Zellic

Prepared for
Matteo Lunghi
Liquid Labs, Inc.

Prepared by
Katerina Belotskaia
Dimitri Kamenski
Zellic

April 16, 2025

GTE
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, Inc. from March 27th to April 8th, 2025. During this engagement, Zellic reviewed GTE'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:

- Are there any order-matching bugs?
- Are there any inconsistencies in the cost of trades?
- Are there any inconsistencies in the amendment logic?
- Can the simple bonding curve yield inconsistent token returns?
- Does Launchpad have potential gaps that could lead to nongraduation of tokens?
- Are there refund or approval oversights in the router logic?

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

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

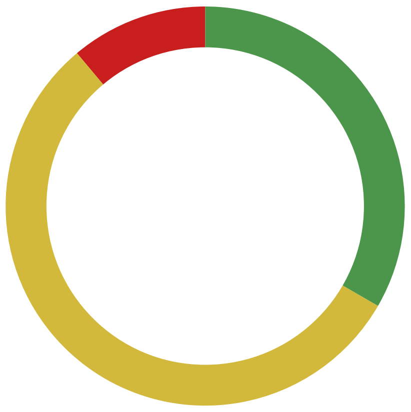
1.4. Results

During our assessment on the scoped GTE contracts, we discovered nine findings. One critical issue was found. Five were of medium impact and three were of low impact.

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

Breakdown of Finding Impacts

Impact Level	Count
🟥 Critical	1
🟧 High	0
🟨 Medium	5
🟩 Low	3
⬜ Informational	0



2. Introduction

2.1. About GTE

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

> GTE is a protocol that offers various on-chain trading products, from bonding curve launches, to an amm, to an on-chain spot clob (thanks to megaeth's high gas limit), finally to perps.

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 Contracts

Type	Solidity
Platform	EVM-compatible

Target	gte-contracts
Repository	https://github.com/liquid-labs-inc/gte-contracts ↗
Version	`addf91f02833a263745f1bff0f48f60c3da1a4ae`

Programs	`clob/*`
	`launchpad/SimpleLaunchpad`
	`launchpad/BondingCurves/SimpleBondingCurve`
	`router/GTERouter.sol`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 2.6 person-weeks. The assessment was conducted by two consultants over the course of 1.8 calendar weeks.

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 ↗

Dimitri Kamenski
Engineer
dimitri@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

March 27, 2025	Kick-off call
March 27, 2025	Start of primary review period
April 8, 2025	End of primary review period

3. Detailed Findings

3.1. Full base-token balance can be drained from CLOBManager

Target	CLOB, CLOBManager		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	High	**Impact**	Critical

Description

The `amend` function allows either the owner of an order or an approved operator to update an existing order. In the case where the `side` and `price` of the order remain the same, the internal `_executeAmendAmount` function will be called. This function uses the provided `args.amountInBase` as an `amount` and, for sell orders, calculates the delta amount based on the change in base-token amount.

The resulting `baseTokenDelta` is of `int256` type and can be either positive or negative. If the result is positive, the surplus of base tokens is returned to the user; otherwise, the user must provide the missing amount of base tokens. The `amount` and `order.amount` values are initially of type `uint256` and are cast to `int256` to calculate `baseTokenDelta`.

The issue is that in Solidity, when performing a type cast like `int256(amount)`, no overflow or bounds checks are performed. If the `uint256` value exceeds the maximum value of `int256`, the result will be a negative value.

If a user provides a new `amount` that exceeds the maximum value of `int256` (which is 57896044618658097711785492504343953926634992332820282019728792003956564819967), casting it to `int256` will cause an overflow, resulting in a negative number. As a result, the expression `int256(order.amount) - int256(amount)` will produce a positive value. This means that the current `order.amount` can be significantly less than the new `amount` value, but due to the overflow, the resulting difference will still be interpreted as a surplus of base tokens, which will be transferred to the user.

```
function _executeAmendAmount(Book storage ds, Order storage order,
    uint256 amount)
    internal
    returns (int256 quoteTokenDelta, int256 baseTokenDelta)
{
    if (order.side == Side.BUY) {
        [...]
    } else {
        baseTokenDelta = int256(order.amount) - int256(amount);
        ds.metadata.baseTokenOpenInterest
    = uint256(int256(ds.metadata.baseTokenOpenInterest) - baseTokenDelta);
    }
    order.amount = amount;
}
```

Impact

If a user provides an `args.amountInBase` value exceeding the maximum `int256`, the cast to `int256` overflows and results in a large negative number. For example,

```
args.amountInBase =
105792089237316195423570985008687907853269984665640564039407584007913129639935
order.amount = 10000000000000000000
int256(args.amountInBase) =
-10000000000000000000000000000000000000000000000000000000050000000000000000000001
```

This leads to the following calculation:

```
baseTokenDelta = int256(order.amount) - int256(args.amountInBase)
= 10000000000000000000 - (
-10000000000000000000000000000000000000000000000000000000050000000000000000000001
)
=
1000000000000000000000000000000000000000000000000000000006000000000000000000001
```

The `_executeAmendAmount` function returns this `baseTokenDelta` value, which is passed to `_settleAmend`. This function is responsible for invoking the appropriate method on the clobManager to either distribute or collect tokens — either directly or through the internal account balance — depending on the settlement type and the sign of the delta.

If the `settlement` type is set to `ACCOUNT`, the internal user balance (`accountTokenBalances`) within the clobManager is credited with this `baseTokenDelta` via a call to `clobManager.creditAccount`. Furthermore, using the `clobManager.withdraw` function, a user can withdraw all base tokens from the clobManager contract if their internal `accountTokenBalance` is greater than or equal to the contract's current balance.

