Ostium Security Review

Pashov Audit Group

Conducted by: Said, eeyore, saksham

January 21st 2025 - January 28th 2025

Contents

1. About Pashov Audit Group 2

2. Disclaimer 2

3. Introduction 2

4. About Ostium 3

5. Risk Classification 3

5.1. Impact 3

5.2. Likelihood 4

5.3. Action required for severity levels 4

6. Security Assessment Summary 5

7. Executive Summary 6

8. Findings 8

8.1. High Findings 8

[H-01] Undervalued funding fee when accFundingRate is negative 8

8.2. Medium Findings 10

[M-01] Missing slippage protection in trade closure 10

[M-02] updateTp not validating and correcting the TP 10

[M-03] Lack of maximum opening fee check could cause issues 11

[M-04] The correctTp() returns overinflated values 13

[M-05] Users can prevent oracle fee charging 15

8.3. Low Findings 17

[L-01] Lack of minimum collateral amount could cause issue 17

[L-02] Removing listed pair will disallow closing trades 19

[L-03] Missing validation for MAX_TRADE_SIZE_REF 20

[L-04] An invariant can be broken 20

[L-05] New PairFundingFeesV2 parameters are not validated 21

[L-06] Oracle fee is not refunded in case of timeout 21

[L-07] Incorrect collateral value passed 21

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 **0xOstium/smart-contracts** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Ostium

Ostium is a platform built on Arbitrum that enables onchain perpetual trading of Real World Assets (RWA) through virtual price exposure, avoiding the need for tokenization. It uses a dual-vault liquidity model, where a Liquidity Buffer and Market Making Vault work together to manage trader PnL and Open Interest imbalances, with high-speed oracles providing real-time pricing. The system automates liquidations and order execution through external services, while its risk-adjusted fee structure balances exposure and encourages market stability.

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 - e8d0b546bc420cd31e280ea0ff901e8514fcc0f4

fixes review commit hash - ee3640b7da1d44f880b58e6b0ae850b5cbd41788

Scope

The following smart contracts were in scope of the audit:

- `OstiumLockedDepositNft`
- `OstiumOpenPnl`
- `OstiumPairInfos`
- `OstiumPairsStorage`
- `OstiumPriceRouter`
- `OstiumPriceUpKeep`
- `OstiumPrivatePriceUpKeep`
- `OstiumRegistry`
- `OstiumTimelockManager`
- `OstiumTimelockOwner`
- `OstiumTradesUpKeep`
- `OstiumTrading`
- `OstiumTradingCallbacks`
- `OstiumTradingStorage`
- `OstiumVault`
- `OstiumVerifier`
- `ChainUtils`
- `TradingCallbacksLib`
- `Delegatable`

7. Executive Summary

Over the course of the security review, Said, eeyore, saksham engaged with Ostium to review Ostium. In this period of time a total of **13** issues were uncovered.

Protocol Summary

Protocol Name	Ostium
Repository	https://github.com/0xOstium/smart-contracts
Date	January 21st 2025 - January 28th 2025
Protocol Type	RWA Perpetual Trading

Findings Count

Severity	**Amount**
High	1
Medium	5
Low	7
Total Findings	**13**

Summary of Findings

ID	Title	Severity	Status
[H-01]	Undervalued funding fee when accFundingRate is negative	High	Resolved
[M-01]	Missing slippage protection in trade closure	Medium	Acknowledged
[M-02]	updateTp not validating and correcting the TP	Medium	Resolved
[M-03]	Lack of maximum opening fee check could cause issues	Medium	Acknowledged
[M-04]	The correctTp() returns overinflated values	Medium	Resolved
[M-05]	Users can prevent oracle fee charging	Medium	Resolved
[L-01]	Lack of minimum collateral amount could cause issue	Low	Acknowledged
[L-02]	Removing listed pair will disallow closing trades	Low	Resolved
[L-03]	Missing validation for MAX_TRADE_SIZE_REF	Low	Resolved
[L-04]	An invariant can be broken	Low	Resolved
[L-05]	New PairFundingFeesV2 parameters are not validated	Low	Resolved
[L-06]	Oracle fee is not refunded in case of timeout	Low	Acknowledged
[L-07]	Incorrect collateral value passed	Low	Acknowledged

8. Findings

8.1. High Findings

[H-01] Undervalued funding fee when accFundingRate is negative

Severity

Impact: Medium

Likelihood: High

Description

The funding fee accounting differs between when Longs pay to Shorts and when Shorts pay to Longs, and Shorts always pay less funding fee than they should when oiDelta is negative.

This issue arises from the fact that, regardless of which side pays, the scaling is always applied to the `valueShort` accounting. As such, Shorts always receive the full funding fee from Longs but only pay a partial funding fee to Longs.

Consider this:

- oiDelta is positive:

Longs pay Shorts the full `accFundingRate`. The full `accFundingRate` is correctly added to `accPerOiLong`, and `accFundingRate` scaled by `* openInterestLong / openInterestShort` is added to `accPerOiShort`. This scaling is necessary to correctly reflect the per-token amount paid to Shorts.

- oiDelta is negative:

Shorts should pay Longs the full `accFundingRate`, but the full `accFundingRate` is incorrectly added to `accPerOiLong` which in fact is undervalued, where it should be scaled by `* openInterestShort /`

`openInterestLong`. Similarly, `accFundingRate` scaled by `* openInterestLong / openInterestShort` is added to `accPerOiShort` which result in less fees being paid by Shorts, where it should not be reduced like this, and the full `accFundingRate` should be added to `accPerOiShort`.

As can be seen, when oiDelta is negative, the funding fee paid to Longs is undervalued by the size of oiDelta.

Recommendations

Correct this discrepancy by properly accounting for the funding fee when oiDelta is negative.

8.2. Medium Findings

[M-01] Missing slippage protection in trade closure

Severity

Impact: High

Likelihood: Low

Description

The closeTradeMarketCallback function (OstiumTradingCallback.sol) lacks slippage protection for the amount of collateral a trader receives when closing their position. When a trade closure is fulfilled (via a trusted forwarder), market conditions might be unfavorable compared to when the closure was initiated, resulting in the trader receiving significantly less collateral than expected. This is particularly important since the fulfillment can happen at any time within the allowed window, and the executing forwarder (willingly or unwillingly) might process the closure during adverse market conditions.

```
function closeTradeMarketCallback(
    bytes32[] calldata priceData,
    Trade memory trade,
    TradeInfo memory tradeInfo
) {
    // Calculates collateral to return
    uint256 collateralToReturn = getTradeValue(percentProfit, trade.collateral);
    // Transfers funds without minimum amount verification
}
```

Recommendations

Add a minCollateralToReceive parameter to the closeTradeMarketCallback function

[M-02] `updateTp` not validating and correcting the TP

Severity

Impact: Low

Likelihood: High

Description

When `updateTp` is called, the provided TP value is not validated to ensure it is appropriate relative to the `openPrice` and the trade position (`buy`). Additionally, the TP is not adjusted to ensure it does not exceed the allowed maximum gain.

```
function updateTp
      (uint16 pairIndex, uint8 index, uint192 newTp) external notDone {
        address sender = _msgSender();
        IOstiumTradingStorage storageT = IOstiumTradingStorage
          (registry.getContractAddress('tradingStorage'));

        if (!checkNoPendingTrigger
          (sender, pairIndex, index, IOstiumTradingStorage.LimitOrder.TP)) {
            revert TriggerPending(sender, pairIndex, index);
        }

        IOstiumTradingStorage.Trade memory t = storageT.getOpenTrade
          (sender, pairIndex, index);

        if (t.leverage == 0) {
            revert NoTradeFound(sender, pairIndex, index);
        }

        storageT.updateTp(sender, pairIndex, index, newTp);

        emit TpUpdated(storageT.getOpenTradeInfo
          (sender, pairIndex, index).tradeId, sender, pairIndex, index, newTp);
    }
```

Recommendations

Consider validating the provided tp and call `TradingCallbacksLib.correctTp` to adjust the provided tp.

[M-03] Lack of maximum opening fee check could cause issues

Severity

Impact: Medium

Likelihood: Medium

Description

When new trade is executed and registered, opening fees will be deducted from users collateral.

```
function registerTrade(
      uint256tradeId,
      IOstiumTradingStorage.Tradememorytrade,
      uint256latestPrice
    )
        private
        returns (IOstiumTradingStorage.Trade memory)
    {
        // ...

        // 2.1 Charge opening fee
        {
>>>         (uint256 reward, uint256 vaultReward) = storageT.handleOpeningFees(

                                   trade.pairIndex, latestPrice, trade.collateral * trad
            );

            trade.collateral -= reward;

            emit DevFeeCharged(tradeId, trade.trader, reward);

            if (vaultReward > 0) {
                IOstiumVault vault = IOstiumVault(registry.getContractAddress
                  ('vault'));
                storageT.transferUsdc(address(storageT), address
                  (this), vaultReward);
                vault.distributeReward(vaultReward);
                trade.collateral -= vaultReward;
                emit VaultOpeningFeeCharged(tradeId, trade.trader, vaultReward);
            }

            // 2.2 Charge the oracle fee
            reward = storageT.handleOracleFees(trade.pairIndex, true);

            trade.collateral -= reward;

            emit OracleFeeCharged(tradeId, trade.trader, reward);
        }
       // ...
    }
```

It can be seen that `handleOpeningFees` depends on trades opened by other users, which may differ between the trader's open request and the actual trade execution. This could cause traders to pay unexpectedly high fees due to trading size fluctuations from other traders.

```
function handleOpeningFees(
        uint16 _pairIndex,
        uint256 latestPrice,
        uint256 _leveragedPositionSize,
        uint32 leverage,
        bool isBuy
    ) external onlyCallbacks returns (uint256 devFee, uint256 vaultFee) {
        uint256 oiCap = openInterest[_pairIndex][2];

                 uint256 oiLong = openInterest[_pairIndex][0] * latestPrice / PRECISIO

                 uint256 oiShort = openInterest[_pairIndex][1] * latestPrice / PRECISI

        uint256 openInterestMax = oiLong > oiShort ? oiLong : oiShort;
        // @audit - the logic is incorrect here?
        oiCap = openInterestMax > oiCap ? openInterestMax : oiCap;

        int256 oiDelta = oiLong.toInt256() - oiShort.toInt256();
        uint256 usageOi = isBuy ? oiLong : oiShort;

        (devFee, vaultFee) = IOstiumPairInfos(registry.getContractAddress
          ('pairInfos')).getOpeningFee(
            _pairIndex,
            isBuy ? _leveragedPositionSize.toInt256
              () : -_leveragedPositionSize.toInt256(),
            leverage,
            oiDelta,
            oiCap,
            usageOi
        );

        devFees += devFee;
    }
```

Recommendations

Consider allowing users to specify the maximum fee they expect to pay from their collateral.

[M-04] The `correctTp()` returns overinflated values

Severity

Impact: Medium

Likelihood: Medium

Description

When a user adds collateral to their position, the `leverage` parameter is updated and becomes lower than the `initialLeverage` parameter. Due to the requirement that maximum profits must not exceed 900% of the initial collateral from which position was create, the maximum profit percentage value returned from `_currentPercentProfit()` is scaled down when new collateral is added:

```
p = p * leverage / leverageToUse;
```

This creates a situation where, after additional collateral is added, the maximum percentage value returned from `_currentPercentProfit()` will never reach `900%`, even in cases where calculations based on `initialLeverage` exceed 900% by a large margin. This is due to limiting the maximum percentage value to `900%` before scaling it with the new `leverage` value:

```
p = p > maxPnlP ? maxPnlP : p;
```

The combination of these updates causes the `correctTp()` function to miscalculate the `tp`. Specifically, the requirement in the `if()` check is never met, leading to an overinflated `tp` compared to the real maximum gains a user position can achieve.

Due to this incorrect calculation, positions with added collateral will not close at maximum profit as expected via `TP` limit order. Instead, the user will need to manually close the position with market order after the maximum gain from the initial collateral is reached, which can lead to inefficiencies and missed opportunities.

A similar issue occurs when the value is retrieved from the `currentPercentProfit()` view function.

Recommendations

Introduce a scaling factor to the `if()` check in the `correctTp()` function to correctly account for the adjusted leverage:

```
uint256 maxGain = int16(MAX_GAIN_P) * int32
          (PRECISION_6) * leverage / initialLeverage;
        if (
            tp == 0
                || _currentPercentProfit(openPrice.toInt256(), tp.toInt256
                  (), buy, int32(leverage), int32(initialLeverage))
-                   == int16(MAX_GAIN_P) * int32(PRECISION_6)
+                   >= maxGain
        ) {
```

[M-05] Users can prevent oracle fee charging

Severity

Impact: Medium

Likelihood: Medium

Description

The issue arises in the `closeTradeMarketCallback()` function because, in the case of a full close (`closePercentage == 100e2`) and , the `MARKET_CLOSED` is the cancellation reason, trader is charged half of the oracle fee.

```
} else if (closePercentage == 100e2) {
                // Charge only half
                uint256 oracleFee = storageT.handleOracleFees
                  (t.pairIndex, false);

                storageT.transferUsdc(t.trader, address(storageT), oracleFee);

                emit OracleFeeCharged(a.orderId, t.trader, oracleFee);
            }
```

This is problematic because the function executions inside the `OstiumTradingCallbacks` contract should never depend on user actions such as `USDC` allowance. Users are expected to approve only specific amounts for any contract, and they should not grant open-ended allowances to smart contracts.

