a boolean parameter isTaker which indicates whether it is the maker or taker order which is being verified.

However the errors raised in the _verifyOrder function do not specify the isTaker value and therefore are ambiguous as to which order the revert occurred with.

Recommendation

Consider including the isTaker information in the reverts in the _verifyOrder function to provide more data about the error.

Resolution

Ethereal Team: Acknowledged.

I-09 | Gas Savings Reading Constant Storage Slots

Category	Severity	Location	Status
Optimization	● Info	Global	Acknowledged

Description

There can be gas savings made by changing libraries to read from constant storage slots.

Recommendation

Consider implementing constant storage slots in the libraries used.

Resolution

Ethereal Team: Acknowledged.

I-10 | decimals() Not Required For ERC20 Standard

Category	Severity	Location	Status
Compatibility	● Info	ERC20Helpers: 50	Acknowledged

Description

ERC20Helpers.isERC20() validates if a token can be added by checking if it is an ERC-20 token. One of the checks looks for a return value from the decimals() function. Since this function is not required to be implemented, some ERC-20 tokens may be excluded from the use in the protocol.

Recommendation

If you wish to include ERC-20 tokens that do not implement decimals(), other protocols default the decimal value to 18 in a try/catch block.

Resolution

Ethereal Team: We will only support ERC20 tokens with a decimals() function and assuming 18 decimals can lead to downstream problems if the assumption doesn’t hold true.

I-11 | Lacking Event Info

Category	Severity	Location	Status
Events	● Info	ExchangeGateway.sol: 228	Resolved

Description

The Deposit event does not include details on how much fees were charged during the deposit, nor what the original total deposit amount was before fees.

This information may be useful to off-chain systems which are reading for Deposit events and is in contrast to the WithdrawFinalized event which includes the fee.

Recommendation

Consider emitting the fee amount along with the Deposit function.

Resolution

Ethereal Team: The issue was resolved in PR#110.

I-12 | Incorrect Comment

Category	Severity	Location	Status
Documentation	● Info	ExchangeGateway.sol: 403	Resolved

Description

The comment in the _handleLinkSigner function on line 403 mentions that subaccounts are an excludedSigners address. However it is normal accounts which are excluded, not sub accounts.

Recommendation

Correct the comment to indicate that it is accounts which can be in the excludedSigners mapping and not subaccounts.

Resolution

Ethereal Team: Resolved.

I-13 | Protocol Cannot Handle Bad Debt

Category	Severity	Location	Status
Logical Error	● Info	PerpEngine.sol: 403	Acknowledged

Description

The `_settlePosition` function is invoked for both sides of the trade when matching orders. The function calculates the PnL, determines the new USD balance, and updates positions.

The `usdTokenBalance + pricePnl - fundingPnl` is converted to `uint256` after it is calculated. However, the `toUint256` function reverts when the provided value is negative. As a result, settlements will fail when a trader loses their entire balance.

Since the liquidation process also follows the same execution flow, liquidations will also fail in that case. Even if a partial liquidation is attempted, it would also fail because the PnL is calculated based on the entire position size.

Recommendation

Consider setting the trader's balance to 0 when it falls below zero due to negative PnL. However, this would also require covering the negative amount, either from protocol-owned liquidity (e.g., fees) or from other traders.

Alternatively, set high liquidation thresholds and large margin requirements to ensure this situation never occurs, even during sudden and large price movements.

Resolution

Ethereal Team: There will be a large enough maintenance margin buffer where this will never happen.

Furthermore, If a loss was incurred due to a e.g. MarkChange, the offchain matching engine would cap the loss to prevent an insolvency (i.e. bankruptcy price) where the balance would never go below 0. In short, the sequencer would never trigger a trade where the account would be negative. If it did, it would be a bug and this should revert and halt, which it currently does.

I-14 | Users Are Never Charged For Gas

Category	Severity	Location	Status
Warning	● Info	Global	Resolved

Description

The protocol never charges users for gas, and all gas required to settle on-chain actions must be paid by the sequencer. Even if this is intentional, users can cause the sequencer to consume more gas by frequently linking and revoking signers.

Recommendation

Be aware of this situation. Either charge users gas for on-chain actions or restrict certain actions (especially those that don't require any fees) from being called repeatedly.

Resolution

Ethereal Team: This is resolved by deposit/withdrawal fees, increasing minOrderQty, API rate limits, and rate limits on maximum linked signers per day/week.

.

I-15 | minDeposit Circumvented

Category	Severity	Location	Status
Validation	● Info	ExchangeGateway.sol: 501	Acknowledged

Description

In the _handleInitiateWithdraw function there is no validation to ensure that the remaining deposit is larger than the configured minDeposit for the deposit token.

As a result, users may deposit and then withdraw funds in order to leave a dust amount of collateral in their account. This may cause unexpected issues in the off-chain system if a malicious actor were to create many accounts with a small amount of collateral deposited in each.

This would require the sequencer to hold and validate state for a potentially large number of accounts. In the worst case this could cause the sequencer to OOM at a certain scale, though this case is exceptionally unlikely.

Recommendation

Consider validating that the minDeposit is still met, optionally with a buffer to account for price changes of open positions, in the _handleInitiateWithdraw function.

Resolution

Ethereal Team: We have deposit and withdrawal fees to disincentivize this behavior.

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