Zellic

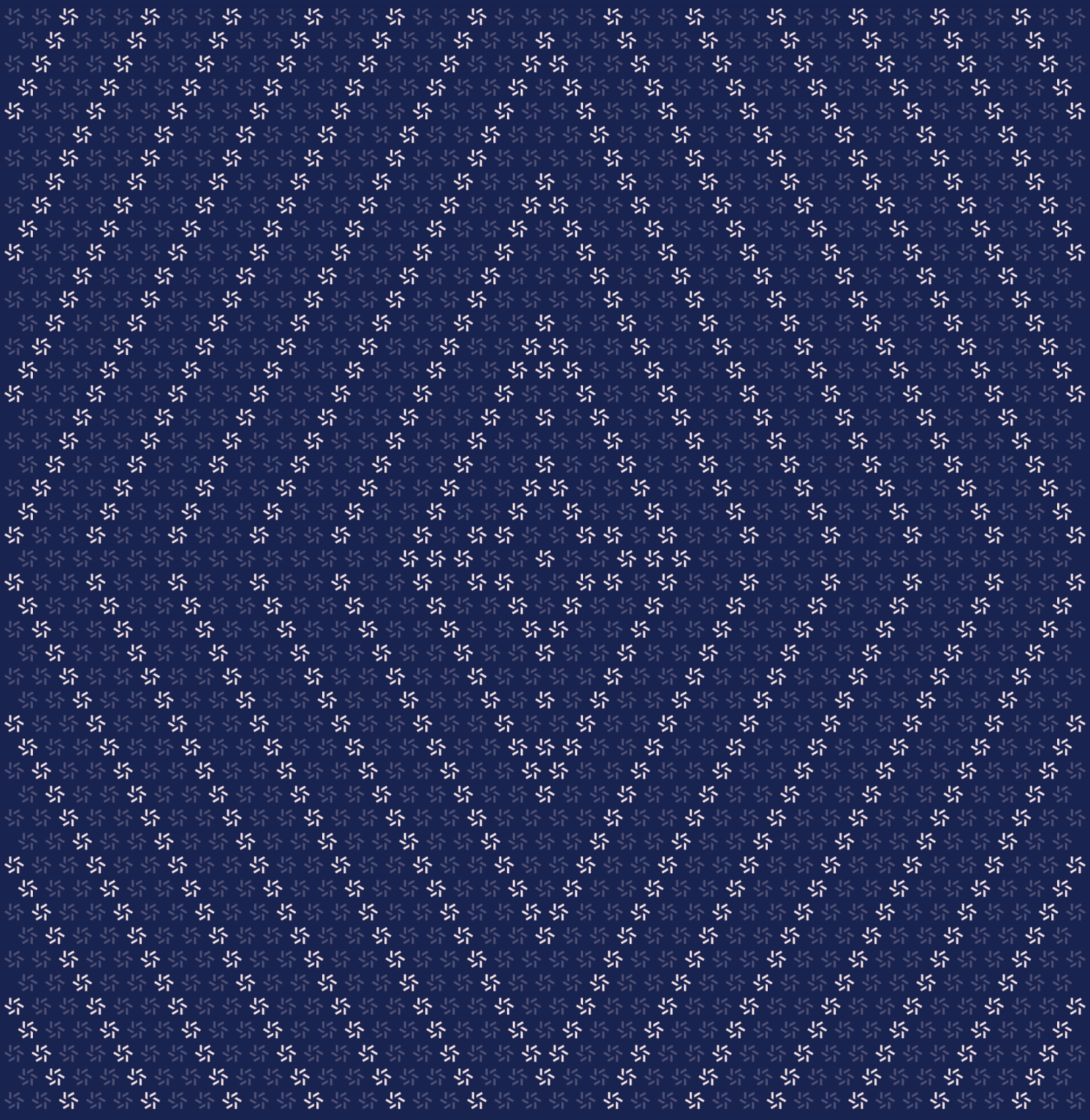
Prepared for
Liquid Labs Global, Inc.

Prepared by
Katerina Belotskaia
Juchang Lee
Zellic

November 21, 2024

GTE CLOB

Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Liquid Labs Global, Inc. from November 7th to November 15th, 2024. During this engagement, Zellic reviewed GTE CLOB's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Is the fee calculation implemented correctly?
- Are fee deductions made on time?
- Could a malicious attacker make users lose tokens?
- Could a malicious attacker trigger a lock-up of user funds?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody
- Implementation of red-black tree (RBT)

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped GTE CLOB contracts, we discovered three findings. No critical issues were found. One finding was of high impact and two were of medium impact.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Liquid Labs Global, Inc. in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	1
Medium	2
Low	0
Informational	0

2. Introduction

2.1. About GTE CLOB

Liquid Labs Global, Inc. contributed the following description of GTE CLOB:

> GTE core is a simple CLOB implementation for the EVM, specifically designed to be deployed on the upcoming MegaETH network. It is a fairly bare-bones market with operator approvals for more complex periphery actions and extensibility in the broader MegaETH ecosystem. The goal of GTE core is to have a CEX-like feel.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case

basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

GTE CLOB Contracts

Type	Solidity
Platform	EVM-compatible

Target	
Repository	https://github.com/liquid-labs-inc/gte-clob-v1 ↗
Version	`1ce31351416f4a9b1857b950f2bc190e414b8b0b`

Programs	`CLOB.sol`
	`CLOBFactory.sol`
	`interface/*.sol`
	`types/*.sol`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 2.1 person-weeks. The assessment was conducted by two consultants over the course of one calendar week.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io ↗

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Katerina Belotskaia
Engineer
kate@zellic.io ↗

Juchang Lee
Engineer
lee@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

Nov 7, 2024	Kick-off call
Nov 7, 2024	Start of primary review period
Nov 15, 2024	End of primary review period

3. Detailed Findings

3.1. The `createMarket` function is permissionless

Target	CLOBFactory		
Category	Coding Mistakes	**Severity**	High
Likelihood	High	**Impact**	High

Description

The `createMarket` function in the CLOBFactory contract is currently permissionless, allowing any user to create a market with arbitrary `quoteToken` and `baseToken` pairs.

```
function createMarket(
        address quoteToken,
        address baseToken,
        ConfigParams memory config,
        SettingsParams memory settings
    ) external returns (address) {
    _assertValidTokenPair(quoteToken, baseToken);

    uint256 quoteDecimals = IERC20Metadata(quoteToken).decimals();
    uint256 baseDecimals = IERC20Metadata(baseToken).decimals();
    [...]
    }
}
```

Impact

This behavior does not align with the stated requirements, where market creation should be restricted to the contract owner.