In the case of `closeTradeMarketCallback()`, if `MARKET_CLOSED` is the cancellation reason, the function will likely revert due to insufficient allowance when attempting to deduct the small oracle fee.

This situation could be intentional on the part of the trader (e.g., deliberately limiting the allowance) or simply due to an unexpected lack of allowance. Regardless, functions within `OstiumTradingCallbacks` contract should not depend on any user action that can lead to unexpected revert.

Recommendations

The oracle fee should either:

1. **Never be charged** in the event of a cancellation reason, or
2. **Be charged upfront** to eliminate dependency on user-controlled allowance.

8.3. Low Findings

[L-01] Lack of minimum collateral amount could cause issue

When opening a trade (`openTrade`), it doesn't validate the collateral or remaining collateral against the minimum value, as long as the leveraged position size is still greater than the configured minimum position.

```
function openTrade(
        IOstiumTradingStorage.Trade calldata t,
        IOstiumTradingStorage.OpenOrderType orderType,
        uint256 slippageP // for market orders only
    ) external notDone notPaused pairIndexListed(t.pairIndex) {
        // ...

        if (
            t.leverage == 0 || t.leverage < pairsStored.pairMinLeverage
              (t.pairIndex)
                || t.leverage > pairsStored.pairMaxLeverage(t.pairIndex)
        ) revert WrongLeverage(t.leverage);

        if (t.collateral > maxAllowedCollateral) {
            revert AboveMaxAllowedCollateral();
        }

>>>     if (t.collateral * t.leverage / 100 < pairsStored.pairMinLevPos
  (t.pairIndex)) {
            revert BelowMinLevPos();
        }

        if (t.tp != 0 && (t.buy ? t.tp <= t.openPrice : t.tp >= t.openPrice)) {
            revert WrongTP();
        }

        if (t.sl != 0 && (t.buy ? t.sl >= t.openPrice : t.sl <= t.openPrice)) {
            revert WrongSL();
        }

        storageT.transferUsdc(sender, address(storageT), t.collateral);
        // ...
    }
```

This could cause issues. For instance, when `registerTrade` is triggered, it calculates `vaultReward` and `reward`, which are based on the leveraged position size, as well as oracle fees, which could potentially be greater than the collateral.

```
function registerTrade(
      uint256tradeId,
      IOstiumTradingStorage.Tradememorytrade,
      uint256latestPrice
    )
        private
        returns (IOstiumTradingStorage.Trade memory)
    {
        IOstiumTradingStorage storageT = IOstiumTradingStorage
          (registry.getContractAddress('tradingStorage'));
        IOstiumPairInfos pairInfos = IOstiumPairInfos
          (registry.getContractAddress('pairInfos'));

        // 2.1 Charge opening fee
        {
            (uint256 reward, uint256 vaultReward) = storageT.handleOpeningFees(

                                 trade.pairIndex, latestPrice, trade.collateral * trad
            );

>>>         trade.collateral -= reward;

            emit DevFeeCharged(tradeId, trade.trader, reward);

            if (vaultReward > 0) {
                IOstiumVault vault = IOstiumVault(registry.getContractAddress
                  ('vault'));
                storageT.transferUsdc(address(storageT), address
                  (this), vaultReward);
                vault.distributeReward(vaultReward);
>>>             trade.collateral -= vaultReward;
                emit VaultOpeningFeeCharged(tradeId, trade.trader, vaultReward);
            }

            // 2.2 Charge the oracle fee
            reward = storageT.handleOracleFees(trade.pairIndex, true);
>>>         trade.collateral -= reward;

            emit OracleFeeCharged(tradeId, trade.trader, reward);
        }
        // ..
    }
```

Or when market is closed and open trade is executed, it will revert when `oracleFees` greater than `trade.collateral`.

```
function openTradeMarketCallback
      (IOstiumPriceUpKeep.PriceUpKeepAnswer calldata a) external notDone {
        // ...

        if (cancelReason == CancelReason.NONE) {
            trade = registerTrade(a.orderId, trade, uint192(a.price));
            uint256 tradeNotional = storageT.getOpenTradeInfo
              (trade.trader, trade.pairIndex, trade.index).oiNotional;
            IOstiumOpenPnl(registry.getContractAddress
              ('openPnl')).updateAccTotalPnl(

                                 a.price, trade.openPrice, 0, tradeNotional, trade.pai
            );
            emit MarketOpenExecuted
              (a.orderId, trade, priceImpactP, tradeNotional);
        } else {
            // Charge only half
            uint256 oracleFees = storageT.handleOracleFees
              (trade.pairIndex, false);
>>>         storageT.transferUsdc(address
  (storageT), trade.trader, trade.collateral - oracleFees);

            emit OracleFeeCharged(a.orderId, trade.trader, oracleFees);
            emit MarketOpenCanceled
              (a.orderId, trade.trader, trade.pairIndex, cancelReason);
        }
        storageT.unregisterPendingMarketOrder(a.orderId, true);
    }
```

Consider to add minimum collateral check when opening new trade.

[L-02] Removing listed pair will disallow closing trades

When a pair index is removed, traders can still request `closeTradeMarket`, `topUpCollateral`, and `removeCollateral` for existing open trades using the removed pair. However, when the trade is executed, it will be reverted when trying to update the group collateral.

```
function updateGroupCollateral(
      uint16_pairIndex,
      uint256_amount,
      bool_long,
      bool_increase
    ) external {
        if (
            msg.sender != registry.getContractAddress('callbacks')
                && msg.sender != registry.getContractAddress('trading')
        ) revert NotAuthorized(msg.sender);

>>>     if (!isPairIndexListed[_pairIndex]) revert PairNotListed(_pairIndex);


                  uint256[2] storage collateralOpen = groupsCollaterals[pairs[_pairInde
        uint256 index = _long ? 0 : 1;

        if (_increase) {
            collateralOpen[index] += _amount;
        } else {

                          collateralOpen[index] = collateralOpen[index] > _amount ? col
        }
    }
```

This will prevent currently open trades using the removed pair index from being settled.

Consider implementing functionality to settle open trades from the removed pair index.

[L-03] Missing validation for `MAX_TRADE_SIZE_REF`

The protocol defines `MAX_TRADE_SIZE_REF` as a constant but lacks validation checks when setting trade size references. This could lead to scenarios where trade size references exceed the intended maximum limit, potentially affecting price impact calculations.

```
// Constant defined but not enforced
uint256 constant MAX_TRADE_SIZE_REF = 10000000e6; // 10M
```

Add validation check when setting trade size reference and validate the value returned from the chainlink report.

[L-04] An invariant can be broken

In situations where `liqThresholdP` is updated in the `setLiqThresholdP()` function, the `maxNegativePnlOnOpenP` is not rechecked to ensure it remains below the new value.

This can lead to the `maxNegativePnlOnOpenP < liqThresholdP` invariant being broken. In extreme situations involving large price impacts, this could result in the creation of liquidatable positions.

Consider rechecking the `maxNegativePnlOnOpenP` in the `setLiqThresholdP()` function as well.

[L-05] New `PairFundingFeesV2` parameters are not validated

During the initialization of new funding fee parameters in the `initializeV2()` function, these values are not sanity-checked against the given maximum values.

Since a sanity check is performed in other setter functions when the `PairFundingFeesV2` parameters are updated, consider adding proper validation to the `initializeV2()` function as well.

[L-06] Oracle fee is not refunded in case of timeout

When a user creates a pending partial close order, the order may time out. However, in the case of a partial close, an additional oracle fee is charged in advance for the operation.

If such an order times out and the user closes the pending market order without the oracle being used via the `closeTradeMarketTimeout()` function, the oracle fee should be refunded, provided it is not a `retry == true` call.

[L-07] Incorrect collateral value passed

In both the `openTradeMarketCallback()` and `executeAutomationOpenOrderCallback()` functions, the collateral value passed to cancellation reason checks is not yet reduced by the `open` and `oracle` fees. This occurs because the cancellation checks are performed before the `registerTrade()` function call.

As a result, the `withinExposureLimits()` check is always performed on an inflated value. In extreme situations, this can lead to the rejections of orders that actually fits into the `tradingStorage.openInterest(pairIndex, 2)` or `pairsStorage.groupMaxCollateral(pairIndex)` limits.