Clober DEX Audit Report

Prepared by KupiaSec

Version 1.2

Auditors

KupiaSec

Nov 28, 2024

Contents

1 About KupiaSec **2**

2 Disclaimer **2**

3 Risk Classification **2**

4 Protocol Summary **2**

5 Audit Scope **2**

6 Executive Summary **3**

7 Findings **4**

7.1 Medium Risk . . . 4

7.1.1 The tick price check in the `SimpleOracleStrategy.updatePosition()` function does not take fee rates into account. . . . 4

7.2 Low Risk . . . 5

7.2.1 Missing oracle price validation in `updatePosition` function leading to potential invalid position updates . . . 5

7.3 Informational . . . 7

7.3.1 The process of opening a pool does not ensure that `quote != base`. . . . 7

1 About KupiaSec

KupiaSec is a team of Web3 security experts that operates with transparency and a meritocratic spirit.

KupiaSec executes the modified MPA model for the Private Audits, a.k.a. **Solo Audit by a Lead + Internal Competition + Mitigation Review**.

- Solo Audit by a Lead

  KupiaSec assigns a senior auditor as a Lead Auditor based on the protocol category. The Lead Auditor is responsible for the first phase and will be the main point of contact for the client. The Lead Auditor shares the analysis and findings with the team.

- Internal Competition

  KupiaSec assigns 5~7 assist auditors to conduct the second phase. The auditors compete to find the most issues and the best solutions. This phase ensures the protocol goes through a rigorous review process by "many eyes" in a competitive environment.

- Mitigation Review

  After the protocol team has fixed the issues, KupiaSec conducts a final review to ensure all the issues are resolved.

2 Disclaimer

The KupiaSec team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

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

4 Protocol Summary

Clober presents a new algorithm for order book DEX “LOBSTER - Limit Order Book with Segment Tree for Efficient oRder-matching” that enables on-chain order matching and settlement on decentralized smart contract platforms. With Clober, market participants can place limit and market orders in a fully decentralized, trustless way at a manageable cost.

5 Audit Scope

- src/Rebalancer.sol
- src/SimpleOracleStrategy.sol

Summary of Commits

Project Name	Clober DEX
Repository	clober-rebalancer
Initial Commit	00b8d17287c6...
Mitigation Commit	2a477db700ba...

6 Executive Summary

KupiaSec executed a modified Multi-Phase Audit model, a.k.a. **Solo Audit by a Lead + Internal Competition + Mitigation Review**.

@kennedy1030 conducted the audit as the Lead Auditor and 5 auditors competed in the second phase.

Execution Timeline

Phase-1: Audit by a Lead	Nov 12 - Nov 13
Phase-2: Internal Competition	Nov 13 - Nov 16
Initial Report Delivery	Nov 28, 2024
Phase-3: Mitigation Review	Nov 28
Final Report Delivery	Nov 28, 2024

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	1
Informational	1
Gas Optimizations	0
Total Issues	3

Summary of Findings

[M-1] The tick price check in the `SimpleOracleStrategy.updatePosition()` function does not take fee rates into account.	Open
[L-1] Missing oracle price validation in `updatePosition` function leading to potential invalid position updates	Open
[I-1] The process of opening a pool does not ensure that `quote != base`.	Open

7 Findings

7.1 Medium Risk

7.1.1 The tick price check in the `SimpleOracleStrategy.updatePosition()` function does not take fee rates into account.

Description: The `updatePosition()` function requires only that priceA < priceB, which is insufficient.

While `priceB - priceA` represents the trading profit, fees are deducted during the trading. Therefore, the profit must exceed the fee rates.

```
    function updatePosition(bytes32 key, uint256 oraclePrice, Tick tickA, Tick tickB, uint24 rate)
        external
        onlyOperator
    {
        uint256 priceA = tickA.toPrice();
        uint256 priceB = Tick.wrap(-Tick.unwrap(tickB)).toPrice();

230     if (priceA >= priceB) revert InvalidPrice();
        if (rate > RATE_PRECISION) revert InvalidValue();

        Config memory config = _configs[key];
        if (
            oraclePrice * (RATE_PRECISION + config.priceThresholdA) / RATE_PRECISION < priceA
                || oraclePrice * (RATE_PRECISION - config.priceThresholdB) / RATE_PRECISION > priceB
        ) revert ExceedsThreshold();

        (BookId bookIdA,) = rebalancer.getBookPairs(key);

        IBookManager.BookKey memory bookKeyA = bookManager.getBookKey(bookIdA);
        uint8 decimalsA = _getCurrencyDecimals(bookKeyA.quote);
        uint8 decimalsB = _getCurrencyDecimals(bookKeyA.base);

        // @dev Convert oracle price to the same decimals as the reference oracle
        oraclePrice = oraclePrice * 10 ** decimalsB / 10 ** decimalsA;
        oraclePrice = (oraclePrice * 10 ** referenceOracle.decimals()) >> 96;

        Position memory position = _positions[key];
        position.oraclePrice = SafeCast.toUint128(oraclePrice);
        position.tickA = tickA;
        position.tickB = tickB;
        position.rate = rate;

        _positions[key] = position;
        delete _lastRawAmounts[key];
        emit UpdatePosition(key, oraclePrice, tickA, tickB, rate);
    }
```

Impact: Users may experience losses during the rebalancing process.