Recommendations

Add an `onlyOwner` modifier to the `createMarket` function to ensure that only the factory owner can create markets.

Remediation

This issue has been acknowledged by Liquid Labs Global, Inc., and a fix was implemented in commit 8b11f544 ↗.

3.2. Revoked operators still retain user approvals

Target	CLOBFactory		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Low	**Impact**	Medium

Description

The protocol allows users to delegate market functionalities such as `deposit`, `withdraw`, `postLimitOrder`, `postFillOrder`, `reduce`, and `cancel` to operators acting on their behalf. The `approveOperator` function in the CLOBFactory contract enables users to approve new operators. However, only operators approved by the owner of the CLOBFactory contract can be used.

Additionally, the owner has the ability to revoke previously allowed operators, but this action does not disable existing approvals of those operators by users. This creates a scenario where operators that are no longer trusted by the CLOBFactory owner can still act on behalf of users if they were approved before the revocation.

```
function approveOperator(address operator) external {
    if (!allowedOperators[operator]) {
        revert OperatorNotAllowed();
    }

    operatorApprovals[msg.sender][operator] = true;
    emit OperatorApproved(msg.sender, operator);
}

function allowOperator(address operator) external onlyOwner {
    allowedOperators[operator] = true;
    emit OperatorAllowed(operator);
}

function disallowOperator(address operator) external onlyOwner {
    allowedOperators[operator] = false;
    emit OperatorDisallowed(operator);
}
```

Impact

Allowing prohibited operators to remain active could lead to potential security risks.

Recommendations

We recommend implementing the following additional checks in the `onlySenderOrOperator` modifier:

```
modifier onlySenderOrOperator(address account) {
    if (msg.sender != account) {
        ICLOBFactory factory = ICLOBFactory(_getStorage().config.factory);
        if (!factory.operatorApprovals(account, msg.sender) ||
    !factory.allowedOperators(msg.sender)) {
            revert InvalidAccountOrOperator();
        }
    }
    _;
}
```

Remediation

This issue has been acknowledged by Liquid Labs Global, Inc., and a fix was implemented in commit 8b11f544 ↗.

3.3. Inconsistent handling of orders with minimum `amountInBaseLots`

Target	CLOB		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Low	**Impact**	Medium

Description

The `postFillOrder` function accepts the minimum `amountInBaseLots` specified by a constant `MIN_FILL_ORDER_AMOUNT_BASE_LOTS`, which is currently set to 1.

Additionally, this function accepts two types of orders: `IMMEDIATE_OR_CANCEL` and `FILL_OR_KILL`. The `IMMEDIATE_OR_CANCEL` type allows partial filling of the provided order. The `FILL_OR_KILL` type requires the order to be fully filled; otherwise, the transaction is expected to revert.

```
function _processFillBidOrder(
    Book storage ds,
    address account,
    Order memory newOrder,
    PostFillOrderArgs memory args
) private returns (PostFillOrderResult memory) {
    MatchResult memory result = _matchIncomingBid(ds, newOrder);
    [...]
    if (args.fillOrderType == FillOrderType.FILL_OR_KILL &&
    result.totalMatchedBaseLots != result.budgetedBaseLots)
    {
        revert FOKNotFilled();
    }
    [...]
}
```

However, for bid orders, there is a case where this expectation does not align with actual behavior.

The condition for verifying whether an order is fully filled for the `FILL_OR_KILL` order type is implemented in the `_processFillBidOrder` function. Specifically, the transaction reverts if `result.totalMatchedBaseLots` (the total amount matched for the order) does not equal `result.budgetedBaseLots` (the initial order amount in base lots, excluding fees).

However, an issue arises when `amountInBaseLots` is equal to 1. In this case, the `budgetedBaseLots` is rounded down to zero, causing the process of searching for matches for a zero order to be skipped. Consequently, the `_matchIncomingBid` function returns 0 for both `budgetedBaseLots` and `totalMatchedBaseLots` amounts, which satisfies the condition for a successfully completed order.

```
function _matchIncomingBid(Book storage ds, Order memory incomingOrder)
    internal returns (MatchResult memory) {
    uint256 budgetedBaseLots = incomingOrder.amountInBaseLots * 10000 / (
        10000 + ds.config.takerFeeBps);
    MatchResult memory result = MatchResult({
        budgetedBaseLots: budgetedBaseLots,
        totalMatchedBaseLots: 0,
        totalQuoteTokenSentInAtoms: 0,
        totalBaseTokenReceivedInAtoms: 0,
        totalQuoteTokenFeesInAtoms: 0
    });
    while (ds.getBestAsk() <= incomingOrder.priceInTicks && budgetedBaseLots >
    0) {
        [...]
    }
    [...]
    return result;
}
```

Additionally, the `postFillOrder` function may also be susceptible to this issue when the minimum acceptable `amountInBaseLots` is 1, resulting in a transaction that completes successfully but leaves the order neither filled nor added to the order book.

Impact

This issue leads to unnecessary gas consumption for transactions that produce no meaningful result, as the orders are neither filled nor added to the order book.

Recommendations

To mitigate this issue, we recommend increasing the minimum allowed `amountInBaseLots` value.

Remediation

This issue has been acknowledged by Liquid Labs Global, Inc., and a fix was implemented in commit `8b11f544` ↗.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Potential DOS with many small orders

There is the potential for denial of service (DOS) if a lot of small orders that are about to expire are put in and then removed as they expire. The removal of expired orders is done during matching, and if there are many orders to process at this time, it can be gas-consuming and cause DOS. However, it takes about three times as much gas as it would if an attacker tried to place enough orders to cause a DOS. Therefore, the likelihood of an attack is very small and almost impossible.

This issue has been acknowledged by Liquid Labs Global, Inc..

Liquid Labs Global, Inc. provided the following response:

> DOS prevention is based on order placement, enforced by max limits per tx and min limit amount in base lots. the back end of dos occurring (cancelling / expiry being included in fills) is covered by this just as much as really small "legit" fills are

4.2. Tokens not sorted for market creation

The `createMarket` function does not verify that tokens are sorted, which allows the creation of duplicate markets for the same token pair in reversed order. Currently, the `createMarket` function verifies that

- The tokens are not identical.
- Neither token is set to zero.
- A market does not already exist for the provided `quoteToken` and `baseToken` pair with the specified configuration.

