Gains Network Security Review

Pashov Audit Group

Conducted by: Peakbolt, ast3ros, Said

May 28th 2024 - June 10th 2024

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **GainsNetwork-org/gTrade-contracts** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Gains Network

Gains Network is a liquidity-efficient decentralized leveraged trading platform. Trades are opened with DAI, USDC or WETH collateral, regardless of the trading pair. The leverage is synthetic and backed by the respective gToken vault, and the GNS token. Trader profit is taken from the vaults to pay the traders PnL (if positive), or receives trader losses from trades if their PnL was negative.

5. Risk Classification

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

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - 0ffc82581ea1e4c002435fa3bb328080cced4179

fixes review commit hash - fc44d22a485e6be9eafa70f0967449c8f21d60ea

Scope

The following smart contracts were in scope of the audit:

- `BorrowingFeesUtils`
- `ConstantsUtils`
- `PairsStorageUtils`
- `PriceAggregatorUtils`
- `PriceImpactUtils`
- `ReferralsUtils`
- `TradingCallbackUtils`
- `TradingCommonUtils`
- `TradingInteractionsUtils`
- `TradingStorageUtils`
- `UpdateLeverageLifecycles`
- `DecreasePositionUtils`
- `IncreasePositionUtils`
- `UpdatePositionSizeLifecycles`

7. Executive Summary

Over the course of the security review, Peakbolt, ast3ros, Said engaged with Gains Network to review Gains Network. In this period of time a total of **15** issues were uncovered.

Protocol Summary

Protocol Name	Gains Network
Repository	https://github.com/GainsNetwork-org/gTrade-contracts
Date	May 28th 2024 - June 10th 2024
Protocol Type	Leveraged trading platform

Findings Count

Severity	**Amount**
Critical	1
High	4
Medium	6
Low	4
Total Findings	**15**

Summary of Findings

ID	Title	Severity	Status
[C-01]	Decreasing position size via leverage update can be abused to steal from diamond	Critical	Resolved
[H-01]	FeeTierPoints is incorrectly increased twice	High	Resolved
[H-02]	newLeverage wrongly calculated inside requestIncreasePositionSize	High	Resolved
[H-03]	Overcharging of closing and trigger fees	High	Resolved
[H-04]	Price impact cannot properly be removed	High	Resolved
[M-01]	Incorrect calculation of new liquidation price	Medium	Resolved
[M-02]	Double counting of existing open interest	Medium	Resolved
[M-03]	Open interest calculation is incorrect	Medium	Resolved
[M-04]	updateTradePosition() fails to limit TP/SL distance	Medium	Resolved
[M-05]	removePriceImpactOpenInterest() fails to account for expired OI	Medium	Resolved
[M-06]	addPriceImpactOpenInterest() fails to scale expiredOiUsd	Medium	Resolved
[L-01]	Use _trade.collateralAmount directly to prevent precision loss	Low	Resolved
[L-02]	requestIncreasePositionSize could fail	Low	Resolved
[L-03]	Bypassing max and min leverage limits	Low	Resolved
[L-04]	__placeholder is not set to 0 inside the Trade data	Low	Resolved

8. Findings

8.1. Critical Findings

[C-01] Decreasing position size via leverage update can be abused to steal from diamond

Severity

Impact: High

Likelihood: High

Description

The trader can decrease position size using leverage update, which will then realize the partial profit/loss based on position size delta.

However, in the case of a profit, `handleTradePnl()` will incorrectly send the closing fees to the trader instead of the vault. This allows the trader to receive more profit than expected by stealing from the diamond. Over time, this will slowly drain the diamond as the closing fees are still distributed despite not receiving it.

Suppose the trader has the following position,

- existing leverage = 10
- existing collateral = 100 DAI
- existing position size = 1000 DAI
- existing profit of 500 DAI

And the trader reduces position size by leverage delta,

- delta leverage = 5
- delta position size = 500 DAI
- delta profit value = 500 * 500/1000 = 250 DAI
- delta trade value = 500 + 250 = 750 DAI

After deducting fees,

- borrowing fee = 1
- staking/vault fees = 5

We will get the following values,

- `_availableCollateralInDiamond` = 0 - 5 = -5
- `collateralSentToTrader` = 0 + 250 - 1 - 5 = 244