As a result, a malicious user can exploit this overflow to receive an excessive amount of base tokens.

```
function _settleAmend(
    Book storage ds,
    address maker,
    Settlement settlement,
    int256 quoteTokenDelta,
    int256 baseTokenDelta
) internal {
    ICLOBManager clobManager = ICLOBManager(ds.config.factory);

    if (settlement == Settlement.INSTANT) {
        [...]
    } else {
        [...]
        if (baseTokenDelta > 0) {
            clobManager.creditAccount(maker, address(ds.config.baseToken),
    uint256(baseTokenDelta));
        } else if (baseTokenDelta < 0) {
            clobManager.debitAccount(maker, address(ds.config.baseToken),
    uint256(-baseTokenDelta));
        }
    }
}

function creditAccount(CLOBManagerStorage storage self, address account,
    address token, uint256 amount) internal {
    self.accountTokenBalances[account][token] += amount;
}
```

Recommendations

To prevent overflow when casting from `uint256` to `int256`, it is strongly recommended to use the SafeCast library provided by OpenZeppelin. The function `SafeCast.toInt256(uint256 value)` will safely revert the transaction if the input exceeds the bounds of `int256`.

Remediation

This issue has been acknowledged by Liquid Labs, Inc., and fixes were implemented in the following commits:

- 0661898b ↗
- 0680c394 ↗

Liquid Labs, Inc. provided the following response to this finding:

> Added `SafeCast` on all `uint256` to `uint128` conversions.

3.2. Invalid `baseReserve` stalls buy on bonding curve

Target	SimpleBondingCurve		
Category	Coding Mistakes	**Severity**	High
Likelihood	Low	**Impact**	Medium

Description

The SimpleBondingCurve contract provides exchange rates between quote token and base token prior to graduation events. Graduation is when the `BONDING_SUPPLY` has been reached and remaining balances in the Launchpad are sent to a `UniswapV2` token pool.

The `_getQuoteAmount()` function (used in both `buy()` and `sell()`) is as follows:

```
function _getQuoteAmount(uint256 baseAmount, uint256 quoteReserve,
    uint256 baseReserve, bool isBuy)
    internal
    pure
    returns (uint256 quoteAmount)
{
    uint256 baseReserveAfter = isBuy ? baseReserve - baseAmount : baseReserve
    + baseAmount;

    return (quoteReserve * baseAmount) / baseReserveAfter;
}
```

If the initial `r.baseReserve == BONDING_SUPPLY`, as the `baseReserve` reaches the `BONDING_SUPPLY`, the denominator will tend towards zero, ultimately reverting with division by zero.

This sets the lower bound for SimpleLaunchpad configuration of `VIRTUAL_BASE` to `1`.

Impact

If an administrator does not understand that the `VIRTUAL_BASE` lower bound is 1, they may configure the bonding curve such that the Launchpad can launch tokens but will never allow graduation breaking key assumptions of early investors.

Recommendations

Add validation to the `setVirtualReserves()` that prevents 0 values.

Remediation

This issue has been acknowledged by Liquid Labs, Inc., and a fix was implemented in commit `7422a467` ↗.

Liquid Labs, Inc. provided the following response to this finding:

> Virtual base can no longer be set to 0 by the owner of SimpleBondingCurve.sol

3.3. The `VIRTUAL_BASE` upper-bound stalls buy on bonding curve

Target	SimpleBondingCurve		
Category	Coding Mistakes	**Severity**	High
Likelihood	Low	**Impact**	Medium

Description

In addition to Finding 3.2. ↗, further prevention of graduation is possible through subsequent administrative misunderstanding of the `VIRTUAL_BASE` upper-bound values.

Focusing on the final operation in `_getQuoteAmount()` function, we have

```
return (quoteReserve * baseAmount) / baseReserveAfter;
```

It is clear the value returned can round to zero if the `quoteReserve * baseAmount < baseReserveAfter`.

Assuming only one base token remains before graduation would imply `quoteReserve < baseReserveAfter` produces a 0 quote-token input into the bonding curve. If 0 input is registered for 1 token output, the `SimpleLaunchpad::buy()` will revert with `DustAttackInvalid()`.

Impact

If an administrator does not understand that the `VIRTUAL_BASE` upper bound can cause rounding, they may configure the bonding curve such that the Launchpad can launch tokens but will never allow graduation breaking key assumptions of early investors.

Recommendations

Ensure that the final `quoteReserve` achievable after buying `BONDING_SUPPLY` is larger than the `baseReserveAfter`.

Remediation

This issue has been acknowledged by Liquid Labs, Inc., and a fix was implemented in commit `b2e9f242` ↗.

Liquid Labs, Inc. provided the following response to this finding:

Small buys that would cost 0 quote, causing a DustAttackInvalid revert are now allowed if its the final buy that causes graduation, preventing the DOS.

3.4. Front-running orders is possible through order amendments

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

Description

The CLOB contract organizes orders by a defined tick spacing that is validated on all limit and fill orders. However, amendments do not validate tick spacing for price amendments. This leads to orders that fall outside the traditional tick-spaced–orders bounds.

If many orders exist in a single price-tick range (say 6.15e18), a malicious user can game the ordering by amending their order's price to 6.150000001e18. This has negligible effect on the price but will push their order in front of all existing orders.

Impact

Ordering of orders in the CLOB is not guaranteed, leading to unfair distribution of sellers to makers.

Recommendations

Implement the `BookLib.assertLimitPriceInBounds(ds, args.priceLimit);` for all price amendments to the order book.

Remediation

This issue has been acknowledged by Liquid Labs, Inc., and a fix was implemented in commit `5e82bd34` ↗.

3.5. Global `bondingCurve` misuse in SimpleLaunchPad functions

Target	SimpleLaunchPad		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

The SimpleLaunchPad contract defines a global `bondingCurve` variable, which can be updated by the contract owner via the `updateBondingCurve` function. Additionally, the `bondingCurve` variable is used in the `launch` function, and its current value is saved into the `LaunchData` structure for every newly launched token.

This `bondingCurve` address is subsequently used in the `buy` and `sell` functions to calculate the `quoteAmount` of tokens to be supplied or received. Additionally, it is utilized in the view functions `quoteBaseForQuote` and `quoteQuoteForBase` to compute the expected `baseAmount` based on a provided `quoteAmount`, and vice versa.

However, all of these functions reference the global `bondingCurve` address, rather than using the instance stored in `LaunchData` for the corresponding launched token, which may result in referencing an incorrect bonding curve contract if the global address has been updated, potentially leading to inaccurate token-amount calculations.

Impact

The impact depends on the `bondingCurve` implementation being updated to. In our case, all previously launched tokens become inaccessible, since the new `bondingCurve` has zero reserves for them, resulting in all calculations returning zero.

Recommendations

Update all relevant functions (`buy`, `sell`, `quoteBaseForQuote`, and `quoteQuoteForBase`) to reference the `bondingCurve` address stored in the `LaunchData` for each specific launched token, rather than using the global `bondingCurve` variable.

Remediation

This issue has been acknowledged by Liquid Labs, Inc., and a fix was implemented in commit `316301c5` ↗.

Liquid Labs, Inc. provided the following response to this finding:

Global bonding curve address is no longer referenced anywhere, except when creating a launch. Updating the global bonding curve will now only affect new launches.

3.6. WETH unwrap fails due to incorrect token flow

Target	GTERouter		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

The `executeRoute` function allows executing multiple actions, `postFillOrder` and `swapExactTokensForTokens`, using the provided user funds and transferring the resulting tokens to the caller.

At the end of the `executeRoute` execution, if the resulting token is WETH and the `isWrapping` flag is set to true, the internal `_handleUnwrap` function is triggered. This function attempts to withdraw the final swap result from the WETH contract and transfer the corresponding amount of native tokens to the caller.

However, both `_executeClobPostFillOrder` and `_executeUniV2SwapExactTokensForTokens` functions already handle transferring the resulting tokens to the caller. The `_executeClobPostFillOrder` function withdraws the result of the last `postFillOrder` execution on behalf of the caller from the clobFactory, if it is the final swap in the sequence and the user has specified the `settlement` type as `INSTANT`. The `_executeUniV2SwapExactTokensForTokens` function uses `msg.sender` as the `recipient` if it is the final swap in the sequence.

As a result, during `_handleUnwrap`, the contract will attempt to transfer the resulting tokens to the caller a second time, which may lead to failing operations, since GTERouter balances are already depleted.

```
function executeRoute(
    [...]
    ) external payable override nonReentrant {
        [...]
        for (uint256 i = 0; i < hops.length; i++) {
            [...]
            if (currSelector == this.executeClobPostFillOrder.selector) {
                (route.prevAmountOut, route.nextTokenIn)
    = _executeClobPostFillOrder(route, hops[i]);
            } else if (currSelector ==
    this.executeUniV2SwapExactTokensForTokens.selector) {
                (route.prevAmountOut, route.nextTokenIn)
    = _executeUniV2SwapExactTokensForTokens(route, hops[i]);
            } else {
                revert UnsupportedSelector();
            }

            route.prevSelector = currSelector;
        }

        [...]
        if (route.nextTokenIn == address(weth) && isWrapping)
    _handleUnwrap(route.prevAmountOut);
    }
```

Impact

Due to the redundant transfer logic in the `executeRoute` function, the router attempts to withdraw and transfer tokens to the caller a second time via `_handleUnwrap`, even though the resulting tokens have already been transferred during the execution of `_executeClobPostFillOrder` or `_executeUniV2SwapExactTokensForTokens`.

The design assumes that the router does not hold any additional token balance at the end of the action. Therefore, when `_handleUnwrap` tries to withdraw tokens, it is likely to revert, as there are no remaining funds to transfer. And this behavior will cause the entire transaction to fail.

Recommendations

Consider removing the `_handleUnwrap` logic from the `executeRoute` entirely and delegating the responsibility of unwrapping WETH to the user. Alternatively, consider adding specific logic to `_executeClobPostFillOrder` and `_executeUniV2SwapExactTokensForTokens` to handle cases where unwrapping is required.

In situations where the final token is WETH and the `isWrapping` flag is set to `true`,

- the `_executeUniV2SwapExactTokensForTokens` should specify the recipient as the GTERouter itself instead of the caller, and
- the `_executeClobPostFillOrder` function should withdraw tokens on behalf of the GTERouter contract instead of the caller.

This adjustment ensures that the router retains control of the resulting WETH, allowing `_handleUnwrap` to execute successfully without reverts.

Remediation

This issue has been acknowledged by Liquid Labs, Inc., and a fix was implemented in commit `200ce4a1` ↗.

3.7. Invalid fee tiers cause permanent order-matching denial of service

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

Description

The FeeData.sol contract establishes fee categories for accounts using the CLOB. Those fees are read during all maker/taker fee calculations during order matching. If an administrative user sets the fee category of a user greater than 15, all orders will revert with `IndexOutOfBounds()`.

Impact

This scenario requires an administrative user to call `setAccountFeeTier()` with an invalid fee tier. If that same account then submits any order to the order book, the action will revert, due to the invalid fee configuration.

Recommendations

We recommend adding validation to the `setAccountFeeTier()` such that the account tier cannot exceed 15.

Remediation

This issue has been acknowledged by Liquid Labs, Inc., and a fix was implemented in commit `ebfd69a7` ↗.

Liquid Labs, Inc. provided the following response to this finding:

> Fee tier length is now asserted during `setAccountFeeTier` so that an upgrade cannot result in accidentally adding and using more than 16 tiers.

