GUARDIAN AUDITS

SMART CONTRACT SECURITY AUDIT OF

PariFi

Summary

Audit Firm Guardian

Prepared By Owen Thurm, Daniel Gelfand, 0xKato

Client Firm PariFi Finance

Final Report Date September 3, 2023

Audit Summary

PariFi engaged Guardian to review the security of its decentralized synthetics perpetuals exchange. From the 18th of August to the 1st of September, a team of 3 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Issues Detected Throughout the course of the audit numerous high impact issues were uncovered and promptly remediated by the PariFi team. Several issues impacted the fundamental behavior of the protocol, following their remediation Guardian believes the protocol to uphold the functionality described for the perpetuals exchange.

Code Quality Given the number of high-impact issues detected and the scope of remediations necessary, Guardian supports an independent security review of the protocol at a finalized frozen commit.

🔗 Blockchain network: **Arbitrum**

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

Table of Contents

Project Information

Project Overview 4

Audit Scope & Methodology 5

Smart Contract Risk Assessment

Findings & Resolutions 7

Addendum

Disclaimer 48

About Guardian Audits 49

Project Overview

Project Summary

Project Name	PariFi Finance
Language	Solidity
Codebase	https://github.com/Parifi/parifi-contracts-internal
Commit(s)	6719ad104276eb928938ed2e6c6d2443f50a50f8 457f9f2e247e681e2e183a910d25335cb0473fc7

Audit Summary

Delivery Date	September 3, 2023
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	6	6	0	0	0	0
High	7	7	0	0	0	0
Medium	18	18	0	0	0	0
Low	7	7	0	0	0	0

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
DF	DataFabric.sol	7cd7efa8241bf08a15fdb444054c24060129c838
FM	FeeManager.sol	053c71212be4f00064a18fdb4bb6942edcb6d73f
MKTV	ParifiVault.sol	b549eab723beee764493f07c190e54e46cf0d4f8
ODS	OrderDS.sol	50f12147bb6c7d9c880a9e7aa0d43122945bce51
ORDM	OrderManager.sol	1cdb2f61a9699bdc2eb69f0d35de50af5e872aeb
FWD	PariFiForwarder.sol	1d0abdf2d72e540402c05cd9f5c9ad89ffb6fd81
PF	PriceFeed.sol	d0a7afe2c0caed701b718262d9b092122a7c5cc7
RBAC	RBAC.sol	d20e7a9221317bba3ade35564fdb64c5d4b7e6b6
LE	LibError.sol	d2f0fdb1a193f0aef007a49d71dd6487edd67237
IDF	IDataFabric.sol	1e1284b51c09ac0c99ab31838a16f3e7c2fa5501
IFM	IFeeManager.sol	1ae3e49a5e356b1512a15beab90680b46dd5bf65
IPV	IParifiVault.sol	3950beb79ff885ce722baadbe53b13f3ace0d19c
IORDM	IOrderManager.sol	a9b13486eb7b89e7fe8297d55558a90be1ecde34
IPF	IPriceFeed.sol	6b48f01143d226c57f122025a88fa462ef46791c
IRBAC	IRBAC.sol	9a536452c28452644fce409910bfac18c5d134f4

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
MKTV-1	Users Prevented From Withdrawing Liquidity	Access Control	Critical	Pending
FM-1	Fees Unable To Be Distributed	Logical Error	Critical	Pending
ORDM-1	Users Can Modify Any Position	Access Control	Critical	Pending
ORDM-2	Decreasing Position Size Does Not Account For PnL	Logical Error	Critical	Pending
ORDM-3	Users In Profit Errantly Liquidated	Logical Error	Critical	Pending
ORDM-4	Insolvent Closes Steal Collateral From Other Positions	Logical Error	Critical	Pending
ORDM-5	Incorrect OrderType Used For Execution Price	Logical Error	High	Pending
ORDM-6	Price Updated In Wrong Direction	Logical Error	High	Pending
ORDM-7	Small Positions Prevented From Being Closed	Validation	High	Pending
ORDM-8	Misconfigured Markets Can Break The Protocol	Validation	High	Pending
ORDM-9	updatedAvgPrice Always Rounds Down	Rounding	High	Pending
ORDM-10	Lack Of Reserve Validation	Validation	High	Pending
ORDM-11	Open Interest Validation Becomes Meaningless	Validation	High	Pending