Now when `handleTradePnl()` handle it,

- vault sends trader `collateralSentToTrader - collateralLeftInStorage` = 244 - (-5) = 249 DAI
- the closing fee of 5 DAI is incorrectly sent from the vault to the trader, instead of sending it to the diamond

```
function handleTradePnl(
        ITradingStorage.Trade memory _trade,
        int256 _collateralSentToTrader,
        int256 _availableCollateralInDiamond,
        uint256 _borrowingFeeCollateral
    ) external {
        if (_collateralSentToTrader > _availableCollateralInDiamond) {
            //@audit this will incorrectly send closing fees to trader when
            // _availableCollateralInDiamond is negative
            getGToken(_trade.collateralIndex).sendAssets(
                uint256
                  (_collateralSentToTrader - _availableCollateralInDiamond),
                _trade.user
            );

            if (_availableCollateralInDiamond > 0)
                transferCollateralTo(
                  _trade.collateralIndex,
                  _trade.user,
                  uint256
                )
        } else {
            getGToken(_trade.collateralIndex).receiveAssets(
                uint256
                  (_availableCollateralInDiamond - _collateralSentToTrader),
                _trade.user
            );
            if (_collateralSentToTrader > 0)
                transferCollateralTo(
                  _trade.collateralIndex,
                  _trade.user,
                  uint256
                )
        }

        emit ITradingCallbacksUtils.BorrowingFeeCharged
          (_trade.user, _trade.collateralIndex, _borrowingFeeCollateral);
    }
```

Recommendations

When reducing position by leverage, send the closing fees from vault to diamond, instead of sending to the trader.

8.2. High Findings

[H-01] `FeeTierPoints` is incorrectly increased twice

Severity

Impact: Medium

Likelihood: High

Description

The execution of the position size increase will call `TradingCommonUtils.updateFeeTierPoints()` to update the trader fee tier points by the position size delta.

However, the trader fee tier points have been updated in the preceding call `TradingCommonUtils.processOpeningFees()`.

That means the trader fee tier points are incorrectly updated twice, causing the trader to receive double the points. A trader can exploit this to earn more points and get fee reduction quicker, causing protocol, vault users, and stakers to incur a loss in fee revenue.

```
function executeIncreasePositionSizeMarket(
        ITradingStorage.PendingOrder memory _order,
        ITradingCallbacks.AggregatorAnswer memory _answer
    ) external {
            ...
            // 5. If passes further validation, execute callback
            if (cancelReason == ITradingCallbacks.CancelReason.NONE) {
                // 5.1 Update trade collateral / leverage / open price in
                // storage, and reset trade borrowing fees
                IncreasePositionSizeUtils.updateTradeSuccess
                  (existingTrade, values);


                // 5.2 Distribute opening fees
                //@audit this will update trader fee tier points for the
                // increase of position size
                TradingCommonUtils.processOpeningFees(
                    existingTrade,
                    values.positionSizeCollateralDelta,
                    _order.orderType
                );

                // 5.3 Store trader fee tier points for position size delta
                //@audit this will incorrectly update trader fee tier points
                // again
                TradingCommonUtils.updateFeeTierPoints(
                    existingTrade.collateralIndex,
                    existingTrade.user,
                    existingTrade.pairIndex,
                    values.positionSizeCollateralDelta
                );
            }
        }

        ;;;
    }
```

Recommendations

This can be resolved by removing the redundant update of fee tier points.

[H-02] `newLeverage` wrongly calculated inside `requestIncreasePositionSize`

Severity

Impact: High

Likelihood: Medium

Description

When users call `increasePositionSize` and request an increase in position size, it will eventually trigger `IncreasePositionSizeUtils.validateRequest` to validate the request. However, when calculating `newLeverage`, it incorrectly calculates `(existingPositionSizeCollateral + positionSizeCollateralDelta * 1e3) / newCollateralAmount` instead of `(existingPositionSizeCollateral + positionSizeCollateralDelta) * 1e3 / newCollateralAmount`, causing the `newLeverage` to be lower than it should be.

```
function validateRequest(
        ITradingStorage.Trade memory _trade,
        IUpdatePositionSizeUtils.IncreasePositionSizeInput memory _input
    ) internal view returns (uint256 positionSizeCollateralDelta) {
        // ....

                uint256 newCollateralAmount = _trade.collateralAmount + _input.collat
        uint256 newLeverage = isLeverageUpdate
            ? _trade.leverage + _input.leverageDelta
>>>         :
  (existingPositionSizeCollateral + positionSizeCollateralDelta * 1e3) / newCollateral
        {

            uint256 openingFeesCollateral = ((_getMultiCollatDiamond
              ().pairOpenFeeP(_trade.pairIndex) *
                2 +
                _getMultiCollatDiamond().pairTriggerOrderFeeP
                  (_trade.pairIndex)) *
                TradingCommonUtils.getPositionSizeCollateralBasis(
                    _trade.collateralIndex,
                    _trade.pairIndex,
                    positionSizeCollateralDelta
                )) /
                ConstantsUtils.P_10 /
                100;

            uint256 newPositionSizeCollateral = existingPositionSizeCollateral +
                positionSizeCollateralDelta -
                (
                  (borrowingFeeCollateral + openingFeesCollateral) * newLeverage) /
                1e3;

            if (newPositionSizeCollateral <= existingPositionSizeCollateral)
                revert IUpdatePositionSizeUtils.NewPositionSizeSmaller();
        }
```