3.8. Inconsistent `amountIn` handling in the first swap of `executeRoute`

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

Description

The GTERouter contract includes the `_executeUniV2SwapExactTokensForTokens` function, which is called in a loop from the `executeRoute` function to perform a sequence of swaps.

The caller supplies an initial `amountIn` and a set of `hops` data describing the individual swap steps. The `amountIn` is transferred to the GTERouter at the beginning of `executeRoute`, but for the first swap, `_executeUniV2SwapExactTokensForTokens` uses the `amountIn` defined within the `hop` parameters rather than the `amountIn` passed directly to `executeRoute`.

However, there is no check to ensure that the `amountIn` specified in the first `hop` actually matches the user's provided `amountIn`. This opens the possibility for a mismatch between the actual funds deposited and the parameters used in the first swap.

Impact

If the `amountIn` in the first swap is manipulated or incorrect, the GTERouter may attempt to perform a swap using an unintended or mismatched amount. This can lead to unexpected swap behavior and cause funds to be incorrectly routed or swapped.

Recommendations

Consider using the `route.prevAmountOut` for all swaps, including the first.

Remediation

This issue has been acknowledged by Liquid Labs, Inc., and a fix was implemented in commit `45ab6c74` ↗.

Liquid Labs, Inc. provided the following response to this finding:

> For Uniswap v2 swaps, the `amountIn` is now always the previous hop's amount out. The previous amount out defaults to executeRoute's amount in for the first hop.

3.9. Noncompetitive orders can be replaced with smaller orders

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

Description

The `_removeNonCompetitiveOrder()` function allows the removal of orders when the order book is at its limit. The removal is only supposed to be triggered if the incoming order is worth more. However, it only validates that the `newOrder.price > minBidPrice` or `newOrder.price < maxAskPrice` respectfully, depending on whether the order is a bid limit or ask limit.

However, competitive potential of an order should be calculated by `volume * price`. Otherwise, this will allow a malicious user to cancel all orders in the furthest tick boundaries from the current price, replacing them with slightly better prices but much smaller volumes.

This could lead to the book having concentrated liquidity only within a few price ticks from the current price, leading to unnecessary volatility in pricing.

Impact

Valid, superior orders are canceled, leading to volatile pricing.

Recommendations

Validate that replaced orders are worth more both by volume and price.

Remediation

This issue has been acknowledged by Liquid Labs, Inc.

Liquid Labs, Inc. provided the following response to this finding:

> Removing non competitive orders is a failsafe so that a book with over 2 billion orders doesn't stall. We use `minLimitOrderAmount` to ensure this does not occur. Making the removal of the farthest orders conditional on new order having more value adds too much additional logic.

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. Test suite

When building a complex contract ecosystem with multiple moving parts and dependencies, comprehensive testing is essential. This includes testing for both positive and negative scenarios. Positive tests should verify that each function's side effect is as expected, while negative tests should cover every revert, preferably in every logical branch.

The test coverage for this project should be expanded to include all contracts, not just surface-level functions. It is important to test the invariants required for ensuring security. Therefore, we recommend building a rigorous test suite that includes all contracts to ensure the system operates securely and as intended. This should also include additional test cases covering edge scenarios. For example, cases involving type conversions should be tested to ensure that potential overflows are properly handled.

Good test coverage has multiple effects.

- It finds bugs and design flaws early (preaudit or prerelease).
- It displays code maturity.
- It bolsters customer trust in your product.
- It improves understanding of how the code functions, integrates, and operates — for developers and auditors alike.
- It increases development velocity long-term.

The last point seems contradictory, given the time investment to create and maintain tests. To expand upon that, tests help developers trust their own changes. It is difficult to know if a code refactor — or even just a small one-line fix — breaks something if there are no tests. This is especially true for new developers or those returning to the code after a prolonged absence. Tests have your back here. They are an indicator that the existing functionality *most likely* was not broken by your change to the code.

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: CLOBManager.sol

Function: `approveOperator(address operator)`

This function allows any caller to set an operator. The provided operator address must be one that is allowed by the owner of this contract. The user's operator is permitted to call the `deposit` and `withdraw` functions from this contract. Additionally, the `postFillOrder`, `postLimitOrder`, `amend`, and `cancel` market functions are available for execution by the user's operator.

Inputs

- `operator`
  - **Control**: Full control.
  - **Constraints**: `allowedOperators[operator]` is true.
  - **Impact**: The operator can act on behalf of the caller's account.

Branches and code coverage

Intended branches

- The provided operator has been successfully set up.
  - ☑ Test coverage

Negative behavior

- The provided operator is not allowed by the owner.
  - ☐ Negative test

Function call analysis

- `CLOBManagerStorageLib.approveOperator(ds, operator)`
  - **What is controllable?** `operator`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.

- **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the provided operator is not allowed by the owner.

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

This function allows a trusted market contract created by the owner of the contract using the `createMarket` function to add tokens to the internal account balance.

Inputs

- `account`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The account will be credited.
- `token`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The token address to credit.
- `amount`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The amount of tokens to credit.

Branches and code coverage

Intended branches

- The internal balance of the account has been increased by the `amount`.
  - ☐ Test coverage

Negative behavior

- The caller is not a trusted market.
  - ☑ Negative test

Function call analysis

- `CLOBManagerStorageLib.creditAccount(this._getStorage(), account, token, amount)`

- **What is controllable?** `token`, `account`, and `amount`.
- **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
- **What happens if it reverts, reenters or does other unusual control flow?** Increases the internal account balance using the specified amount.

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

This function allows a trusted market contract created by the owner of the contract using the `createMarket` function to decrease the internal account balance.

Inputs

- `account`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The account address whose internal balance will be decreased.
- `token`
  - **Control**: Full control.
  - **Constraints**: The internal balance of the account should be sufficient.
  - **Impact**: The token address whose balance will be decreased for the account.
- `amount`
  - **Control**: Full control.
  - **Constraints**: The internal balance of the account should be sufficient.
  - **Impact**: The amount of tokens that will be transferred to this contract.

Branches and code coverage

Intended branches

- The balance of the account has been decreased by the `amount`.
  - ☐ Test coverage

Negative behavior

- The caller is not a trusted market.
  - ☐ Negative test
- The internal user's balance is not enough.
  - ☐ Negative test

Function call analysis

- `CLOBManagerStorageLib.debitAccount(this._getStorage(), account, token, amount)`
    - **What is controllable?** `token`, `account`, and `amount`
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Decreases the internal account balance using the specified `amount` and reverts if the balance is insufficient.

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

This function allows an account itself or approved operator to provide the deposit of the arbitrary token. Funds can be provided from the operator or from the account, depending on the `fromOperator`. But only the `account` will be credited.

Inputs

- `account`
    - **Control**: Full control.
    - **Constraints**: If `account` is not equal to `msg.sender`, `msg.sender` should be `isApprovedOperator`.
    - **Impact**: The deposit is credited to this account.
- `token`
    - **Control**: Full control.
    - **Constraints**: No constraints.
    - **Impact**: The specified token address will be transferred.
- `amount`
    - **Control**: Full control.
    - **Constraints**: The `funder` should have enough tokens to transfer.
    - **Impact**: The specified token amount will be transferred.
- `fromOperator`
    - **Control**: Full control.
    - **Constraints**: No constraints.
    - **Impact**: If true, tokens will be transferred from the caller.

Branches and code coverage

Intended branches

- The provided account address has been credited by the specified amount.
  - ☐ Test coverage
- The `msg.sender` has provided tokens to the contract in the case `fromOperator` is true.
  - ☐ Test coverage
- The `account` has provided tokens to the contract in the case `fromOperator` is false.
  - ☐ Test coverage

Negative behavior

- The caller is not an account and is not an approved operator.
  - ☑ Negative test

Function call analysis

- `SafeTransferLib.safeTransferFrom(token, funder, address(this), amount)`
  - **What is controllable?** `token` and `amount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the funder does not have enough balance or has not approved a sufficient allowance.
- `CLOBManagerStorageLib.creditAccount(ds, account, token, amount)`
  - **What is controllable?** `account`, `token`, and `amount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** This function increases the internal account balance with the specified `amount`.

Function: `disapproveOperator(address operator)`

This function allows any caller to disapprove an operator.

Inputs

- `operator`

- **Control**: Full control.
- **Constraints**: No constraints.
- **Impact**: The operator cannot act on behalf of the caller's account after `disapproveOperator` execution.

Branches and code coverage

Intended branches

- The provided operator has been successfully disapproved.
  - ☑ Test coverage

Function call analysis

- `CLOBManagerStorageLib.disapproveOperator(ds, operator)`
  - **What is controllable?** `operator`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Even if an operator has not been approved before, they can still be disapproved.

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

This function allows a trusted market contract created by the owner of the contract using the `createMarket` function to transfer tokens from the account.

Inputs

- `account`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The account address from which tokens will be transferred to this contract.
- `token`
  - **Control**: Full control.
  - **Constraints**: The balance of the account and allowance should be sufficient.
  - **Impact**: The token address that will be transferred to this contract.

- `amount`
    - **Control**: Full control.
    - **Constraints**: The balance of the account and allowance should be sufficient.
    - **Impact**: The amount of tokens that will be transferred to this contract.

Branches and code coverage

Intended branches

- Tokens have been successfully transferred.
    - ☐ Test coverage

Negative behavior

- The caller is not a trusted market.
    - ☐ Negative test
- The allowance is not enough.
    - ☐ Negative test
- The account balance is not enough.
    - ☐ Negative test

Function call analysis

- `SafeTransferLib.safeTransferFrom(token, account, address(this), amount)`
    - **What is controllable?** `token`, `account`, and `amount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Transfers `amount` of `tokens` from the `account` to this contract.

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

This function allows a trusted market contract created by the owner of the contract using the `createMarket` function to transfer tokens directly to the account.

Inputs

- `account`
    - **Control**: Full control.

    - **Constraints**: N/A.
    - **Impact**: The receiver of tokens.
- `token`
    - **Control**: Full control.
    - **Constraints**: The balance of the contract should be sufficient.
    - **Impact**: The token address to transfer.
- `amount`
    - **Control**: Full control.
    - **Constraints**: The balance of the contract should be sufficient.
    - **Impact**: The amount of tokens to transfer.

Branches and code coverage

Intended branches

- Tokens have been successfully transferred to the account.
    - ☐ Test coverage

Negative behavior

- The caller is not a trusted market.
    - ☑ Negative test

Function call analysis

- `SafeTransferLib.safeTransfer(token, account, amount)`
    - **What is controllable?** `token`, `account`, and `amount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Transfers the provided `amount` of `tokens` to the specified `account`.

Function: `settleIncomingOrder(SettleParams params)`

This function allows a trusted market contract created by the owner of the contract using the `createMarket` function to perform account settlement.

Inputs

- `params`

- **Control**: Full control.
- **Constraints**: N/A.
- **Impact**: Contains the `taker` address, `quoteToken` and `baseToken`, the side `BUY` or `SELL`, `settlement` type, `takerQuoteAmount` and `takerBaseAmount`, and also `makerCredits` info.