Findings & Resolutions

ID	Title	Category	Severity	Status
FWD-1	Relayer May Censor Transactions	Validation	Medium	Pending
MKTV-2	Maximum Open Interest Configuration Risk	Configuration	Medium	Pending
ORDM-12	Unexpected Limit Execution	Unexpected Behavior	Medium	Pending
ORDM-13	Slippage Applies To Both Sides	Logical Error	Medium	Pending
ORDM-14	maxLeverage Bypassed	Validation	Medium	Pending
ORDM-15	Lacking Validation For New Markets	Validation	Medium	Pending
ORDM-16	Lacking Validation When Updating Markets	Validation	Medium	Pending
MKTV-3	_withdrawalFee Not Validated In Constructor	Validation	Medium	Pending
ORDM-17	Pyth Prices Not Updated Before Being Used	Logical Error	Medium	Pending
ORDM-18	ExecutionFee To Cover The Keeper's Gas	Gas Remuneration	Medium	Pending
ORDM-19	Fees May Not Be Skipped Due To Misconfiguration	Validation	Medium	Pending
ORDM-20	Leverage Relies On EMA	Logical Error	Medium	Pending
ORDM-21	User's Leverage Changes With The Index Price	Unexpected Behavior	Medium	Pending

Findings & Resolutions

ID	Title	Category	Severity	Status
ORDM-22	Fees Based On EMA	Logical Error	Medium	Pending
ORDM-23	Read-only Reentrancy Potential	Reentrancy	Medium	Pending
ORDM-24	Inefficient Limit Order Validation	Validation	Medium	Pending
ORDM-25	Fees Can Be Avoided With Small Order Sizes	Rounding	Medium	Pending
ORDM-26	Risk Of Stale Pricing	Validation	Medium	Pending
MKTV-4	Dead Address Can Be Constant	Constants	Low	Pending
ORDM-27	Unnecessary userPosition Storage Declaration	Optimization	Low	Pending
ORDM-28	Unnecessary userOrder Storage Declaration	Optimization	Low	Pending
ORDM-29	Leverage Rounded Down	Rounding	Low	Pending
ORDM-30	Unnecessary Handling Of 0 Collateral	Optimization	Low	Pending
ORDM-31	Loss Amount Is Rounded Down	Rounding	Low	Pending
ORDM-32	Superfluous orderToPositionId Mapping	Superfluous Code	Low	Pending

MKTV-1 | Users Prevented From Withdrawing Liquidity

Category	Severity	Commit	Location	Status
Access Control	● Critical	6719ad104276eb928938ed2e6c6d2443f50a50f8	MarketVault.sol	Pending

Description

Whenever a user deposits into the `MarketVault`, the `lastDepositedTimestamp[receiver]` is updated to the current block's timestamp. This is then used to ensure depositors have been in the vault for a `MINIMUM_DEPOSIT_PERIOD` when withdrawing or redeeming.

The problem is that a malicious user can deposit 1 wei of assets with another user as the `receiver`, updating the receiver's `lastDepositedTimestamp`. This can be used to prevent users from ever exiting their LP, leading to loss of funds.

Recommendation

Do not allow users to deposit for arbitrary receivers.

Resolution

FM-1 | Fees Unable To Be Distributed

Category	Severity	Commit	Location	Status
Logical Error	● Critical	6719ad104276eb928938ed2e6c6d2443f50a50f8	FeeManager.sol: 83	Pending

Description

The fee distribution interval early return logic is reversed such that fees can only be distributed inside of the `DELAY` window. After the `DELAY` window has passed fees can no longer be distributed.

```
// Distribute fees at regular intervals of every 1 hour
if (lastTransferTimestamp + DELAY < block.timestamp) return;
```

Recommendation

Replace the interval early return logic with: `lastTransferTimestamp + DELAY > block.timestamp`.

Resolution

ORDM-1 | Users Can Modify Any Position

Category	Severity	Commit	Location	Status
Access Control	● Critical	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 552	Pending

Description

An arbitrary `positionId` can be provided to the `modifyPosition` function without validation that the `msg.sender` is the owner. As a result, users may modify or close any arbitrary position, even if it doesn't belong to them.

Recommendation

Validate that the `msg.sender` is the owner of the supplied `positionId`.