Proof Of Concept:

Recommended Mitigation: There should be a proper check to ensure that `priceB - priceA` can cover the fees.

Client: Confirmed. Fixed in PR

KupiaSec: Verified.

7.2 Low Risk

7.2.1 Missing oracle price validation in `updatePosition` function leading to potential invalid position updates

Description: The `updatePosition` function does not validate the oracle price before updating the position, which could lead to the contract accepting an invalid oracle price if the price is outside the acceptable range. This could result in position updates with invalid prices, allowing incorrect or unintended trading strategies to be executed based on an invalid price.

Impact: The lack of oracle price validation in the `updatePosition` function could allow invalid prices to be used, leading to incorrect or malicious position updates and potentially causing the `rebalance` function to revert due to failed price checks in the `computeOrders` function.

Proof Of Concept:

1. A position with a specific `key` is updated using the `updatePosition` function.
2. The oracle price provided in the function call may be outside the valid range, but no check is performed before the position is updated.
3. The system could accept and store the invalid price, potentially allowing the position to be updated incorrectly.

For example:

- If the oracle price deviates significantly from the expected range due to some error or manipulation, the position will still be updated, allowing for potentially malicious price manipulation.

Recommended Mitigation: Add the missing `_isOraclePriceValid` check to the `updatePosition` function to ensure that the provided oracle price is within the acceptable range before updating the position.

```
function updatePosition(
    bytes32 key,
    uint256 oraclePrice,
    Tick tickA,
    Tick tickB,
    uint24 rate
) external onlyOperator {
    uint256 priceA = tickA.toPrice();
    uint256 priceB = Tick.wrap(-Tick.unwrap(tickB)).toPrice();

    if (priceA >= priceB) revert InvalidPrice();
    if (rate > RATE_PRECISION) revert InvalidValue();

+    Config memory config = _configs[key];

    // Check if the oracle price is valid before updating the position
+    if (!_isOraclePriceValid(oraclePrice, config.referenceThreshold, tickA.quote, tickA.base)) {
+        revert InvalidOraclePrice();
+    }

    if (
        oraclePrice * (RATE_PRECISION + config.priceThresholdA) / RATE_PRECISION < priceA
            || oraclePrice * (RATE_PRECISION - config.priceThresholdB) / RATE_PRECISION > priceB
    ) revert ExceedsThreshold();

    (BookId bookIdA,) = rebalancer.getBookPairs(key);
    IBookManager.BookKey memory bookKeyA = bookManager.getBookKey(bookIdA);
    uint8 decimalsA = _getCurrencyDecimals(bookKeyA.quote);
    uint8 decimalsB = _getCurrencyDecimals(bookKeyA.base);

    // Convert oracle price to the same decimals as the reference oracle
    oraclePrice = oraclePrice * 10 ** decimalsB / 10 ** decimalsA;
    oraclePrice = (oraclePrice * 10 ** referenceOracle.decimals()) >> 96;
```

```
    Position memory position = _positions[key];
    position.oraclePrice = SafeCast.toUint128(oraclePrice);
    position.tickA = tickA;
    position.tickB = tickB;
    position.rate = rate;

    _positions[key] = position;
    delete _lastRawAmounts[key];
    emit UpdatePosition(key, oraclePrice, tickA, tickB, rate);
}
```

Client: Fixed in PR

KupiaSec: Verified.

7.3 Informational

7.3.1 The process of opening a pool does not ensure that `quote != base`.

Description: In the `_open()` function, there is no check to ensure that `bookKeyA.quote` is not equal to `bookKeyA.base`.

```
    function _open(
        IBookManager.BookKey calldata bookKeyA,
        IBookManager.BookKey calldata bookKeyB,
        bytes32 salt,
        address strategy
    ) public selfOnly returns (bytes32 key) {
        if (!(bookKeyA.quote.equals(bookKeyB.base) && bookKeyA.base.equals(bookKeyB.quote))) revert
        ↪  InvalidBookPair();
        if (address(bookKeyA.hooks) != address(0) || address(bookKeyB.hooks) != address(0)) revert
        ↪  InvalidHook();
        if (strategy == address(0)) revert InvalidStrategy();

        BookId bookIdA = bookKeyA.toId();
        BookId bookIdB = bookKeyB.toId();
        if (!bookManager.isOpened(bookIdA)) bookManager.open(bookKeyA, "");
        if (!bookManager.isOpened(bookIdB)) bookManager.open(bookKeyB, "");

        key = _encodeKey(bookIdA, bookIdB, salt);
        if (_pools[key].strategy != IStrategy(address(0))) revert AlreadyOpened();

        _pools[key].bookIdA = bookIdA;
        _pools[key].bookIdB = bookIdB;
        _pools[key].strategy = IStrategy(strategy);
        bookPair[bookIdA] = bookIdB;
        bookPair[bookIdB] = bookIdA;

        emit Open(key, bookIdA, bookIdB, salt, strategy);
    }
```

Impact: Proof Of Concept:

Recommended Mitigation: Verify that `bookKeyA.quote` is not equal to `bookKeyA.base`.

```
+       if (bookKeyA.quote.equals(bookKeyA.base)) revert InvalidBookPair();
```

Client:

KupiaSec: