Gains Network Security Review

Pashov Audit Group

Conducted by: Said, ast3ros, pontifex

July 18th 2024 - July 25th 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 - 59a031048323276bfbec6c8751ae7c0ee81df5ab

fixes review commit hash - 60cd9dde908940d1acb0b1f67803269b7e21522a

Scope

The following smart contracts were in scope of the audit:

- `GNSPairsStorage`
- `GNSPriceAggregator`
- `GNSPriceImpact`
- `GNSTradingStorage`
- `BorrowingFeesUtils`
- `ConstantsUtils`
- `PairsStorageUtils`
- `PriceAggregatorUtils`
- `PriceImpactUtils`
- `TradingCallbackUtils`
- `TradingCommonUtils`
- `TradingInteractionUtils`
- `TradingStorageUtils`
- `UpdateLeverageLifecycles`
- `DecreasePositionSizeUtils`
- `IncreasePositionSizeUtils`
- `UpdatePositionSizeLifecycles`

7. Executive Summary

Over the course of the security review, Said, ast3ros, pontifex engaged with Gains Network to review Gains Network. In this period of time a total of **6** issues were uncovered.

Protocol Summary

Protocol Name	Gains Network
Repository	https://github.com/GainsNetwork-org/gTrade-contracts
Date	July 18th 2024 - July 25th 2024
Protocol Type	Leveraged trading platform

Findings Count

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

Summary of Findings

ID	Title	Severity	Status
[H-01]	Inconsistent spread and price impact charges	High	Resolved
[M-01]	Lack of slippage protection on trade closures	Medium	Resolved
[M-02]	getLiqPnlThresholdP could revert	Medium	Resolved
[M-03]	Wrong trade price impact calculation when closing trades	Medium	Resolved
[M-04]	Traders can avoid the new liquidation configuration	Medium	Resolved
[L-01]	Users cannot control their loss level when setting stop loss	Low	Resolved

8. Findings

8.1. High Findings

[H-01] Inconsistent spread and price impact charges

Severity

Impact: Medium

Likelihood: High

Description

The protocol implements different spread and price impact charging mechanisms for trades opened before and after v9.2. However, inconsistency in these mechanisms allows users to exploit pre-v9.2 trades to avoid full spread and price impact charges when modifying their positions.

When increasing position sizes, the spread and price impact percentage are charged 50% regardless of trade opened before or after v9.2.

```
function prepareCallbackValues(
        ITradingStorage.Trade memory _existingTrade,
        ITradingStorage.Trade memory _partialTrade,
        ITradingCallbacks.AggregatorAnswer memory _answer
    ) internal view returns
      (IUpdatePositionSizeUtils.IncreasePositionSizeValues memory values) {
        ...
        // 3. Calculate price impact values
        (, values.priceAfterImpact) = _getMultiCollatDiamond
          ().getTradePriceImpact(
            TradingCommonUtils.getMarketExecutionPrice
            //(_answer.price, _answer.spreadP, _existingTrade.long, true), // @audit c
            _existingTrade.pairIndex,
            _existingTrade.long,
            _getMultiCollatDiamond().getUsdNormalizedValue(
                _existingTrade.collateralIndex,
                values.positionSizeCollateralDelta
            ),
            false,
            true,
            0
        );
        ...
    }
```

However, when reducing position size or close trade, if the trade opened before v9.2, no spread or price impact is charged.

```
function getTradeClosingPriceImpact(
        ITradingCommonUtils.TradePriceImpactInput memory _input
    ) external view returns (
      uint256priceImpactP,
      uint256priceAfterImpact,
      uint256tradeValueCollateralNoFactor
    ) external view returns
      (uint256 priceImpactP, uint256 priceAfterImpact, uint256 tradeValueCollateralNoF
        ITradingStorage.Trade memory trade = _input.trade;

        // 0. If trade opened before v9.2, return market price
        //(no closing spread or price impact)
        if (_getMultiCollatDiamond().getTradeLiquidationParams
          (trade.user, trade.index).maxLiqSpreadP == 0) {
            return (0, _input.marketPrice, 0);
        }
        ...
    }
```

This inconsistency creates an exploit:

- Users with open trades from before v9.2 can increase their position size, paying only 50% of spread and price impact.
- They can then decrease their position size or close the trade without any spread or price impact charges.

Recommendations

When increasing position sizes, charge the 100% spread and price impact percentage if the trade was opened before v9.2.

8.2. Medium Findings

[M-01] Lack of slippage protection on trade closures

Severity

Impact: Medium

Likelihood: Medium

Description

When closing trades or reducing position sizes, the execution price can deviate significantly from expected due to two main factors:

- Spread percentage update by admin.
- Price impact percentage fluctuation.

The `spreadP` and calculated `priceImpactP` used may have changed between when a trade closure is requested and when it's actually executed in the callback. However, there is currently no slippage protection mechanism to prevent trades from being closed at unexpectedly unfavorable prices due to these fluctuations.

This issue also affects the `decreasePositionSize` function in a similar way as `closeTradeMarketCallback`.

```
function closeTradeMarketCallback(
        ITradingCallbacks.AggregatorAnswer memory _a
    ) internal tradingActivatedOrCloseOnly {
        ...

        //(uint256 priceImpactP, uint256 priceAfterImpact, ) = TradingCommonUtils.getT
            ITradingCommonUtils.TradePriceImpactInput(
                t,
                _a.price,
                _a.spreadP, // @audit spreadP can be updated between closeTrade
                // and closeTradeMarketCallback
                TradingCommonUtils.getPositionSizeCollateral
                  (t.collateralAmount, t.leverage)
            )
        );
        ...
    }
```

Recommendations

- Allow users to specify a maximum acceptable slippage when requesting to close a trade.

[M-02] `getLiqPnlThresholdP` could revert

Severity

Impact: High

Likelihood: low

Description

When liquidation parameters are configured, it is possible for `startLeverage` and `endLeverage` to have the same value. This can be useful for an asset group where a constant liquidation threshold is desired, regardless of leverage.

However, when `getLiqPnlThresholdP` is called and the provided `_leverage` is equal to `startLeverage` / `endLeverage`, it will revert if `startLeverage` equal to `endLeverage` due to division by 0.

```
function getLiqPnlThresholdP(
        IPairsStorage.GroupLiquidationParams memory _params,
        uint256 _leverage
    ) public pure returns (uint256) {
        // For trades opened before v9.2, use legacy liquidation threshold
        // LEGACY_LIQ_THRESHOLD_P = 90 * P_10; // -90% pnl
        if
          (_params.maxLiqSpreadP == 0) return ConstantsUtils.LEGACY_LIQ_THRESHOLD_P;

        if
          (_leverage < _params.startLeverage) return _params.startLiqThresholdP;
        if (_leverage > _params.endLeverage) return _params.endLiqThresholdP;

        return
            _params.startLiqThresholdP -
            ((_leverage - _params.startLeverage) *
              (_params.startLiqThresholdP - _params.endLiqThresholdP)) /
>>>         (_params.endLeverage - _params.startLeverage);
    }
```

Recommendations

Consider returning `_params.startLiqThresholdP` / `_params.endLiqThresholdP` when `_leverage` is equal to `_params.startLeverage` / `_params.endLeverage`.

```
function getLiqPnlThresholdP(
        IPairsStorage.GroupLiquidationParams memory _params,
        uint256 _leverage
    ) public pure returns (uint256) {
        // For trades opened before v9.2, use legacy liquidation threshold
        // LEGACY_LIQ_THRESHOLD_P = 90 * P_10; // -90% pnl
        if
          (_params.maxLiqSpreadP == 0) return ConstantsUtils.LEGACY_LIQ_THRESHOLD_P;

-        if
- (_leverage < _params.startLeverage) return _params.startLiqThresholdP;
+        if
+ (_leverage <= _params.startLeverage) return _params.startLiqThresholdP;
-        if (_leverage > _params.endLeverage) return _params.endLiqThresholdP;
+        if (_leverage >= _params.endLeverage) return _params.endLiqThresholdP;

        return
            _params.startLiqThresholdP -
            ((_leverage - _params.startLeverage) *
              (_params.startLiqThresholdP - _params.endLiqThresholdP)) /
            (_params.endLeverage - _params.startLeverage);
    }
```