However, it is still possible to swap the two tokens and create a duplicate market.

```
function createMarket(
    address quoteToken,
    address baseToken,
    ConfigParams memory config,
    SettingsParams memory settings
) external returns (address) {
    _assertValidTokenPair(quoteToken, baseToken);
```

```
    uint256 quoteDecimals = IERC20Metadata(quoteToken).decimals();
    uint256 baseDecimals = IERC20Metadata(baseToken).decimals();

    _assertValidParams(config, settings, quoteDecimals, baseDecimals);

    bytes32 tokenPairHash = keccak256(abi.encode(quoteToken, baseToken));
    bytes32 configHash = _hashConfig(config);

    if (markets[tokenPairHash][configHash] != address(0)) {
        revert MarketAlreadyExists();
    }
    [...]
    }
```

We recommend implementing token-order verification during market creation to prevent duplicate markets for the same token pair and configuration.

This issue has been acknowledged by Liquid Labs Global, Inc..

Liquid Labs Global, Inc. provided the following response:

> market hash is based on token order and creation is owner gated. its fine to reverse base and quote as it wont cause market hash collisions, but inverse spot markets wont be done as theyre not very useful

4.3. Gas optimization

In the `_executeCancel` function of the CLOB contract, the length of an `args.orderIds` array can be cached to save gas in the loop. Also, the index incrementing for the loop can be made unchecked to optimize gas usage.

This issue has been acknowledged by Liquid Labs Global, Inc., and a fix was implemented in commit `8b11f544` ↗.

4.4. Inconsistent event implementation for `orderId` field

The events in the contract are not implemented consistently. Most events emit the `orderId` field as a `uint256`, but the `OrderMatched` event emits the entire `Order` object, which includes the `id` as a custom `OrderId` type instead of a `uint256`.

```
event OrderMatched(Order takerOrder, Order makerOrder,
    uint256 tradedBaseLots);
```

Consider updating the `OrderMatched` event to emit the `orderId` field as a `uint256` for consistency with other events.

This issue has been acknowledged by Liquid Labs Global, Inc., and a fix was implemented in commit 8b11f544 ↗.

4.5. Ability to take a free flash loan from `deposit`

The `deposit` function in the CLOB contract allows users to deposit funds into the contract's internal balance. The function is implemented in such a way that either the `addBaseToken` or `addQuoteToken` functions is called to update the user's balance before the actual token transfer occurs.

```
function deposit(address account, address token, uint256 amount)
    external onlySenderOrOperator(account) {
    Book storage ds = _getStorage();

    ds.isBaseTokenSafe(token)
        ? ds.accounts.addBaseToken(account, amount)
        : ds.accounts.addQuoteToken(account, amount);

    emit Deposit(account, token, amount);
    IERC20(token).safeTransferFrom(account, address(this), amount);
}
```

At the same time, `postLimitOrder` and `postFillOrder` enable users to specify the settlement type as `ACCOUNT`, which utilizes the internal balance for settlements. Additionally, the `withdraw` function allows users to retrieve funds from their internal balance.

It is important to highlight that the `deposit` function is not strictly ERC-20 compliant. For example, ERC-777 tokens, while similar in interface to ERC-20, invoke an external hook on the sender contract. During a call to `safeTransferFrom` of the token contract, a caller could exploit the tokensToSend hook ↗ to execute code between the update of the internal balance and the transfer of tokens from the caller's account.

This behavior could effectively allow a user to take a free flash loan from their deposited balance. While this does not pose a direct security risk, we want to document this potential behavior for consideration.

Additionally, we would recommend switching the order of the two lines so that the call to `addBaseToken` / `addQuoteToken` is after the transfer. This ensures users must transfer the funds before updating the internal balance.

```
function deposit(address account, address token, uint256 amount)
    external onlySenderOrOperator(account) {
    [...]
    IERC20(token).safeTransferFrom(account, address(this), amount);

    ds.isBaseTokenSafe(token)
        ? ds.accounts.addBaseToken(account, amount)
        : ds.accounts.addQuoteToken(account, amount);
}
```

5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1. Module: CLOB.sol

Function: `cancel(address account, CancelArgs args)`

This function allows users to cancel their orders. The caller of this function can be either the owner of the order or an assigned operator for the order owner.

Inputs

- `account`
  - **Control**: Full control.
  - **Constraints**: `order.owner` should be equal to the `account` address.
  - **Impact**: The owner of the order to be canceled.
- `args`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Contains the array `orderIds` and `settlement`.

Branches and code coverage

Intended branches

- The orders were removed properly.
  - ☑ Test coverage
- The `totalQuoteTokenRefundedInAtoms` matches the expected value.
  - ☑ Test coverage
- The `totalBaseTokenRefundedInAtoms` matches the expected value.
  - ☑ Test coverage

Negative behavior

- The account is not an owner of the provided orders.
  - ☑ Negative test
- The `orderIds` contains duplicate IDs.
  - ☐ Negative test

Function call analysis

- `this._executeCancel(ds, account, args)`
  - **What is controllable?** `account` and `args`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the `totalQuoteTokenRefundedInAtoms` and `totalBaseTokenRefundedInAtoms`. Because every order is deleted before the next one will be handled, there should be no issues related to double counting of funds from the same order.
  - **What happens if it reverts, reenters or does other unusual control flow?** This function reverts if `order.owner` is not an `account`.
- `this._refundBid(account, totalQuoteTokenRefundedInAtoms, args.settlement) -> SafeERC20.safeTransfer(ds.config.quoteToken, account, quoteTokenAmountInAtoms)`
  - **What is controllable?** `account`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** If it reenters, there is no impact because the contract state has been updated.
- `this._refundAsk(account, totalBaseTokenRefundedInAtoms, args.settlement) -> SafeERC20.safeTransfer(ds.config.baseToken, account, baseTokenAmountInAtoms)`
  - **What is controllable?** `account`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** If it reenters, there is no impact because the contract state has been updated.

Function: `collectFees(address recipient)`

This allows the factory contract to collect fees. Only the owner of factory contract can execute the `collectFees` function in the CLOBFactory contract. The full fee amount will be withdrawn.