Branches and code coverage

Intended branches

- the `side` is `Side.BUY`, the `takerFee` is equal to the expected fee.
  - [ ] Test coverage
- the `side` is `Side.SELL`, the `takerFee` is equal to the expected fee.
  - [ ] Test coverage
- the `settlement` is `INSTANT` and `Side.side` is `BUY`, the expected amount of the `quoteToken` has been transferred to the contract from the `taker`.
  - [ ] Test coverage
- the `settlement` is `INSTANT` and `Side.side` is `BUY`, the expected amount of the `baseToken` has been transferred to the `taker`.
  - [ ] Test coverage
- the `settlement` is `INSTANT` and `Side.side` is `SELL`, the expected amount of the `baseToken` has been transferred to the contract from the `taker`.
  - [ ] Test coverage
- the `settlement` is `INSTANT` and `Side.side` is `SELL`, the expected amount of the `quoteToken` has been transferred to the `taker`.
  - [ ] Test coverage
- the `settlement` is `ACCOUNT` and `Side.side` is `BUY`, the expected amount of the `quoteToken` has been debited from the `taker` balance.
  - [ ] Test coverage
- the `settlement` is `ACCOUNT` and `Side.side` is `BUY`, the expected amount of the `baseToken` has been credited to the `taker` balance.
  - [ ] Test coverage
- the `settlement` is `ACCOUNT` and `Side.side` is `SELL`, the expected amount of the `baseToken` has been debited from the `taker` balance.
  - [ ] Test coverage
- the `settlement` is `ACCOUNT` and `Side.side` is `SELL`, the expected amount of the `quoteToken` has been credited to the `taker` balance.
  - [ ] Test coverage

Negative behavior

- The caller is not a trusted market.
    - ☑ Negative test

Function call analysis

- `FeeDataLib.getTakerFee(ds.feeData, this.takerFees, params.taker, params.takerBaseAmount)`
    - **What is controllable?** `params.taker` and `params.takerBaseAmount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the fee amount charged from the taker based on the `takerBaseAmount`. The fee for the account depends on the `FeeTiers` index, which is set up by the owner. By default, this is equal to `ZERO` index.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `FeeDataLib.accrueFee(ds.feeData, params.baseToken, takerFee)`
    - **What is controllable?** `params.baseToken`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `FeeDataLib.getTakerFee(ds.feeData, this.takerFees, params.taker, params.takerQuoteAmount)`
    - **What is controllable?** `params.taker` and `params.takerQuoteAmount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the fee amount charged from the taker based on the `takerQuoteAmount`. The fee for the account depends on the `FeeTiers` index, which is set up by the owner. By default, this is equal to `ZERO` index.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `FeeDataLib.accrueFee(ds.feeData, params.quoteToken, takerFee)`
    - **What is controllable?** `params.baseToken`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `SafeTransferLib.safeTransferFrom(params.quoteToken, params.taker, address(this), params.takerQuoteAmount)`
    - **What is controllable?** `params.quoteToken`, `params.taker`, and

    `params.takerQuoteAmount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Transfers `takerQuoteAmount` from the taker account if the side is `BUY`, representing the payment for filling the order.
- `SafeTransferLib.safeTransfer(params.baseToken, params.taker, params.takerBaseAmount)`
    - **What is controllable?** `params.baseToken`, `params.taker`, and `params.takerBaseAmount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Transfers `takerBaseAmount` to the taker account if the side is `BUY`, representing the proceeds from filling the order.
- `SafeTransferLib.safeTransferFrom(params.baseToken, params.taker, address(this), params.takerBaseAmount)`
    - **What is controllable?** `params.baseToken`, `params.taker`, and `params.takerBaseAmount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Transfers `takerBaseAmount` from the taker account if the side is `SELL`, representing the payment for filling the order.
- `SafeTransferLib.safeTransfer(params.quoteToken, params.taker, params.takerQuoteAmount)`
    - **What is controllable?** `params.quoteToken`, `params.taker`, and `params.takerQuoteAmount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Transfers `takerQuoteAmount` to the taker account if the side is `SELL`, representing the proceeds from filling the order.
- `CLOBManagerStorageLib.debitAccount(ds, params.taker, params.quoteToken, params.takerQuoteAmount)`
    - **What is controllable?** `params.taker`, `params.quoteToken`, and `params.takerQuoteAmount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Decreases the internal account balance using the specified `takerQuoteAmount` amount and reverts if the balance is insufficient.

- `CLOBManagerStorageLib.creditAccount(ds, params.taker, params.baseToken, params.takerBaseAmount)`
  - **What is controllable?** `params.taker`, `params.baseToken`, and `params.takerBaseAmount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Increases the internal account balance using the specified `takerBaseAmount` amount.
- `CLOBManagerStorageLib.debitAccount(ds, params.taker, params.baseToken, params.takerBaseAmount)`
  - **What is controllable?** `params.taker`, `params.baseToken`, and `params.takerBaseAmount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Decreases the internal account balance using the specified `takerBaseAmount` amount and reverts if the balance is insufficient.
- `CLOBManagerStorageLib.creditAccount(ds, params.taker, params.quoteToken, params.takerQuoteAmount)`
  - **What is controllable?** `params.taker`, `params.quoteToken`, and `params.takerQuoteAmount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Increases the internal account balance using the specified `takerQuoteAmount` amount.
- `this._settleMakerFill(params.quoteToken, params.baseToken, params.makerCredits, params.side) -> FeeDataLib.getMakerFee(ds.feeData, this.makerFees, credit.maker, credit.quoteAmount)`
  - **What is controllable?** `credit.maker` and `credit.quoteAmount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the fee amount charged from the maker based on the `quoteAmount`. The fee for the account depends on the `FeeTiers` index, which is set up by the owner. By default, this is equal to `ZERO` index.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._settleMakerFill(params.quoteToken, params.baseToken, params.makerCredits, params.side) -> FeeDataLib.accrueFee(ds.feeData, quoteToken, makerFee)`
  - **What is controllable?** `quoteToken`.

- **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
- **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here. But the fee will be charged only in the case when the `clobSide` is `BUY`, and accordingly the `maker` side is `SELL` and `credit.quoteAmount` is more than zero. If `credit.baseAmount` is more than zero, in this case, it means than the maker order has been expired, the fee should not be charged, and the maker will get the full refund.