Describe your recommendation here

[M-03] Wrong trade price impact calculation when closing trades

Severity

Impact: Low

Likelihood: High

Description

When calculating the price impact for closing trades, the function `_getTradePriceImpact` adds the trade's open interest to the existing open interest, instead of subtracting it. This leads to an overestimation of the price impact when closing trades, as it incorrectly assumes that closing a position increases market impact in the same direction as opening a position.

```
function _getTradePriceImpact(
        uint256 _marketPrice,
        bool _long,
        uint256 _startOpenInterestUsd,
        uint256 _tradeOpenInterestUsd,
        uint256 _onePercentDepthUsd,
        bool _open,
        uint256 _protectionCloseFactor
    )
        internal
        pure
        returns (
            uint256 priceImpactP, // 1e10 (%)
            uint256 priceAfterImpact // 1e10
        )
    {
        if (_onePercentDepthUsd == 0) {
            return (0, _marketPrice);
        }
        priceImpactP =
            (
            //(_startOpenInterestUsd + _tradeOpenInterestUsd / 2) * _protectionCloseFa
            _onePercentDepthUsd /
            1e18 /
            2;
        ...
    }
```

Recommendations

- When closing trades, the tradeOpenInterestUsd should be subtracted from the startOpenInterestUsd.

```
priceImpactP = ((_open ? _startOpenInterestUsd + _tradeOpenInterestUsd / 2

                    : _startOpenInterestUsd - _tradeOpenInterestUsd / 2) * _prot
          _onePercentDepthUsd /
          1e18 /
          2;
```

[M-04] Traders can avoid the new liquidation configuration

Severity

Impact: Medium

Likelihood: Medium

Description

The new update introduces group liquidation parameters, where liquidation thresholds are now configurable and depend on leverage. Higher leverage results in lower liquidation thresholds, making positions more sensitive to price changes and more prone to liquidation.

```
function getLiqPnlThresholdP(
        IPairsStorage.GroupLiquidationParams memory _params,
        uint256 _leverage
    ) public pure returns (uint256) {
        // For trades opened before v9.2, use legacy liquidation threshold
        // LEGACY_LIQ_THRESHOLD_P = 90 * P_10; // -90% pnl
>>>     if
  (_params.maxLiqSpreadP == 0) return ConstantsUtils.LEGACY_LIQ_THRESHOLD_P;
        if
          (_leverage < _params.startLeverage) return _params.startLiqThresholdP;
        if (_leverage > _params.endLeverage) return _params.endLiqThresholdP;

        return
            _params.startLiqThresholdP -
            ((_leverage - _params.startLeverage) *
              (_params.startLiqThresholdP - _params.endLiqThresholdP)) /
            (_params.endLeverage - _params.startLeverage);
    }
```