Resolution

ORDM-2 | Decreasing Position Size Does Not Account For PnL

Category	Severity	Commit	Location	Status
Logical Error	● Critical	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 339	Pending

Description

In the `_decreasePosition` function users can decrease their position size without realizing any of the positive or negative PnL for their position. This way a user can decrease their size to a trivial amount if they do not immediately start to profit.

Recommendation

When users decrease their position size account for a proportional amount of their current PnL being realized.

Resolution

ORDM-3 | Users In Profit Errantly Liquidated

Category	Severity	Commit	Location	Status
Logical Error	● Critical	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 441	Pending

Description

When the position's PnL is obtained in the liquidate function with `(uint256 pnlInCollateral,) = getProfitOrLossInCollateral(_positionId, executionPrice)`
The protocol assumes that the `pnlInCollateral` is always a loss, `uint256 lossInCollateral = pnlInCollateral + feesInCollateral`.

As a result, a position with a large profit will be treated as if it is in a large loss and can be liquidated.

Recommendation

Require that the `isProfit` returned from the `getProfitOrLossInCollateral` function is `false`.

Resolution

ORDM-4 | Insolvent Closes Steal Collateral From Other Positions

Category	Severity	Commit	Location	Status
Logical Error	● Critical	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 282	Pending

Description

When the `pnlInCollateral` is greater in magnitude than the `collateralAmount` and `isProfit` is `false` the entire `pnlInCollateral` amount is transferred to the `feeManager` and distributed to the protocol and the LPers.

This steals the delta (`|pnlInCollateral| - collateralAmount`) from the collateral of other positions in the `OrderManager`, otherwise if that collateral amount isn't in the `OrderManager` contract from other positions the tx will simply revert with a balance underflow.

Recommendation

In the event where the `pnlInCollateral` is greater than the available collateral and `isProfit` is `false`, only send the available collateral from the position to the `feeManager`.

Resolution

ORDM-5 | Incorrect OrderType Used For Execution Price

Category	Severity	Commit	Location	Status
Logical Error	● High	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 683	Pending

Description

In the `calculateLeverage` function the `executionPrice` is hardcoded to use the `OPEN_NEW_POSITION` order type, however the `calculateLeverage` function is used for other order types such as `DECREASE_POSITION` where the resulting `executionPrice` ought to be using the less favorable price for decreases.

Recommendation

Allow the `calculateLeverage` function to take in an `orderType` and supply the appropriate `orderType` when validating leverage for each `orderType`.

Resolution

ORDM-6 | Price Updated In Wrong Direction

Category	Severity	Commit	Location	Status
Logical Error	● High	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 177	Pending

Description

A potential problem arises when price is updated according to the market's `deviationPoints`. The update only considers whether the user is long or short, if the user opens a long they receive a superior execution.

If the user closes a long position, the user receives a less favorable execution. The liquidity curve is intended to imitate widening or narrowing market spreads. If spreads are tight, execution should be favorable both when buying and selling. However, that is not the behavior displayed.

Recommendation

Take into consideration whether the `orderType` being executed is an increase or decrease `orderType` when adjusting the `updatedPrice` by the `deviationPoints`.

Resolution

ORDM-7 | Small Positions Prevented From Being Closed

Category	Severity	Commit	Location	Status
Validation	● High	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 213	Pending

Description

The `_deductFeesFromPosition` function reverts if the remaining position collateral after deducting fees is less than the configured minimum collateral for the market.

However this `_deductFeesFromPosition` function is called during `_closePosition` execution, therefore positions that have accumulated enough fees to be put under the minimum collateral amount cannot be closed.

Recommendation

Do not validate the minimum collateral amount in the `_deductFeesFromPosition` when closing a position.

Resolution

ORDM-8 | Misconfigured Markets Can Break The Protocol

Category	Severity	Commit	Location	Status
Validation	● High	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 852	Pending

Description

The `addNewMarket` function allows a market to be added with the specified `_marketId`, however the `_newMarket.marketId` is not validated to be the same as the provided `_marketId`.

Additionally, the `Market` struct should not store a `marketId` as it is unnecessary and can always be accessed from an order or position object.

Recommendation

Remove the `marketId` attribute on the `Market` struct as it is unnecessary and leads to misconfiguration.