Inputs

- `recipient`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The receiver of the fee.

Branches and code coverage

Intended branches

- The recipient received the expected amount of fee.
  - ☐ Test coverage

Negative behavior

- The caller is not a factory contract.
  - ☑ Negative test
- Recipient address is zero.
  - ☐ Negative test

Function call analysis

- `SafeERC20.safeTransfer(ds.config.quoteToken, recipient, feesCollected)`
  - **What is controllable?** `recipient`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** The function will revert if `recipient` is zero address or if the `quoteToken` balance of this contract is not sufficient.

Function: `deposit(address account, address token, uint256 amount)`

The function allows to deposit base or quote tokens on behalf of the account. This function can be called by the account itself or an operator authorized for this account.

Inputs

- `account`
  - **Control**: Full control.
  - **Constraints**: The `msg.sender` should be equal to the `account` address or to the operator of the `account`.
  - **Impact**: The internal balance of this account will be updated; tokens will be transferred on behalf of the `account`.
- `token`
  - **Control**: Full control.
  - **Constraints**: The address of the `token` is equal to the base or quote token address.
  - **Impact**: The token to be deposited.
- `amount`
  - **Control**: Full control.
  - **Constraints**: The balance of the account.
  - **Impact**: The amount of tokens to be deposited.

Branches and code coverage

Intended branches

- The base balance of `account` is updated.
  - ☑ Test coverage
- The quote balance of `account` is updated.
  - ☑ Test coverage

Negative behavior

- The caller is not an `account` or `operator` of this account.
  - ☑ Negative test
- Not enough tokens to transfer.
  - ☐ Negative test
- `_allowances` is less than `amount`.
  - ☑ Negative test
- `token` is not base or quote.
  - ☑ Negative test

Function call analysis

- `BookLib.isBaseTokenSafe(ds, token)`
  - **What is controllable?** `token`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns true if the `token` address is equal to the `config.baseToken` address; if the token is equal to the `config.quoteToken`, this function returns false.
  - **What happens if it reverts, reenters or does other unusual control flow?** This function reverts if the provided `token` is not base or quote.
- `AccountLib.addBaseToken(ds.accounts, account, amount)`
  - **What is controllable?** `account` and `amount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Cannot revert or reenter.
- `AccountLib.addQuoteToken(ds.accounts, account, amount)`
  - **What is controllable?** `account` and `amount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Cannot revert or reenter.
- `SafeERC20.safeTransferFrom(IERC20(token), account, address(this), amount)`
  - **What is controllable?** `account` and `amount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.

- **What happens if it reverts, reenters or does other unusual control flow?** Can revert if `_allowances` is not enough or the `account` token balance is not sufficient to transfer `amount` tokens. Can reenter, in this case because the internal balance is updated before the external call, the balance of the account.

Function: `postFillOrder(address account, PostFillOrderArgs args)`

If the order is a buy order, the `fillOrderType` is `FILL_OR_KILL`, and the `totalMatchedBaseLots` is not equal to the `budgetedBaseLots` (this is the `amountInBaseLots` without the fee), this means that the order was not fully filled. This function will revert. If the order is a sell order, the order also should be fully filled, but because in this case the fee is not charged from the base token, the resulting `amountInBaseLots` should be zero.

If the `fillOrderType` is `IMMEDIATE_OR_CANCEL`, the order may be partially filled.

Inputs

- `account`
  - **Control**: Full control.
  - **Constraints**: The account should be equal to the `msg.sender` or assigned operator.
  - **Impact**: The address on whose behalf the order will be filled.
- `args`
  - **Control**: Full control.
  - **Constraints**: The `priceInTicks` should be more than `MIN_LIMIT_PRICE_IN_TICKS`. The `amountInBaseLots` should be more than `MIN_FILL_ORDER_AMOUNT_BASE_LOTS`.
  - **Impact**: Contains `amountInBaseLots`, `priceInTicks`, `side`, `fillOrderType`, and `settlement`.

Branches and code coverage

Intended branches

- The order is `FILL_OR_KILL` and was fully filled.
  - ☑ Test coverage
- The order is `IMMEDIATE_OR_CANCEL` and was partially filled.
  - ☐ Test coverage

Negative behavior

- The order is `FILL_OR_KILL` and was partially filled.
  - ☐ Negative test
- The caller is not an account or operator.
  - ☑ Negative test

- Invalid `priceInTicks`.
  - ☐ Negative test
- Invalid `amountInBaseLots`.
  - ☐ Negative test

Function call analysis

- `BookLib.assertLimitPriceInBounds(args.priceInTicks)`
  - **What is controllable?** `args.priceInTicks`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if `priceInTicks` is less than `MIN_LIMIT_PRICE_IN_TICKS`.
- `BookLib.assertFillOrderAmountInBounds(args.amountInBaseLots)`
  - **What is controllable?** `args.amountInBaseLots`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if `amountInBaseLots` is less than `MIN_FILL_ORDER_AMOUNT_BASE_LOTS`, which is equal to 1.
- `BookLib.incrementNextOrderId(ds)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns `metadata.nextOrderId` and increments it for the next order ID.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._processFillBidOrder(ds, account, newOrder, args) -> this._matchIncomingBid(ds, newOrder)`
  - **What is controllable?** `newOrder`.
  - **If the return value is controllable, how is it used and how can it go wrong?** If the returned `totalMatchedBaseLots` is equal to the returned `budgetedBaseLots` and the `fillOrderType` is `FILL_OR_KILL`, the transaction will be reverted because the order was not fully filled.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems.
- `this._processFillBidOrder(ds, account, newOrder, args) -> this._settleIncomingBid(ds, account, args.settlement, result.totalQuoteTokenSentInAtoms, result.totalBaseTokenReceivedInAtoms)`
  - **What is controllable?** `account` and `settlement`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reentrancy is possible, because the external `safeTransferFrom` and `safeTransfer` functions are called, but there is not any impact because the con-

tract state has been already updated. The function can revert if the quote account balance is insufficient to pay the `totalQuoteTokenSentInAtoms`.