The `newLeverage` is used to calculate `newPositionSizeCollateral`, which will be used for checking against `existingPositionSizeCollateral`. Incorrectly calculating `newLeverage` will cause `newPositionSizeCollateral` to be higher than it should be and could lead to incorrectly passing the validation check. And it will also calculate and check `isWithinExposureLimits` using lower `newLeverage` amount, potentially bypass `exposureLimits`.

Recommendations

Update the `newLeverage` calculation :

```
uint256 newLeverage = isLeverageUpdate
            ? _trade.leverage + _input.leverageDelta
-            :
- (existingPositionSizeCollateral + positionSizeCollateralDelta * 1e3) / newCollateral
+            :
+ (existingPositionSizeCollateral + positionSizeCollateralDelta) * 1e3 / newCollateral
```

[H-03] Overcharging of closing and trigger fees

Severity

Impact: Medium

Likelihood: High

Description

Currently, the calculation for closing and trigger fees applies a fixed `5%` fee to all `non-MARKET_CLOSE` orders. This fee should only apply to `LIQ_CLOSE` orders. For `TP_CLOSE` and `SL_CLOSE` orders, the calculation should use the pair close fee percentage and pair trigger order fee percentage, respectively. As a result, `TP_CLOSE` and `SL_CLOSE` orders are being overcharged.

```
function processClosingFees(
        ITradingStorage.Trade memory _trade,
        uint256 _positionSizeCollateral,
        ITradingStorage.PendingOrderType _orderType
    ) external returns (ITradingCallbacks.Values memory values) {
        // 1. Calculate closing fees
        values.positionSizeCollateral = getPositionSizeCollateralBasis(
            _trade.collateralIndex,
            _trade.pairIndex,
            _positionSizeCollateral
        ); // Charge fees on max(min position size, trade position size)


             values.closingFeeCollateral = _orderType == ITradingStorage.PendingOr
            ? (values.positionSizeCollateral * _getMultiCollatDiamond
              ().pairCloseFeeP(_trade.pairIndex)) /
                100 /
                ConstantsUtils.P_10
            :
            //(_trade.collateralAmount * 5) / 100; // @audit charge fixed 5% for non-M


             values.triggerFeeCollateral = _orderType == ITradingStorage.PendingOr
            ? (values.positionSizeCollateral * _getMultiCollatDiamond
              ().pairTriggerOrderFeeP(_trade.pairIndex)) /
                100 /
                ConstantsUtils.P_10
            : values.closingFeeCollateral; // @audit charge fixed 5% for
            // non-MARKET_CLOSE
        ...
    }
```

Recommendations

Revise the fee calculation logic to apply the pair close fee percentage and pair trigger order fee percentage specifically for `TP_CLOSE` and `SL_CLOSE` orders, while retaining the 5% fee for `LIQ_CLOSE` orders.

[H-04] Price impact cannot properly be removed

Severity

Impact: Medium

Likelihood: High

Description

When user's increase position size or update leverage request is successfully executed, it will update user's trade position by calling `updateTradePosition`

and providing the new `newCollateralAmount` and `newLeverage`.

```
function updateTradeSuccess(
        ITradingStorage.Trade memory _existingTrade,
        IUpdatePositionSizeUtils.IncreasePositionSizeValues memory _values
    ) internal {
        // 1. Charge borrowing fees and opening fees from trade collateral

                _values.newCollateralAmount -= _values.borrowingFeeCollateral + _valu

        // 2. Update trade in storage
>>>     _getMultiCollatDiamond().updateTradePosition(
            ITradingStorage.Id(_existingTrade.user, _existingTrade.index),
            uint120(_values.newCollateralAmount),
            uint24(_values.newLeverage),
            uint64(_values.newOpenPrice)
        );

        // 3. Reset trade borrowing fees to zero
        _getMultiCollatDiamond().resetTradeBorrowingFees(
            _existingTrade.collateralIndex,
            _existingTrade.user,
            _existingTrade.pairIndex,
            _existingTrade.index,
            _existingTrade.long
        );
    }
```