- `this._settleMakerFill(params.quoteToken, params.baseToken, params.makerCredits, params.side) -> FeeDataLib.getMakerFee(ds.feeData, this.makerFees, credit.maker, credit.baseAmount)`
    - **What is controllable?** `credit.maker` and `credit.quoteAmount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the fee amount charged from the maker based on the `baseAmount`. The fee for the account depends on the `FeeTiers` index, which is set up by the owner. By default, this is equal to `ZERO` index.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._settleMakerFill(params.quoteToken, params.baseToken, params.makerCredits, params.side) -> FeeDataLib.accrueFee(ds.feeData, baseToken, makerFee)`
    - **What is controllable?** `baseToken`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here. But the fee will be charged only in the case when the `clobSide` is `SELL`, and accordingly the `maker` side is `BUY` and `credit.baseAmount` is more than zero. If `credit.quoteAmount` is more than zero, in this case, it means than the maker order has been expired, the fee should not be charged, and the maker will get the full refund.
- `this._settleMakerFill(params.quoteToken, params.baseToken, params.makerCredits, params.side) -> CLOBManagerStorageLib.creditAccount(ds, credit.maker, quoteToken, credit.quoteAmount)`
    - **What is controllable?** `credit.maker` and `credit.quoteAmount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Increases the internal account balance using the specified `quoteAmount` amount.
- `this._settleMakerFill(params.quoteToken, params.baseToken, params.makerCredits, params.side) -> CLOBManagerStorageLib.creditAccount(ds, credit.maker, baseToken,`

`credit.baseAmount)`

- **What is controllable?** `credit.maker` and `credit.baseAmount`.
- **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
- **What happens if it reverts, reenters or does other unusual control flow?** Increases the internal account balance using the specified `baseAmount` amount.

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

This function allows an account itself or approved operator to withdraw the deposited token. Funds can be transferred to the operator or to the account, depending on the `toOperator`.

Inputs

- `account`
  - **Control**: Full control.
  - **Constraints**: If `account` is not equal to `msg.sender`, `msg.sender` should be `isApprovedOperator`.
  - **Impact**: The amount will be withdrawn from this account.
- `token`
  - **Control**: Full control.
  - **Constraints**: The account should have a nonzero balance of this token to withdraw.
  - **Impact**: The token will be withdrawn.
- `amount`
  - **Control**: Full control.
  - **Constraints**: The account should have enough tokens to withdraw.
  - **Impact**: The amount will be withdrawn.
- `toOperator`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: If the caller is an operator and `toOperator` is true, withdrawn funds will be transferred to the operator.

Branches and code coverage

Intended branches

- Funds have been withdrawn successfully to the account if `toOperator` is false and the caller is an operator.
  - ☐ Test coverage
- Funds have been withdrawn successfully to the operator if `toOperator` is true and the caller is an operator.
  - ☐ Test coverage
- Funds have been withdrawn successfully to the account if `toOperator` is false and the caller is the account itself.
  - ☐ Test coverage
- Funds have been withdrawn successfully to the account if `toOperator` is true and the caller is the account itself.
  - ☐ Test coverage

Negative behavior

- Withdraw the full balance and try to withdraw again.
  - ☐ Negative test
- The caller is not an account and is not an approved operator.
  - ☑ Negative test

Function call analysis

- `CLOBManagerStorageLib.debitAccount(ds, account, token, amount)`
  - **What is controllable?** `account`, `token`, and `amount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Decreases the internal account balance with the specified `amount` and reverts if the balance is less than the provided `amount`.
- `SafeTransferLib.safeTransfer(token, recipient, amount)`
  - **What is controllable?** `token`, `recipient`, and `amount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reentrancy is possible, but the external transfer call is executed after the internal balance of the account is increased, so it is not possible to do a double withdrawal.

5.2. Module: CLOB.sol

Function: `amend(address account, AmendArgs args)`

This function amends an existing limit order for the specified account. It can be called by the account itself or an approved operator, and it handles matching and settling amended orders with the existing order book. It also notifies the CLOBManager for token settlement and applies fees.

Inputs

- `account`
    - **Control**: Full control.
    - **Constraints**: Should be `msg.sender` itself or an approved operator for the `account` and the owner of the order.
    - **Impact**: The owner of the order being amended.
- `args`
    - **Control**: Full control.
    - **Constraints**: `args.limitOrderType` can only be `POST_ONLY`.
    - **Impact**: Contains `orderId`, `amountInBase`, `price`, `cancelTimestamp`, `side`, `limitOrderType`, and `settlement`.

Branches and code coverage

Intended branches

- If `args.cancelTimestamp` is less than the current `block.timestamp`, the order will be canceled successfully without a fee.
    - ☐ Test coverage
- If `args.amountInBase` is less than the `settings.minLimitOrderAmountInBase`, the order will be canceled successfully without a fee.
    - ☐ Test coverage
- The `side` of the order has been changed successfully.
    - ☑ Test coverage
- The `price` of the order has been updated successfully.
    - ☑ Test coverage

Negative behavior

- `order.owner != account`.
    - ☑ Negative test

- The caller is not an `account` or `operator` of the account.
    - ☑ Negative test
- `price % tickSize != 0`.
    - ☐ Negative test
- `args.price == 0`.
    - ☐ Negative test
- `limitOrderType == LimitOrderType.GOOD_TILL_CANCELLED`.
    - ☐ Negative test
- `order.id` does not exist.
    - ☐ Negative test

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

This function allows the account to cancel multiple orders. It can be called by the account itself or an operator authorized for this account.

Inputs

- `account`
    - **Control**: Full control.
    - **Constraints**: Should be `msg.sender` itself or an approved operator for the `account` and the owner of the order.
    - **Impact**: The owner of the order being canceled.
- `args`
    - **Control**: Full control.
    - **Constraints**: All orders from `orderIds` should belong to the `account`.
    - **Impact**: Contains `orderIds` array and `settlement`.