- `this._processFillBidOrder(ds, account, newOrder, args) -> BookLib.accrueTakerFee(ds, result.totalQuoteTokenFeesInAtoms)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** No issues — the function just updates the `totalQuoteTokenFees` and `unclaimedQuoteTokenFees` by the `totalQuoteTokenFeesInAtoms` amount.
- `this._processFillAskOrder(ds, account, newOrder, args) -> this._matchIncomingAsk(ds, newOrder)`
  - **What is controllable?** `newOrder`.
  - **If the return value is controllable, how is it used and how can it go wrong?** The `quoteReceivedAtoms` determines the amount of quote tokens that will be received; the `baseSentAtoms` determines the amount of base tokens that will be sent.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems.
- `this._processFillAskOrder(ds, account, newOrder, args) -> this._settleIncomingAsk(ds, account, args.settlement, quoteReceivedAtoms, baseSentAtoms)`
  - **What is controllable?** `account` and `settlement`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reentrancy is possible, because the external `safeTransferFrom` and `safeTransfer` functions are called, but there is not any impact because the contract state has been already updated. The function can revert if the base account balance is insufficient to pay the `baseSentAtoms`.
- `this._processFillAskOrder(ds, account, newOrder, args) -> BookLib.accrueTakerFee(ds, takerFeeQuoteAtoms)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** No issues — the function just updates the `totalQuoteTokenFees` and `unclaimedQuoteTokenFees` by the `takerFeeQuoteAtoms` amount.

Function: `postLimitOrder(address account, PostLimitOrderArgs args)`

This function allows to post a limit buy and sell order for the account. This function can be called by the account itself or an operator authorized for this account. Also, this function is designed to handle matching and settling new orders with existing orders in the order book. It manages token transfers,

updates the order book, and applies fees.

Inputs

- `account`
  - **Control**: Full control.
  - **Constraints**: The account should be equal to the `msg.sender` or assigned operator.
  - **Impact**: The address on whose behalf the order will be posted.
- `args`
  - **Control**: Full control.
  - **Constraints**: The `cancelTimestamp` should be more than the `block.timestamp`. The `priceInTicks` should be more than `MIN_LIMIT_PRICE_IN_TICKS`. The `amountInBaseLots` should be more than `settings.minLimitOrderAmountInBaseLots`.
  - **Impact**: Contains the information about the order — `amountInBaseLots`, `priceInTicks`, `cancelTimestamp`, `side`, `limitOrderType`, and `settlement`.

Branches and code coverage

Intended branches

- The `BUY` order was successfully posted.
  - ☐ Test coverage
- The `BUY` order was fully executed and was not posted.
  - ☐ Test coverage
- The `BUY` order was partly executed and was posted.
  - ☐ Test coverage
- The `SELL` order was successfully posted.
  - ☐ Test coverage
- The `SELL` order was fully executed and was not posted.
  - ☐ Test coverage
- The `SELL` order was partly executed and was posted.
  - ☐ Test coverage

Negative behavior

- The caller is not an `account` or `operator` of the account.
  - ☑ Negative test
- `cancelTimestamp` < `block.timestamp`.
  - ☐ Negative test
- `priceInTicks` < `MIN_LIMIT_PRICE_IN_TICKS`.
  - ☐ Negative test
- `amountInBaseLots` < `minLimitOrderAmountInBaseLots`.

- ☐ Negative test
- If BUY order — the quote internal account balance is insufficient.
  - ☐ Negative test
- If BUY order — the quote account balance is insufficient.
  - ☐ Negative test
- If SELL order — the base internal account balance is insufficient.
  - ☐ Negative test
- If SELL order — the base account balance is insufficient.
  - ☐ Negative test

Function call analysis

- `this._validateLimitOrderParameters(ds, args) -> BookLib.assertLimitPriceInBounds(args.priceInTicks)`
  - **What is controllable?** `args.priceInTicks`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if `priceInTicks` is less than `MIN_LIMIT_PRICE_IN_TICKS`.
- `this._validateLimitOrderParameters(ds, args) -> BookLib.assertLimitOrderAmountInBounds(ds, args.amountInBaseLots)`
  - **What is controllable?** `args.amountInBaseLots`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if `orderAmountInBaseLots` less than `settings.minLimitOrderAmountInBaseLots`.
- `BookLib.incrementLimitsPlaced(ds, msg.sender)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** If `account` has an exemption from the max limit, the `TRANSIENT_LIMITS_PLACED` will not be incremented. Otherwise, the current `limitsPlaced` will be compared with the `settings.maxLimitsPerTx`, and if the amount exceeds the limit, the transaction will be reverted.
- `BookLib.incrementNextOrderId(ds)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns `metadata.nextOrderId` and increments it for the next order ID.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `OrderLib.isExpired(newOrder)`

- **What is controllable?** `newOrder`.
- **If the return value is controllable, how is it used and how can it go wrong?** Returns `true` if `cancelTimestamp` is less than `block.timestamp` and is not zero; otherwise, it returns false.
- **What happens if it reverts, reenters or does other unusual control flow?** This function cannot revert, but the transaction will be reverted if the return value is true. The transaction will be reverted because the order is expired.

- `this._processLimitBuyOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType)`
  - **What is controllable?** `newOrder` and `args.limitOrderType`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function returns the `postAmountInAtoms`, which determines the remainder in atoms of the order after execution, which will be posted. It could be zero if the full order was executed, or it could be the full order amount if the new order was not executed (or part otherwise). Also, the result of the matching operation will be returned: the `totalQuoteTokenSentInAtoms`, the `totalBaseTokenReceivedInAtoms`, and the `totalQuoteTokenFeesInAtoms`. The `quoteTokenAmountSentInAtoms` + `postAmountInAtoms` should not exceed the initial order amount in atoms for the `BUY` order.
  - **What happens if it reverts, reenters or does other unusual control flow?** This function can revert if the `limitOrderType` is `POST_ONLY` and at the same time, `getBestAsk() <= newOrder.priceInTicks`. The function can also revert if it is not possible to post a new order, if the current `numBids` is equal to the `config.maxNumOrders`.
- `this._processLimitBuyOrder(ds, account, newOrder, args) -> this._settleIncomingBid(ds, account, args.settlement, quoteTokenAmountSentInAtoms + postAmountInAtoms, baseTokenAmountReceivedInAtoms)`
  - **What is controllable?** `account` and `settlement`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reentrancy is possible, because the external `safeTransferFrom` and `safeTransfer` functions are called, but there is not any impact because the contract state has been already updated. The function can revert if the quote account balance is insufficient to pay the `quoteTokenAmountInAtoms`.
- `this._processLimitBuyOrder(ds, account, newOrder, args) -> BookLib.accrueTakerFee(ds, totalQuoteTokenFeeInAtoms)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** No issues — the function just updates the `totalQuoteTokenFees` and `unclaimedQuoteTokenFees` by the `totalQuoteTokenFeeInAtoms` amount.
- `this._processLimitSellOrder(ds, account, newOrder, args) -> this._executeAskLimitOrder(ds, newOrder, args.limitOrderType) -> Book-`

Lib.getBestBid(ds)

- **What is controllable?** newOrder and args.limitOrderType.
- **If the return value is controllable, how is it used and how can it go wrong?** This function returns the postAmountInAtoms, which determines the remainder in atoms of the order after execution, which will be posted. It could be zero if the full order was executed, or it could be the full order amount if the new order was not executed (or part otherwise). Also, the result of the matching operation will be returned: the quoteTokenAmountReceivedInAtoms, the baseTokenAmountSentInAtoms, and the totalQuoteTokenFeeInAtoms. The baseTokenAmountSentInAtoms + postAmountInAtoms should not exceed the initial order amount in atoms for the SELL order.
- **What happens if it reverts, reenters or does other unusual control flow?** This function can revert if limitOrderType is POST_ONLY and at the same time getBestBid() >= newOrder.priceInTicks. The function can also revert if it is not possible to post a new order, if the current numAsks is equal to the config.maxNumOrders.

- this._processLimitSellOrder(ds, account, newOrder, args) -> this._settleIncomingAsk(ds, account, args.settlement, quoteTokenAmountReceivedInAtoms, baseTokenAmountSentInAtoms + postAmountInAtoms)
  - **What is controllable?** account and settlement.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reentrancy is possible, because of safeTransferFrom and safeTransfer, but there is not any impact because the contract state has been already updated. The function can revert if the base account balance is insufficient to pay the baseTokenAmountInAtoms.
- this._processLimitSellOrder(ds, account, newOrder, args) -> BookLib.accrueTakerFee(ds, totalQuoteTokenFeeInAtoms)
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** No issues — the function just updates the totalQuoteTokenFees and unclaimedQuoteTokenFees by the totalQuoteTokenFeeInAtoms amount.

Function: reduce(address account, ReduceArgs args)

This function allows users to adjust their orders. The caller of this function can be either the owner of the order or an assigned operator for the order owner.

Inputs

- account

    - **Control**: Full control.
    - **Constraints**: `order.owner` should be equal to the `account` address.
    - **Impact**: The owner of the order to be reduced.
- `args`
    - **Control**: Full control.
    - **Constraints**: `args.amountInBaseLots` should not be zero.
    - **Impact**: Contains `orderId` and the amount to be reduced.

Branches and code coverage

Intended branches

- The order was removed from the order book properly.
    - ☑ Test coverage
- The order was reduced properly by the provided `amountInBaseLots` value.
    - ☑ Test coverage
- The `quoteTokenOpenInterestInAtoms` was updared properly in the case the order was removed.
    - ☑ Negative test
- The `quoteTokenOpenInterestInAtoms` was updared properly in the case the order was not removed.
    - ☑ Negative test
- The `baseTokenOpenInterestInAtoms` was updared properly in the case the order was removed.
    - ☑ Negative test
- The `baseTokenOpenInterestInAtoms` was updared properly in the case the order was not removed.
    - ☑ Negative test

Negative behavior

- `orderId` is not found
    - ☑ Negative test
- `account` is not an order owner.
    - ☑ Negative test
- `msg.sender` is not an `account` or an assigned operator.
    - ☑ Negative test
- `amountInBaseLots` is zero.
    - ☐ Negative test

Function call analysis

- `this._reduceOrder(ds, order, args)`
    - **What is controllable?** `order` and `args`.

- **If the return value is controllable, how is it used and how can it go wrong?** The returned `side` value determines the bid, or the ask order will be refunded. The `reduceAmount` determines the amount to be refunded in the base lots, and this amount will be converted to atoms. If `isCancel` is true, the `quoteTokenOpenInterestInAtoms` / `baseTokenOpenInterestInAtoms` will not be decreased because they were already updated during the `removeOrderFromBook` function call.
- **What happens if it reverts, reenters or does other unusual control flow?** reverts if order doesn't exist, or if `amountInBaseLots` is zero. There is no external calls here so reentrancy is not possible.

- `BookLib.getQuoteTokenAmountInAtoms(ds, priceInTicks, reduceAmount)`
  - **What is controllable?** `priceInTicks`.
  - **If the return value is controllable, how is it used and how can it go wrong?** The returned value is calculated using the provided `priceInTicks`, the `reduceAmount` is calculated in the `_reduceOrder` function and parameters from the market configuration.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems.
- `this._refundBid(account, refundAtoms, args.settlement)`
  - **What is controllable?** `account` and `args.settlement`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There are no return values.
  - **What happens if it reverts, reenters or does other unusual control flow?** Can reenter before the `quoteTokenOpenInterestInAtoms` update.
- `BookLib.getBaseTokenAmountInAtoms(ds, reduceAmount)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** The returned value is calculated using `reduceAmount` from the `_reduceOrder` function and `baseLotSizeInAtoms` from the market configuration.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems.
- `this._refundAsk(account, refundAtoms, args.settlement)`
  - **What is controllable?** `account` and `args.settlement`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There are no return values.
  - **What happens if it reverts, reenters or does other unusual control flow?** Can reenter before the `baseTokenOpenInterestInAtoms` update.

Function: `withdraw(address account, address token, uint256 amount)`

This function allows to withdraw tokens on behalf of the provided account. This function can be called by the account itself or an operator authorized for this account.

Inputs

- `account`
  - **Control**: Full control.
  - **Constraints**: The `msg.sender` should be equal to the `account` address or to the operator of the `account`.
  - **Impact**: The internal balance of this account will be updated; tokens will be transferred on behalf of the `account`.
- `token`
  - **Control**: Full control.
  - **Constraints**: The address of the `token` is equal to the base or quote token address.
  - **Impact**: The token to be withdrawn.
- `amount`
  - **Control**: Full control.
  - **Constraints**: The internal balance of the `account` should be sufficient.
  - **Impact**: The receiver of the withdrawn tokens.