And inside `updateTradePosition`, it will update the trade data and trigger `TradingCommonUtils.handleOiDelta` to add the new position size delta to the price impact.

```
function handleOiDelta(
        ITradingStorage.Trade memory _trade,
        ITradingStorage.TradeInfo memory _tradeInfo,
        uint256 _newPositionSizeCollateral
    ) external {
        uint256 existingPositionSizeCollateral = getPositionSizeCollateral
          (_trade.collateralAmount, _trade.leverage);

        if (_newPositionSizeCollateral > existingPositionSizeCollateral) {
>>>         addOiCollateral
  (_trade, _newPositionSizeCollateral - existingPositionSizeCollateral);
        } else if
          (_newPositionSizeCollateral < existingPositionSizeCollateral) {
            removeOiCollateral(
              _trade,
              _tradeInfo,
              existingPositionSizeCollateral-_newPositionSizeCollateral
            );
        }
    }
```

```
function addPriceImpactOpenInterest
      (uint128 _openInterestUsd, uint256 _pairIndex, bool _long) internal {


                IPriceImpact.OiWindowsSettings storage settings = priceImpactStorage.

>>>     uint256 currentWindowId = _getCurrentWindowId(settings);

                IPriceImpact.PairOi storage pairOi = priceImpactStorage.windows[setti
            currentWindowId
        ];

        if (_long) {
            pairOi.oiLongUsd += _openInterestUsd;
        } else {
            pairOi.oiShortUsd += _openInterestUsd;
        }

        emit IPriceImpactUtils.PriceImpactOpenInterestAdded(
            IPriceImpact.OiWindowUpdate(
              settings.windowsDuration,
              _pairIndex,
              currentWindowId,
              _long,
              _openInterestUsd
            )
        );
    }
```

It can be observed that the price impact is added to `currentWindowId`, and when the trade is closed and the trade price impact is removed, it will only remove the price impact at `_tradeInfo.lastOiUpdateTs`.

```
function removeOiCollateral(
        ITradingStorage.Trade memory _trade,
        ITradingStorage.TradeInfo memory _tradeInfo,
        uint256 _positionSizeCollateral
    ) public {
        _getMultiCollatDiamond().handleTradeBorrowingCallback(
            _trade.collateralIndex,
            _trade.user,
            _trade.pairIndex,
            _trade.index,
            _positionSizeCollateral,
            false,
            _trade.long
        );
        // @audit - is this always correct?
        _getMultiCollatDiamond().removePriceImpactOpenInterest(
            // when removing OI we need to use the collateral/usd price when the
            // OI was added
            convertCollateralToUsd(
                _positionSizeCollateral,
                _getMultiCollatDiamond().getCollateral
                  (_trade.collateralIndex).precisionDelta,
                _tradeInfo.collateralPriceUsd
            ),
            _trade.pairIndex,
            _trade.long,
            _tradeInfo.lastOiUpdateTs // @audit - this will only remove price
            // impact from initial collateral size
        );
    }
```

This will cause the price impact to not be properly removed and result in a higher price impact than it should be.

Recommendations

Updating the price impact to the same `_tradeInfo.lastOiUpdateTs` can be performed, or separately tracking the window ID when the position size is increased.

8.3. Medium Findings

[M-01] Incorrect calculation of new liquidation price

Severity

Impact: Medium

Likelihood: Medium

Description

When calculating the newLiqPrice in `IncreasePositionSizeUtils.prepareCallbackValues`, the function uses `newCollateralAmount` and `newLeverage`.

```
function prepareCallbackValues(
        ITradingStorage.Trade memory _existingTrade,
        ITradingStorage.Trade memory _partialTrade,
        ITradingCallbacks.AggregatorAnswer memory _answer
    ) internal view returns
      (IUpdatePositionSizeUtils.IncreasePositionSizeValues memory values) {
        ...
        values.newLiqPrice = _getMultiCollatDiamond().getTradeLiquidationPrice(
            IBorrowingFees.LiqPriceInput(
                _existingTrade.collateralIndex,
                _existingTrade.user,
                _existingTrade.pairIndex,
                _existingTrade.index,
                uint64(values.newOpenPrice),
                _existingTrade.long,
                values.newCollateralAmount, // @audit use newCollateralAmount &
                // newLeverage => borrowing fee is overstated
                values.newLeverage
            )
        );
        ...
    }
```