Resolution

ORDM-9 | updatedAvgPrice Always Rounds Down

Category	Severity	Commit	Location	Status
Rounding	● High	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 320	Pending

Description

The `updatedAvgPrice` computed in the `_increasePosition` function is always rounded down, therefore it is possible for users with long positions to increase their position at an `executionPrice` that is higher than their average price and see no change in their position's average price.

For example, an increase order with the following characteristics will not change the average price of the position.

- `positionSize = 1e18`
- `avgPrice = 100e8`
- `orderSize = 1e10`
- `executionPrice = 101e8`

Therefore, the `updatedAvgPrice = (1e18 * 100e8 + 1e10 * 101e8) / (1e18 + 1e10) = 100e8`.

For some tokens this rounding can yield opportunities for traders to manipulate their average price and make risk free profits.

Recommendation

Use roundUp division when calculating the `updatedAvgPrice` for longs and round down division when calculating the `updatedAvgPrice` for shorts.

Resolution

ORDM-10 | Lack Of Reserve Validation

Category	Severity	Commit	Location	Status
Validation	● High	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol	Pending

Description

Although the OrderManager validates that the position does not increase the open interest past the `maximumOi`, LPers are still exposed to great risk as there is no validation on the size of the position relative to the funds available in the market.

With a large enough order which is still within OI bounds, a user in profit can drain the entirety of the MarketVault's reserves and leave LPer's insolvent. Furthermore, any traders in profit would not be able to claim their profit. This would break a core function of the protocol.

Recommendation

Add validation checks to ensure the position size may be only up to a specific percentage of funds available in the market.

Resolution

ORDM-11 | Open Interest Validation Becomes Meaningless

Category	Severity	Commit	Location	Status
Validation	● High	457f9f2e247e681e2e183a910d25335cb0473fc7	OrderManager.sol: 110	Pending

Description

Whenever the Aggregate `vaultOpenInterest` is increased or decreased, it is done so at the current value of the index token:

- User opens a position with size 10 ETH @ $2,000 / ETH
- `vaultOpenInterest` is increased from 0 to $20,000
- The user then closes their position after ETH drops to $1,000 / ETH
- `vaultOpenInterest` is decreased from $20,000 → $10,000
- $10,000 `vaultOpenInterest` remains even though there are no open positions

This inaccuracy leads to cases where the `vaultOpenInterest` is massively inflated or massively reduced depending on whether user's close their positions at a higher or lower index token price than when they opened.

Over the lifecycle of the vault this validation will become entirely detached from the actual open interest of position's using the vault as backing, perturbing the purpose of the validation and preventing actions from occurring when the `vaultOpenInterest` is inflated.

Recommendation

Use the open interest that is tracked on a per market basis and value the open interest of each market in aggregate based on the current price of the each index token during validation.

This solution requires an O(n) approach where n is the number of markets configured for a single vault. This ought to be fine as long as the number of markets for a single vault is limited to a reasonable amount and the aggregate `vaultOpenInterest` is computed once per transaction.

Resolution

FWD-1 | Relayer May Censor Transactions

Category	Severity	Commit	Location	Status
Validation	Medium	457f9f2e247e681e2e183a910d25335cb0473fc7	ParifiForwarder.sol: 189	Pending

Description

The `gasSent` is validated to be greater than `64/63 * the transaction.minGas`, however the `gasSent` is recorded at the beginning of the `execute` function which has a non-trivial amount of logic, that will expend gas, before executing the external call.

Therefore the remaining gas that is forwarded to the external call may be less than the `transaction.minGas`.

Recommendation

Add a buffer to the `gasSent` validation that comfortably covers any expenditure that would occur before the external call.

Resolution

MKTV-2 | Maximum Open Interest Configuration Risk

Category	Severity	Commit	Location	Status
Configuration	Medium	457f9f2e247e681e2e183a910d25335cb0473fc7	ParifiVault.sol: 211	Pending

Description

The admin can configure any `maxVaultOpenInterest` value with the `setMaximumVaultOpenInterest` function. This poses a risk as the admin may accidentally configure a `maxVaultOpenInterest` that is below the current `vaultOpenInterest`.

Consider the following scenario:

- `maxVaultOpenInterest` is 15
- `vaultOpenInterest` is 10
- `maxVaultOpenInterest` is set to 5
- A liquidation that would reduce the `vaultOpenInterest` by 3 cannot execute as it would fail the `maxVaultOpenInterest` validation.