Branches and code coverage

Intended branches

- The base balance of `account` was updated.
  - ☑ Test coverage
- The quote balance of `account` was updated.
  - ☑ Test coverage

Negative behavior

- Caller is not an `account` or `operator` of this account.
  - ☑ Negative test
- `token` is not base or quote.
  - ☑ Negative test
- The `amount` is more than the balance of the account.
  - ☑ Negative test

Function call analysis

- `BookLib.isBaseTokenSafe(ds, token)`
  - **What is controllable?** `token`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns true if the `token` address is equal to the `config.baseToken` address; if the token is equal to the `config.quoteToken`, this function returns false.

- **What happens if it reverts, reenters or does other unusual control flow?** This function reverts if the provided `token` is not base or quote.

- `AccountLib.removeBaseToken(ds.accounts, account, amount)`
  - **What is controllable?** `account` and `amount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the amount is more than the balance. Cannot reenter.
- `AccountLib.removeQuoteToken(ds.accounts, account, amount)`
  - **What is controllable?** `account` and `amount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the amount is more than the balance. Cannot reenter.
- `SafeERC20.safeTransfer(IERC20(token), account, amount)`
  - **What is controllable?** `account` and `amount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** If it reenters, there is no impact because the contract state has been updated.

Function: `_executeCancel(Book ds, address account, CancelArgs args)`

This internal function accepts the array of order IDs that will be canceled and calculates the total amount of tokens that will be refunded.

Inputs

- `account`
  - **Control**: N/A.
  - **Constraints**: Should be the owner of the orders that will be canceled.
  - **Impact**: N/A.
- `args`
  - **Control**: N/A.
  - **Constraints**: If the order does not exist, the function execution will continue.
  - **Impact**: Contains the array of order IDs.

Branches and code coverage

Intended branches

- The `totalQuoteTokenRefundedInAtoms` is equal to the expected total amount.

☑ Test coverage

- The `totalBaseTokenRefundedInAtoms` is equal to the expected total amount.
  ☑ Test coverage

Negative behavior

- The account is not an owner of the order.
  ☐ Negative test
- The order has been already canceled.
  ☑ Negative test

Function: `_matchIncomingAsk(Book ds, Order incomingOrder)`

This internal function finds matching existing orders for the incoming sell order. The expired matched orders are removed from the order book.

Inputs

- `incomingOrder`
  - **Control**: N/A.
  - **Constraints**: N/A.
  - **Impact**: Contains the side info: `BUY` or `SELL`, this order ID, the owner address, the price, and `amountInBaseLots`.

Branches and code coverage

Intended branches

- There are not any matched orders; the resulting `totalQuoteTokenReceivedInAtoms`, `totalBaseTokenSentInAtoms`, and `totalTakerFeeInQuoteAtoms` are zero; and `incomingOrder.amountInBaseLots` is not changed.
  ☐ Test coverage
- The resulting `totalQuoteTokenReceivedInAtoms`, `totalBaseTokenSentInAtoms`, `totalTakerFeeInQuoteAtoms`, and `incomingOrder.amountInBaseLots` are equal to the expected values.
  ☑ Test coverage

Function: `_matchIncomingBid(Book ds, Order incomingOrder)`

This internal function finds matching existing orders for the incoming buy order. The expired matched orders are removed from the order book.

Inputs

- `incomingOrder`
  - **Control**: N/A.
  - **Constraints**: N/A.
  - **Impact**: Contains the side info: `BUY` or `SELL`, this order ID, the owner address, the price, and `amountInBaseLots`.

Branches and code coverage

Intended branches

- There are not any matched orders; the resulting `totalBaseTokenReceivedInAtoms`, `totalQuoteTokenSentInAtoms`, and `totalQuoteTokenFeesInAtoms` are zero; and `incomingOrder.amountInBaseLots` is not changed.
  - ☐ Test coverage
- The resulting `totalBaseTokenReceivedInAtoms`, `totalQuoteTokenSentInAtoms`, `totalQuoteTokenFeesInAtoms`, and `incomingOrder.amountInBaseLots` are equal to the expected values.
  - ☑ Test coverage

Function: `_matchIncomingOrder(Book ds, Order matchedOrder, Order incomingOrder, uint256 budgetedBaseLots)`

This internal function is designed to handle the token accounting for the order maker. If the taker drains the full matched order, this order will be removed.

Inputs

- `matchedOrder`
  - **Control**: N/A.
  - **Constraints**: N/A.
  - **Impact**: The matched order from the order book.
- `incomingOrder`
  - **Control**: N/A.
  - **Constraints**: N/A.
  - **Impact**: The incoming order will be executed by matching it with an existing order.
- `budgetedBaseLots`
  - **Control**: N/A.
  - **Constraints**: N/A.
  - **Impact**: The remains of the order's funds.

Branches and code coverage

Intended branches

- If the `incomingOrder` is a `BUY` order, the quote token balance of the maker account is updated properly.
  - ☑ Test coverage
- If the `incomingOrder` is a `BUY` order, the `baseTokenOpenInterestInAtoms` is updated properly.
  - ☐ Test coverage
- If the `incomingOrder` is a `SELL` order, the base token balance of the maker account is updated properly.
  - ☑ Test coverage
- If the `incomingOrder` is a `SELL` order, the `quoteTokenOpenInterestInAtoms` is updated properly.
  - ☐ Test coverage
- The calculated fee is equal to the expected amount.
  - ☐ Test coverage

Function: `_reduceOrder(Book ds, Order order, ReduceArgs args)`

This is an internal function that is called from the external `reduce` function. If the order has already expired, the full order amount is reduced, or if the remaining amount is less than the minimum, the order will be completely removed. This function emits the `OrderReduced` event.

Inputs

- `order`
  - **Control**: N/A.
  - **Constraints**: `order.id` should not be zero.
  - **Impact**: Contains information about the order.
- `args`
  - **Control**: N/A.
  - **Constraints**: `args.amountInBaseLots` should not be zero.
  - **Impact**: Contains `orderId` and the amount to be reduced.