To calculate the liquidation price, the current borrowing fee amount needs to be considered. Using `newCollateralAmount` and `newLeverage` as inputs can lead to overstating the borrowing fee if these values are higher than the current collateral and leverage. This results in an incorrect new liquidation price, making it higher for long positions and lower for short positions. This

miscalculation can cause unexpected reverts when validating the callback due to the liquidation price being incorrectly calculated as reached.

```
function getTradeBorrowingFee(
        IBorrowingFees.BorrowingFeeInput memory _input
    ) internal view returns (uint256 feeAmountCollateral) {
        ...

        feeAmountCollateral =
        //(_input.collateral * _input.leverage * borrowingFeeP) / 1e3 / ConstantsUtils
    }
```

Recommendations

Pass the current borrowing fee to calculate the new liquidation price instead of recalculating the borrowing fee with `newCollateralAmount` and `newLeverage`.

[M-02] Double counting of existing open interest

Severity

Impact: Medium

Likelihood: Medium

Description

When validating a request to increase position size, the function `validateRequest` checks if the trade stays within exposure limits. However, the amounts and leverage used are `newCollateralAmount` and `newLeverage` instead of `collateralDelta` and `leverageDelta`.

We have `newPositionSizeCollateral = existingPositionSizeCollateral + positionSizeCollateralDelta`. The current open interest level already accounts for the `existingPositionSizeCollateral`. Therefore, to check if the trade after increasing position size stays within exposure limits, we should check if the additional amount `positionSizeCollateralDelta` is within the exposure limits.

If `newCollateralAmount` and `newLeverage` are used for the check, the trade might be considered as exceeding the exposure limits even though the

additional amount is within the limits because the current open interest level is double-counted.

```
function validateRequest(
        ITradingStorage.Trade memory _trade,
        IUpdatePositionSizeUtils.IncreasePositionSizeInput memory _input
    ) internal view returns (uint256 positionSizeCollateralDelta) {
        ...

        // 4. Make sure trade stays within exposure limits
        if (
            !TradingCommonUtils.isWithinExposureLimits(
                _trade.collateralIndex,
                _trade.pairIndex,
                _trade.long,
                newCollateralAmount, // @audit already counted the existing
                // collateral
                newLeverage
            )
        ) revert ITradingInteractionsUtils.AboveExposureLimits();
    }
```

The same issue can be identified in `IncreasePositionSizeUtils.validateCallback`.

```
function validateCallback(
        ITradingStorage.Trade memory _existingTrade,
        IUpdatePositionSizeUtils.IncreasePositionSizeValues memory _values,
        ITradingCallbacks.AggregatorAnswer memory _answer,
        uint256 _expectedPrice,
        uint256 _maxSlippageP
    ) internal view returns (ITradingCallbacks.CancelReason cancelReason) {
        ...
            : !TradingCommonUtils.isWithinExposureLimits(
                _existingTrade.collateralIndex,
                _existingTrade.pairIndex,
                _existingTrade.long,
                _values.newCollateralAmount, // @audit already counted the
                // existing collateral
                _values.newLeverage
            )
        ...
    }
```

Recommendations

Only account for the additional position size when checking if the trade stays within exposure limits. Adjust the code to handle different scenarios:

- If `collateralDelta` > 0 and `leverageDelta` > 0, use `collateralDelta` and `leverageDelta`.
- If `collateralDelta` $= 0$, use `collateralAmount` and `leverageDelta`.

[M-03] Open interest calculation is incorrect

Severity

Impact: Medium

Likelihood: Medium

Description

The open interest (OI) delta is calculated using the `current collateral/USD` price when `adding OI`. Conversely, when `removing OI`, the delta is calculated using the `collateral/USD price at the time the OI was added`. This method works correctly if an order is only opened and closed once. However, if the order position is increased or decreased multiple times, the OI calculation becomes incorrect.