Recommendation

Take care when using the `setMaximumVaultOpenInterest` function. Consider adding validation such that the `maxVaultOpenInterest` cannot be set below the current `vaultOpenInterest`.

Resolution

ORDM-12 | Unexpected Limit Execution

Category	Severity	Commit	Location	Status
Unexpected Behavior	Medium	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 389	Pending

Description

Limit orders are traditionally defined to execute at or better than the limit price. However, the current limit price check disallows execution when the market price is equivalent to the expected price. This may lead to unexpectedly failed entries for users who expected their order to be executed once price reached their limit price.

```
if (userOrder.isLimitOrder) {
   if (
      (userOrder.triggerAbove && executionPrice <= expectedPrice)
         || (!userOrder.triggerAbove && executionPrice >= expectedPrice)
       ) {
      revert LibError.PriceMismatch(executionPrice, expectedPrice);
  }
}
```

Recommendation

Allow limit orders to execute when they are at or better than the configured trigger price.

Resolution

ORDM-13 | Slippage Applies To Both Sides

Category	Severity	Commit	Location	Status
Logical Error	Medium	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 400	Pending

Description

For slippage calculations, `_settleOrder` calculates a percentage above the `expectedPrice` and a percentage below the `expectedPrice`.

However, slippage should not apply to both the upper and lower prices. Traditionally, when a user submits a long, slippage is measured against the best offer price. If a user submits a short, slippage is measured against the best bid price. PariFi is comparing against both a `lowerLimit` and `upperLimit` regardless of position direction.

A user who is long would like to buy the asset at a price lower than the `lowerLimit`. A user who is short would like to sell the asset at a price higher than the `upperLimit`.

Recommendation

Only compare the price against the `upperLimit` for longs and the `lowerLimit` for shorts.

Resolution

ORDM-14 | maxLeverage Bypassed

Category	Severity	Commit	Location	Status
Validation	Medium	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol	Pending

Description

Upon creating a new position the `maxLeverage` is not validated. The `maxLeverage` validation will occur when you make a new order, however fees, price, and the `maxLeverage` itself may change in between the time you create your order and when it gets executed.

Therefore it is possible to bypass the configured `maxLeverage` for a market.

Recommendation

Validate the `maxLeverage` after the order has been executed.

Resolution

ORDM-15 | Lacking Validation For New Markets

Category	Severity	Commit	Location	Status
Validation	Medium	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 853	Pending

Description

The `addNewMarket` function lacks several key validations on new markets being added such as validating the ranges for:

- `openingFee`
- `closingFee`
- `liquidationFee`
- `minCollateral`
- `liquidationThreshold`

Recommendation

Add validation on the configured values for each of the above.

Resolution

ORDM-16 | Lacking Validation When Updating Markets

Category	Severity	Commit	Location	Status
Validation	Medium	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 885	Pending

Description

The `updateExistingMarket` function lacks several key validations on new markets being updated such as validating the ranges for:

- `openingFee`
- `closingFee`
- `liquidationFee`
- `minCollateral`
- `liquidationThreshold`

Recommendation

Add validation on the configured values for each of the above.

Resolution

MKTV-3 | _withdrawalFee Not Validated In Constructor

Category	Severity	Commit	Location	Status
Validation	Medium	6719ad104276eb928938ed2e6c6d2443f50a50f8	MarketVault.sol: 79	Pending

Description

The `_withdrawalFee` is not validated to be within the `WITHDRAWAL_FEE_CAP` in the `MarketVault` constructor.

Recommendation

Validate that the `_withdrawalFee` is within the `WITHDRAWAL_FEE_CAP` in the `MarketVault` constructor.

Resolution

ORDM-17 | Pyth Prices Not Updated Before Being Used

Category	Severity	Commit	Location	Status
Logical Error	Medium	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol	Pending

Description

`priceFeed.updatePythPrice(priceUpdateData)` is not used to update the pyth price before `cancelPendingOrder` or `createNewPosition`.

To be entirely accurate prices should be updated before computing the fees in these functions using those pyth prices.

Recommendation

Update Pyth prices with `priceFeed.updatePythPrice(priceUpdateData)` for entirely up to date pricing when computing fees in the `cancelPendingOrder` and `createNewPosition` functions.

Resolution

ORDM-18 | ExecutionFee To Cover The Keeper's Gas

Category	Severity	Commit	Location	Status
Logical Error	Medium	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol	Pending

Description

Currently there is no remuneration for the keeper executing orders with the `settleOrder` function. It may be prudent to cover the gas costs for the keeper by introducing an `executionFee` that must be sent in when creating an order to cover the keeper's gas expenditure.

Such an `executionFee` would dissuade from potential keeper griefing as currently there are no fees charged upon cancelling decrease orders with the `cancelPendingOrder` function. Malicious actors could see the keeper submit a `settleOrder` transaction and front-run it to cancel their order and avoid paying any fees. Though such an attack is dubious at best on the Arbitrum network.

Recommendation

Consider implementing an `executionFee` to remunerate the keeper's gas expenditure when settling orders. Additionally consider charging a percentage of this `executionFee` upon the cancellation of an order, especially decrease orders as they currently have no fees applied on cancellation.

Resolution

ORDM-19 | Fees May Not Be Skipped Due To Misconfiguration

Category	Severity	Commit	Location	Status
Validation	Medium	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 885	Pending

Description

In the `updateExistingMarket` function the admin may update the market and provide an `_updatedMarket` with a non-paused status e.g. `isLive = true`, however this would avoid the `dataFabric.unpauseMarket` call as the market has been toggled to a live status without calling the `toggleMarketStatus` function.

Recommendation

Either require that markets are created with an `isLive` of `false` or invoke the `dataFabric.unpauseMarket` function in the `updateExistingMarket` when the `_updatedMarket.isLive == true`.

Resolution

ORDM-20 | Leverage Relies On EMA

Category	Severity	Commit	Location	Status
Logical Error	Medium	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol	Pending

Description

A position's leverage is calculated using an EMA price, however this is a lagging indicator and will not be entirely accurate to current prices.

Therefore users will be able to open positions that, at current prices, would be above the `maxLeverage`. However by the EMA are not above `maxLeverage`.

Recommendation

Consider if this is the desired behavior, if not, use current prices to calculate the position's leverage.

Resolution

ORDM-21 | User's Leverage Changes With The Index Price

Category	Severity	Commit	Location	Status
Unexpected Behavior	Medium	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol	Pending

Description

The max leverage check relies on the index token price relative to the collateral token price, if the index token price moves relative to the collateral token price, a position's leverage changes.
This means the leverage of user's position can change even if the price of their collateral token does not move and they don't make any orders.

The leverage of a position will change with the price of the index token. However this is unexpected behavior as often the leverage of a perpetual position is a ratio of the collateral deposited and the initial size of the position, which does not change with the index price.

Recommendation

Determine if this ought to be the expected behavior and if so ensure it is well documented for users.

Resolution

ORDM-22 | Fees Based On EMA

Category	Severity	Commit	Location	Status
Logical Error	Medium	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol	Pending

Description

The fees on lines 524 and 651 are calculated based on an EMA price which is a lagging indicator and will not be entirely accurate to the current token price.

Recommendation

Consider if this is desired. If not, use the `convertMarketToToken` function to compute the value of these fees.

Resolution

ORDM-23 | Read-only Reentrancy Potential

Category	Severity	Commit	Location	Status
Reentrancy	Medium	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 295	Pending

Description

In the `_closePosition` function the position is deleted at the end of the function call, however for tokens with callbacks it would be safer to delete the position and update the market data before the transfer, it may also be wise to delete the position before the `distributeFees` external call as well, even though this is supposedly a trusted external call.

Recommendation

Ensure state is modified before initiating token transfers to protect against potential read-only reentrancies with ERC-777 tokens.

Resolution

ORDM-24 | Inefficient Limit Order Validation

Category	Severity	Commit	Location	Status
Validation	Medium	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 380	Pending

Description

When executing a limit order the `expectedPrice` is validated for the first time, however this validation should be carried out when the order is first created. Otherwise user's may submit many orders that will always fail this validation in order to waste the keeper's gas upon execution of the `settleOrder` function.

Recommendation

Validate the `expectedPrice` for the `limitOrder` when first creating the order.

Resolution

ORDM-25 | Fees Can Be Avoided With Small Order Sizes

Category	Severity	Commit	Location	Status
Rounding	Medium	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol	Pending

Description

Throughout the `OrderManager` contract, fees are calculated using round down division. Therefore it is possible for users to avoid paying fees by using order sizes small enough such that they round down to 0.