Branches and code coverage

Intended branches

- If `block.timestamp > cancelTimestamp`, the order will be removed.
  - ☐ Test coverage
- If `args.amountInBaseLots >= order.amountInBaseLots`, the order will be removed.
  - ☐ Test coverage

- If the remaining amount is less than `minLimitOrderAmountInBaseLots`, the order will be removed.
  - ☑ Test coverage
- The `order.amountInBaseLots` was reduced by the `args.amountInBaseLots`.
  - ☑ Test coverage

Negative behavior

- `order.id` is zero.
  - ☐ Negative test
- `args.amountInBaseLots == 0`.
  - ☐ Negative test

Function: `_refundAsk(address account, uint256 baseTokenAmountInAtoms, Settlement settlement)`

This internal function serves the purpose of refunding base tokens to the provided account. This internal function is called from the `cancel` and `reduce` functions.

Inputs

- `account`
  - **Control**: N/A.
  - **Constraints**: N/A.
  - **Impact**: The user to whom the funds will be refunded.
- `baseTokenAmountInAtoms`
  - **Control**: N/A.
  - **Constraints**: N/A.
  - **Impact**: The amount of base tokens are refunded to the account or added to the account balance.
- `settlement`
  - **Control**: N/A.
  - **Constraints**: `ACCOUNT` or `INSTANT`.
  - **Impact**: In the case of `ACCOUNT`, the internal balances will be updated. In the case of `INSTANT`, tokens will be transferred.

Branches and code coverage

Intended branches

- In the case that the `settlement` is equal to the `ACCOUNT`, the internal account balance of base tokens is updated.
  - ☐ Test coverage

- In the case that the `settlement` is equal to the `INSTANT`, the base token is transferred to the account.
  - ☐ Test coverage

Function: `_refundBid(address account, uint256 quoteTokenAmountInAtoms, Settlement settlement)`

This internal function serves the purpose of refunding quote tokens to the provided account. This internal function is called from the `cancel` and `reduce` functions.

Inputs

- `account`
  - **Control**: N/A.
  - **Constraints**: N/A.
  - **Impact**: The user to whom the funds will be refunded.
- `quoteTokenAmountInAtoms`
  - **Control**: N/A.
  - **Constraints**: N/A.
  - **Impact**: The amount of quote tokens are refunded to the account or added to the account balance.
- `settlement`
  - **Control**: N/A.
  - **Constraints**: `ACCOUNT` or `INSTANT`.
  - **Impact**: In the case of `ACCOUNT`, the internal balances will be updated. In the case of `INSTANT`, tokens will be transferred.

Branches and code coverage

Intended branches

- In the case that `settlement` is equal to `ACCOUNT`, the internal account balance of base tokens is updated.
  - ☐ Test coverage
- In the case that `settlement` is equal to `INSTANT`, the base token is transferred to the account.
  - ☐ Test coverage

Function: `_removeExpiredAsk(Book ds, Order order)`

This internal function is used to remove expired orders and refund the funds from these orders to the account balance.

Inputs

- `order`
  - **Control**: N/A.
  - **Constraints**: N/A.
  - **Impact**: The order to be canceled.

Branches and code coverage

Intended branches

- The base balance of the provided account is updated properly.
  - ☐ Negative test
- The provided order is removed from the order book.
  - ☐ Negative test

Function: `_removeExpiredBid(Book ds, Order order)`

This internal function is used to remove expired orders and refund the funds from these orders to the account balance.

Inputs

- `order`
  - **Control**: N/A.
  - **Constraints**: N/A.
  - **Impact**: The order to be canceled.

Branches and code coverage

Intended branches

- The quote balance of the provided account is updated properly.
  - ☐ Test coverage
- The provided order is removed from the order book.
  - ☐ Test coverage

Function: `_settleIncomingAsk(Book ds, address account, Settlement settlement, uint256 quoteTokenAmountInAtoms, uint256 baseTokenAmountInAtoms)`

This internal function serves the purpose of handling the settlement of an ask order. This function is called from the internal `_processFillAskOrder` and `_processLimitSellOrder`.

Inputs

- `account`
  - **Control**: N/A.
  - **Constraints**: N/A.
  - **Impact**: The account with which the money is settled.
- `settlement`
  - **Control**: N/A.
  - **Constraints**: `ACCOUNT` or `INSTANT`.
  - **Impact**: In the case of `ACCOUNT`, the internal balances will be updated. In the case of `INSTANT`, tokens will be transferred.
- `quoteTokenAmountInAtoms`
  - **Control**: N/A.
  - **Constraints**: N/A.
  - **Impact**: The amount of tokens are sent to the account or added to the account balance.
- `baseTokenAmountInAtoms`
  - **Control**: N/A.
  - **Constraints**: Account should own enough tokens.
  - **Impact**: The amount of tokens are received from the account or removed from the account balance.

Branches and code coverage

Intended branches

- In the case that `settlement` is equal to `ACCOUNT`, the internal account balance is updated.
  - ☐ Test coverage
- In the case that `settlement` is equal to `INSTANT`, the funds are transferred.
  - ☐ Test coverage

Function: `_settleIncomingBid(Book ds, address account, Settlement settlement, uint256 quoteTokenAmountInAtoms, uint256 baseTokenAmountInAtoms)`

This internal function serves the purpose of handling the settlement of an bid order. This function is called from the internal `_processLimitBuyOrder` and `_processFillBidOrder`.

Inputs

- `account`
  - **Control**: N/A.
  - **Constraints**: N/A.
  - **Impact**: The account with which the money is settled.
- `settlement`
  - **Control**: N/A.
  - **Constraints**: `ACCOUNT` or `INSTANT`.
  - **Impact**: In the case of `ACCOUNT`, the internal balances will be updated. In the case of `INSTANT`, tokens will be transferred.
- `quoteTokenAmountInAtoms`
  - **Control**: N/A.
  - **Constraints**: Account should own enough tokens.
  - **Impact**: The amount of tokens are received from the account or removed from the account balance.
- `baseTokenAmountInAtoms`
  - **Control**: N/A.
  - **Constraints**: N/A.
  - **Impact**: The amount of tokens are sent to the account or added to the account balance.

Branches and code coverage

Intended branches

- In the case that `settlement` is equal to `ACCOUNT`, the internal account balance is updated.
  - ☐ Test coverage
- In the case that `settlement` is equal to `INSTANT`, the funds are transferred.
  - ☐ Test coverage

6. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the MegaETH mainnet.

During our assessment on the scoped GTE CLOB contracts, we discovered three findings. No critical issues were found. One finding was of high impact and two were of medium impact.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.