For example, if a position is opened when the collateral price is $100 and then increased when the price is $110, the OI to be removed upon closing is incorrectly calculated using the $110 price.

```
function handleOiDelta(
        ITradingStorage.Trade memory _trade,
        ITradingStorage.TradeInfo memory _tradeInfo,
        uint256 _newPositionSizeCollateral
    ) external {
        uint256 existingPositionSizeCollateral = getPositionSizeCollateral
          (_trade.collateralAmount, _trade.leverage);

        if (_newPositionSizeCollateral > existingPositionSizeCollateral) {
            addOiCollateral
            //(_trade, _newPositionSizeCollateral - existingPositionSizeCollateral); /
        } else if
          (_newPositionSizeCollateral < existingPositionSizeCollateral) {
            removeOiCollateral
            //(_trade, _tradeInfo, existingPositionSizeCollateral - _newPositionSizeCo
        }
    }
```

Recommendations

Track the total open interest of a position and adjust the open interest based on the maximum total open interest when the position is closed.

[M-04] `updateTradePosition()` fails to limit TP/SL distance

Severity

Impact: High

Likelihood: Low

Description

An increase in position size will update the trade's open price due to the partial trade with a different open price. The trade new open price will be updated via `updateTradePosition()`.

However, `updateTradePosition()` fails to adjust the TP/SL using `_limitTpDistance()` and `_limitSlDistance()` despite changing the open price. It could cause the existing TP/SL that are near limits to be incorrectly exceeding the TP/SL limits based on max PnL.

Traders could abuse this to set a TP that exceeds the max profit of 900%.

```
function updateTradePosition(
        ITradingStorage.Id memory _tradeId,
        uint120 _collateralAmount,
        uint24 _leverage,
        uint64 _openPrice
    ) external {
        ITradingStorage.TradingStorage storage s = _getStorage();

                ITradingStorage.Trade storage t = s.trades[_tradeId.user][_tradeId.in

                ITradingStorage.TradeInfo storage i = s.tradeInfos[_tradeId.user][_tr

        if (!t.isOpen) revert IGeneralErrors.DoesntExist();
        if
          (t.tradeType != ITradingStorage.TradeType.TRADE) revert IGeneralErrors.Wrong
        if
          (_collateralAmount * _leverage == 0) revert ITradingStorageUtils.TradePositi
        if (_openPrice == 0) revert ITradingStorageUtils.TradeOpenPriceZero();

        //@audit TODO
        TradingCommonUtils.handleOiDelta(
            t,
            i,
            TradingCommonUtils.getPositionSizeCollateral
              (_collateralAmount, _leverage)
        );

        t.collateralAmount = _collateralAmount;
        t.leverage = _leverage;
        t.openPrice = _openPrice;

        i.createdBlock = uint32(ChainUtils.getBlockNumber());

        emit ITradingStorageUtils.TradePositionUpdated
          (_tradeId, _collateralAmount, t.leverage, t.openPrice);
    }
```

Recommendations

Modify `updateTradePosition()` to adjust the TP/SL using `_limitTpDistance()` and `_limitSlDistance()` .

[M-05] `removePriceImpactOpenInterest()` fails to account for expired OI

Severity

Impact: Medium

Likelihood: Medium

Description

`removePriceImpactOpenInterest()` will ensure that it does not remove more OI than required by deducting the actual removed OI from `expiredOiUsd` when closing a trade.

However, it fails to account for `expiredOiUsd` when removing OI for partial position size reduction. That will cause the reduced OI to be higher than required, causing a lower price impact on subsequent trades.

Suppose the scenario,

- existing position size = 1000 (in the outdated window)
- active OI for trade = 0 (as it's outdated)

And the trader increases position size,

- delta position size = 500
- new position size = 1000 + 500 = 1500
- expired OI for trade = 1000 (from existing position)
- active OI for trade = 500

Now if the trader reduces position size while the latest OI is still active,

- delta position size = -700
- new position size = 1500 - 700 = 800
- removed OI for trade = 700 (now this is incorrect as the active OI was only 500)
- the correct OI to remove should be `min(deltaOiUsd, positionSizeUsd - expiredOiUsd)`

```
function removePriceImpactOpenInterest(
        address _trader,
        uint32 _index,
        bool _isPartial,
        uint128 _oiDeltaUsd
    ) internal {
        ...

        if (notOutdated) {

                        IPriceImpact.PairOi storage pairOi = priceImpactStorage.windo
            addWindowId
            ];

            //@audit When isPartial == true, need to handle the case where
            // _oiDeltaUsd > active OI

            if
              (!_isPartial) _oiDeltaUsd -= priceImpactStorage.tradePriceImpactInfos[_t

            // 3.2 Remove OI from trade last oi updated window
            if (trade.long) {

                                pairOi.oiLongUsd = _oiDeltaUsd < pairOi.oiLongUsd ? p
            } else {

                                pairOi.oiShortUsd = _oiDeltaUsd < pairOi.oiShortUsd ?
            }
        }

        ...
    }
```

Recommendations

The correct OI to remove should be `min(deltaOiUsd, positionSizeUsd - expiredOiUsd)`.