With sponsored order creation through the `PariFiForwarder` this can be a viable method to avoid fees.

Recommendation

Employing such a strategy would cost a significant amount of gas and is unlikely to ever be profitable, however it may be preferred to use round up division when computing these fees.

Resolution

ORDM-26 | Risk Of Stale Pricing

Category	Severity	Commit	Location	Status
Validation	Medium	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 524-525	Pending

Description

When creating a new position, the Pyth price feed is not updated. The fees calculation and the leverage validation are at risk of utilizing a stale price. This is because the `convertMarketToTokenSecondary` function is called which uses the unsafe EMA price point without validating publish time.

```
PythStructs.Price memory pythPrice = pyth.getEmaPriceUnsafe(priceId);
(priceUsd, priceTimestamp) = _calculatePythPrice(pythPrice, false);
```

According to Pyth documentation the `getEmaPriceUnsafe` function, “may return a price from arbitrarily far in the past. It is the caller's responsibility to check the returned `publishTime` to ensure that the update is recent enough for their use case.”

Recommendation

Consider validating the publish time or updating the price feed when creating a new position.

Resolution

MKTV-4 | Dead Address Can Be Constant

Category	Severity	Commit	Location	Status
Optimization	● Low	6719ad104276eb928938ed2e6c6d2443f50a50f8	MarketVault.sol	Pending

Description

To reduce bytecode and favor DRY the dead address referenced multiple times can instead be a constant in the `MarketVault` contract.

Recommendation

Declare the dead address as a constant variable in the `MarketVault` contract.

Resolution

ORDM-27 | Unnecessary userPosition Storage Declaration

Category	Severity	Commit	Location	Status
Optimization	● Low	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 710	Pending

Description

The `userPosition` variable is declared as a `storage` reference variable, however it would be much more efficient to declare it as a `memory` variable.

Recommendation

Declare the `userPosition` variable as a `memory` variable.

Resolution

ORDM-28 | Unnecessary userOrder Storage Declaration

Category	Severity	Commit	Location	Status
Optimization	● Low	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 375	Pending

Description

The `userOrder` variable is declared as a `storage` reference variable, however it would be much more efficient to declare it as a `memory` variable.

Recommendation

Declare the `userOrder` variable as a `memory` variable.

Resolution

ORDM-29 | Leverage Rounded Down

Category	Severity	Commit	Location	Status
Rounding	● Low	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 670	Pending

Description

When computing the leverage of a position in the `calculateLeverage` function round down division is used. The result of the rounding is that the position appearing as if it has slightly lower leverage than it actually does.

Recommendation

Though this rounding will have a minimal impact it may be worthwhile to use `roundUp` division to avoid position's being technically above the allowed leverage yet rounded to within the allowed range.

Resolution

ORDM-30 | Unnecessary Handling Of 0 Collateral

Category	Severity	Commit	Location	Status
Optimization	● Low	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 696	Pending

Description

In the event that `isProfit` is `false` and the `pnlInCollateral` is greater in magnitude than the `collateralAmount` for the position an alternative leverage calculation is used to avoid divide by zero reverts. However any position where the collateral is exactly equal to the `pnlInCollateral` or insufficient to cover the `pnlInCollateral` technically has infinite leverage and should therefore always fail the `maxLeverage` validation.

Recommendation

Simply revert with a `maxLeverage` validation failure in the event that the `pnlInCollateral >= _collateralAmount`.

Resolution

ORDM-31 | Loss Amount Is Rounded Down

Category	Severity	Commit	Location	Status
Rounding	● Low	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol: 754	Pending

Description

When computing the `profitOrLoss` round down division is always used whether or not this variable represents profit or loss.

This way user's are able to avoid a fraction of their losses that are lost from truncation. This amount will almost always be negligible, however the loss amount should use `roundUp` division to act in the protocol's favor.

Recommendation

Use `roundUp` division when the `profitOrLoss` variable represents a loss amount.

Resolution

ORDM-32 | Superfluous orderToPositionId Mapping

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	6719ad104276eb928938ed2e6c6d2443f50a50f8	OrderManager.sol	Pending

Description

The `orderToPositionId` is unnecessary and inefficient as each order can simply have a `positionId` as an attribute, for open orders this id can be ignored.

Recommendation

Remove the `orderToPositionId` mapping and introduce an attribute on the order struct to store the `positionId` the order is meant for.

Resolution

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