When users update their position size or leverage, it will always use initial liquidation params, which means if the order is created before the update, it will always use legacy liquidation threshold.

```
// update leverage
    function _prepareCallbackValues(
        ITradingStorage.Trade memory _existingTrade,
        ITradingStorage.Trade memory _pendingTrade,
        bool _isIncrease
    ) internal view returns
      (IUpdateLeverageUtils.UpdateLeverageValues memory values) {
        if (_existingTrade.isOpen == false) return values;

        values.newLeverage = _pendingTrade.leverage;
        values.govFeeCollateral = TradingCommonUtils.getGovFeeCollateral(
            _existingTrade.user,
            _existingTrade.pairIndex,
            TradingCommonUtils.getMinPositionSizeCollateral
              (_existingTrade.collateralIndex, _existingTrade.pairIndex) /
                2 // use min fee / 2
        );
        values.newCollateralAmount =
            (
                _isIncrease

                                                    ? _existingTrade.collateralAmount - _pendingT

                                                    : _existingTrade.collateralAmount + _pendingT
            ) -
            values.govFeeCollateral;
>>>     values.liqPrice = _getMultiCollatDiamond().getTradeLiquidationPrice(
            IBorrowingFees.LiqPriceInput(
                _existingTrade.collateralIndex,
                _existingTrade.user,
                _existingTrade.pairIndex,
                _existingTrade.index,
                _existingTrade.openPrice,
                _existingTrade.long,

                                 _isIncrease ? values.newCollateralAmount : _existingT
                _isIncrease ? values.newLeverage : _existingTrade.leverage,
                true
            )
        ); // for increase leverage we calculate new trade liquidation price and
        // for decrease leverage we calculate existing trade liquidation price
    }
```

```
// increase position size
    function prepareCallbackValues(
        ITradingStorage.Trade memory _existingTrade,
        ITradingStorage.Trade memory _partialTrade,
        ITradingCallbacks.AggregatorAnswer memory _answer
    ) internal view returns
      (IUpdatePositionSizeUtils.IncreasePositionSizeValues memory values) {
        // ...
        values.newOpenPrice =
            (positionSizePlusPnlCollateral *
                uint256(_existingTrade.openPrice) +
                values.positionSizeCollateralDelta *
                values.priceAfterImpact) /

              (positionSizePlusPnlCollateral + values.positionSizeCollateralDelta);

        // 8. Calculate existing and new liq price
        values.existingLiqPrice = TradingCommonUtils.getTradeLiquidationPrice
          (_existingTrade, true);
>>>     values.newLiqPrice = _getMultiCollatDiamond().getTradeLiquidationPrice(
            IBorrowingFees.LiqPriceInput(
                _existingTrade.collateralIndex,
                _existingTrade.user,
                _existingTrade.pairIndex,
                _existingTrade.index,
                uint64(values.newOpenPrice),
                _existingTrade.long,
                values.newCollateralAmount,
                values.newLeverage,
                false
            )
        );
    }
```

Traders can exploit this by creating small, low-risk trades before the update and then adjusting the position size or leverage afterward. By doing this, they can avoid the new liquidation threshold calculation that depends on leverage and use the previously fixed legacy threshold, which is more desirable for them.

Recommendations

When users update trades and calculate the new liquidation price, update the trades and apply the liquidation configuration if they are from legacy trades.

8.3. Low Findings

[L-01] Users cannot control their loss level when setting stop loss

When users set a stop loss, they expect their positions to be closed at or near the specified price level. However, the current implementation can potentially result in an execution price that is significantly worse than the user-specified stop loss level. It's because the `priceAfterImpact` calculation, which includes both spread and price impact, is applied after the stop loss is triggered.

It's recommended to allow users to set stop loss as the price after close spread and price impact calculation.

```
function executeTriggerCloseOrderCallback(
        ITradingCallbacks.AggregatorAnswer memory _a
    ) internal tradingActivatedOrCloseOnly {
        ...
            if (o.orderType != ITradingStorage.PendingOrderType.LIQ_CLOSE) {

                //(, uint256 priceAfterImpact, ) = TradingCommonUtils.getTradeClosingP
                    ITradingCommonUtils.TradePriceImpactInput(
                        t,
                        v.executionPrice,
                        _a.spreadP,
                        TradingCommonUtils.getPositionSizeCollateral
                          (t.collateralAmount, t.leverage)
                    )
                );
                v.executionPrice = priceAfterImpact;
            }
        ...
    }
```