[M-06] `addPriceImpactOpenInterest()` fails to scale `expiredOiUsd`

Severity

Impact: Medium

Likelihood: Medium

Description

`addPriceImpactOpenInterest()` will keep track of the expired OI when adding OI so that closing of trade will not reduce more OI than necessary.

However, it fails to scale the OI USD value by the current collateral/usd price. As the `tradeInfo.collateralPriceUsd` will be updated with the current collateral/usd price, it will cause the removal of OI to be incorrect as the expired OI is based on the previous collateral/usd price and not the last updated collateral/usd price.

```
function addPriceImpactOpenInterest(
       address_trader,
       uint32_index,
       bool_isPartial,
       uint128_oiDeltaUsd
     ) internal {
       ...

        if (_isPartial) {
            uint256 existingOiWindowId = _getWindowId
              (tradeInfo.lastOiUpdateTs, settings);

            if (_isWindowPotentiallyActive
              (existingOiWindowId, currentWindowId)) {
                removePriceImpactOpenInterest
                  (_trader, _index, false, tradePriceImpactInfo.lastOiDeltaUsd);

                openInterestUsdToAdd += uint128(

                      (tradePriceImpactInfo.lastOiDeltaUsd * currentCollateralPriceUsd
                );
            } else {
                //@audit the added expiredOiUsd should also be scaled with
                // current collateral/usd price

                                    tradePriceImpactInfo.expiredOiUsd += tradePriceImpact
            }
        }
```

Recommendations

Scale the added `expiredOiUsd` with `(currentCollateralPriceUsd) / tradeInfo.collateralPriceUsd`.

8.4. Low Findings

[L-01] Use `_trade.collateralAmount` directly to prevent precision loss

In the `DecreasePositionSizeUtils.validateRequest` function, when calculating the new collateral amount if the leverage is updated, the following formula is used:

```
((TradingCommonUtils.getPositionSizeCollateral(

    ) - positionSizeCollateralDelta
```

However, the result of this formula is effectively `_trade.collateralAmount`, which can be used directly to prevent any precision loss that might occur during the calculation.

```
function validateRequest(
        ITradingStorage.Trade memory _trade,
        IUpdatePositionSizeUtils.DecreasePositionSizeInput memory _input
    ) internal view returns (uint256 positionSizeCollateralDelta) {
        ...

        uint256 newCollateralAmount = isLeverageUpdate
            ? ((TradingCommonUtils.getPositionSizeCollateral
              (_trade.collateralAmount, _trade.leverage) -
                positionSizeCollateralDelta) * 1e3) /
                //(_trade.leverage - _input.leverageDelta) // @audit = trade.collatera
            : _trade.collateralAmount - _input.collateralDelta;
        ...
    }
```

[L-02] `requestIncreasePositionSize` could fail

When users increase their position size, they can set `_input.collateralDelta` to 0 if they only want to increase the leverage.

```
function requestIncreasePositionSize
      (IUpdatePositionSizeUtils.IncreasePositionSizeInput memory _input) external {
        // 1. Base validation
        ITradingStorage.Trade memory trade = _baseValidateRequest
          (_input.user, _input.index);

        // 2. Increase position size validation


        // 3. Transfer collateral delta from trader to diamond contract
        //(nothing transferred for leverage update)
        // @audit - if collateral delta 0, should just skip
>>>     TradingCommonUtils.transferCollateralFrom
  (trade.collateralIndex, _input.user, _input.collateralDelta);

        // 4. Create pending order and make price aggregator request
        ITradingStorage.Id memory orderId = _initiateRequest(
            trade,
            true,
            _input.collateralDelta,
            _input.leverageDelta,
            positionSizeCollateralDelta,
            _input.expectedPrice,
            _input.maxSlippageP
        );

        emit IUpdatePositionSizeUtils.PositionSizeUpdateInitiated(
            orderId,
            trade.user,
            trade.pairIndex,
            trade.index,
            true,
            _input.collateralDelta,
            _input.leverageDelta
        );
    }
```

But if the used collateral token reverts on 0 transfer, the update will always fail.

Skip the transfer if `_input.collateralDelta` is 0.

[L-03] Bypassing max and min leverage limits

When a request is made to increase position size, the new leverage is checked to ensure it is within the minimum and maximum leverage limits.

```
function validateRequest(
        ITradingStorage.Trade memory _trade,
        IUpdatePositionSizeUtils.IncreasePositionSizeInput memory _input
    ) internal view returns (uint256 positionSizeCollateralDelta) {
        ...

        // 2. Revert if new leverage is below min leverage or above max leverage
        bool isLeverageUpdate = _input.collateralDelta == 0;
        {
            uint24 leverageToValidate = isLeverageUpdate
                ? _trade.leverage + _input.leverageDelta
                : _input.leverageDelta;
            if (
                leverageToValidate > _getMultiCollatDiamond().pairMaxLeverage
                //(_trade.pairIndex) * 1e3 || // @audit only check in validateReqquest
                leverageToValidate < _getMultiCollatDiamond().pairMinLeverage
                  (_trade.pairIndex) * 1e3
            ) revert ITradingInteractionsUtils.WrongLeverage();
        }
        ...
    }
```

However, during the callback validation, the new leverage is assumed to be within the min and max limits and is not rechecked. This means the leverage limits can be bypassed if the leverage limits are updated between the request and the callback. Users can potentially exploit this by making a request to increase position size with leverage that is out of the limit just before an update in the system leverage limits by front-runnning the update transaction.

```
function validateCallback(
        ITradingStorage.Trade memory _existingTrade,
        IUpdatePositionSizeUtils.IncreasePositionSizeValues memory _values,
        ITradingCallbacks.AggregatorAnswer memory _answer,
        uint256 _expectedPrice,
        uint256 _maxSlippageP
    ) internal view returns (ITradingCallbacks.CancelReason cancelReason) {
        uint256 maxSlippage = (uint256
          (_expectedPrice) * _maxSlippageP) / 100 / 1e3;

        cancelReason = (
            _existingTrade.long
                ? _values.priceAfterImpact > _expectedPrice + maxSlippage
                : _values.priceAfterImpact < _expectedPrice - maxSlippage
        )
            ? ITradingCallbacks.CancelReason.SLIPPAGE // 1. Check price after
            // impact is within slippage limits
            : _existingTrade.tp > 0 &&

                  (_existingTrade.long ? _answer.price >= _existingTrade.tp : _answer.
            ? ITradingCallbacks.CancelReason.TP_REACHED // 2. Check TP has not
            // been reached
            : _existingTrade.sl > 0 &&

                  (_existingTrade.long ? _answer.price <= _existingTrade.sl : _answer.
            ? ITradingCallbacks.CancelReason.SL_REACHED // 3. Check SL has not
            // been reached
            : (
                _existingTrade.long
                    ?
                      (_answer.price <= _values.existingLiqPrice || _answer.price <= _
                    :
                      (_answer.price >= _values.existingLiqPrice || _answer.price >= _
            )
            ? ITradingCallbacks.CancelReason.LIQ_REACHED // 4. Check current and
            // new LIQ price not reached
            : !TradingCommonUtils.isWithinExposureLimits(
                _existingTrade.collateralIndex,
                _existingTrade.pairIndex,
                _existingTrade.long,
                _values.newCollateralAmount,
                _values.newLeverage
            )
            ? ITradingCallbacks.CancelReason.EXPOSURE_LIMITS // 5. Check trade
            // still within exposure limits
            : ITradingCallbacks.CancelReason.NONE;  // @audit not check if the
            // leverage is within limit when callback
    }
```

Validate the new leverage during the callback to ensure it remains within the min and max leverage limits, even if the limits have been updated between the initial request and the callback.

[L-04] `__placeholder` is not set to 0 inside the Trade data

When a trade is opened, users need to provide `Trade` data that will be stored. Inside the `Trade` struct data, there is a `__placeholder` reserved for future usage. However, open trade operations currently allow users to provide arbitrary values to `__placeholder`. If in the future this field will be used, malicious users can input values to `__placeholder` before an upgrade is executed, avoiding input validation and potentially causing issues. Consider setting `_trade.__placeholder` to 0 inside the `_openTrade` operation.