Branches and code coverage

Intended branches

- The provided `orderId` has already been canceled, but it is ignored.
    - ☑ Test coverage
- The `totalQuoteTokenRefunded` equals the expected amount.
    - ☑ Test coverage
- The `totalBaseTokenRefunded` equals the expected amount.

  - [x] Test coverage
- The `orderIds` have been successfully deleted from the book.
  - [ ] Test coverage
- The `settlement` is `ACCOUNT`, and `account` has been credited.
  - [ ] Test coverage
- The `settlement` is `INSTANT`, and tokens have been transferred directly to the `account` address.
  - [ ] Test coverage

Negative behavior

- The caller is not an `account` or `operator` of the account.
  - [x] Negative test
- The `orderIds` contains an order with a different owner than the given account.
  - [x] Negative test

Function call analysis

- `this._executeCancel(ds, account, args) -> BookLib.getQuoteTokenAmount(ds, order.price, order.amount)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the quote-tokens amount calculated using the provided `order.price` and `order.amount` amount. The result can be rounded down to zero if `order.amount * order.price` is less than `config.baseSize`.
  - **What happens if it reverts, reenters or does other unusual control flow?** Can revert as a result of overflow during `order.amount * order.price` calculation if `order.price` or `matchData.order.amount` is too large.
- `this._executeCancel(ds, account, args) -> BookLib.removeOrderFromBook(ds, order)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** The `metadata.quoteTokenOpenInterest` and `metadata.baseTokenOpenInterest` will be decremented — depends on the side of the order, and `order.id` will be deleted from the `orders` list. Also, `bidTree` and `askTree` will be updated in addition to `orders.nextOrderId` and `orders.prevOrderId`.
- `factory.creditAccount(account, quoteToken, totalQuoteTokenRefunded)`

  - **What is controllable?** `account`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** This function increases the internal account balance using the specified `totalQuoteTokenRefunded` amount. But there is no verification that the actual factory balance is sufficient to replenish the account for this amount.
- `factory.pushToAccount(account, quoteToken, totalQuoteTokenRefunded)`
  - **What is controllable?** `account`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Directly transfers the `totalQuoteTokenRefunded` amount of the `quoteToken` to the provided `account` address — reverts if `factory` does not own enough tokens.
- `factory.creditAccount(account, baseToken, totalBaseTokenRefunded)`
  - **What is controllable?** `account`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** This function increases the internal account balance using the specified `totalBaseTokenRefunded` amount. But there is no verification that the actual factory balance is sufficient to replenish the account for this amount.
- `factory.pushToAccount(account, baseToken, totalBaseTokenRefunded)`
  - **What is controllable?** `account`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Directly transfers the `totalBaseTokenRefunded` amount of the `baseToken` to the provided `account` address — reverts if `factory` does not own enough tokens.

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

This function allows to fill a buy and sell order for the account. It can be called by the account itself or an operator authorized for this account. Also, this function is designed to handle matching with existing orders. It provides the execution result to the CLOBManager contract for token settlement and applies fees. The `postFillOrder` function reverts if the type of the order is `FILL_OR_KILL` and the order was not fully filled during matching. If the `fillOrderType` is `IMMEDIATE_OR_CANCEL`, the order may be partially filled.

Inputs

- `account`
    - **Control**: Full control.
    - **Constraints**: Should be `msg.sender` itself or an approved operator for the `account`.
    - **Impact**: The account on whose behalf the order is being filled.
- `args`
    - **Control**: Full control.
    - **Constraints**: `args.priceLimit % tickSize == 0 && args.priceLimit != 0`.
    - **Impact**: Contains `amount`, `priceLimit`, `side`, `amountIsBase`, `fillOrderType`, and `settlement`.

Branches and code coverage

Intended branches

- The order type is `FILL_OR_KILL`, the order was fully filled, and the `args.amountIsBase` is `true`.
    - ☑ Test coverage
- The order type is `FILL_OR_KILL`, the order was fully filled, and the `args.amountIsBase` is `false`.
    - ☑ Test coverage
- The order type is `IMMEDIATE_OR_CANCEL`, the order was partly filled, and the `args.amountIsBase` is `true`.
    - ☐ Test coverage
- The order type is `IMMEDIATE_OR_CANCEL`, the order was partly filled, and the `args.amountIsBase` is `false`.
    - ☐ Test coverage
- The order type is `IMMEDIATE_OR_CANCEL`, the order was fully filled, and the `args.amountIsBase` is `true`.
    - ☐ Test coverage
- The order type is `IMMEDIATE_OR_CANCEL`, the order was fully filled, and the `args.amountIsBase` is `false`.
    - ☐ Test coverage
- The `takerFee` is equal to the expected amount.
    - ☑ Test coverage
- The multiple expired orders have been closed and fee was not charged from them.

- ☐ Test coverage
- There were multiple matches, and the maker fee is equal to the expected amount.
  - ☐ Test coverage

Negative behavior

- The order type is `FILL_OR_KILL`, the order was not fully filled, and the `args.amountIsBase` is `true`.
  - ☑ Negative test
- The order type is `FILL_OR_KILL`, the order was not fully filled, and the `args.amountIsBase` is `false`.
  - ☑ Negative test
- The caller is not an `account` or `operator` of the account.
  - ☑ Negative test

Function call analysis

- `BookLib.assertLimitPriceInBounds(ds, args.priceLimit)`
  - **What is controllable?** `args.priceLimit`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if `priceLimit` is zero or `price % tickSize != 0`.
- `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 the next order ID.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `OrderLib.toOrder(args, orderId, account)`
  - **What is controllable?** `args.side`, `args.amount`, and `args.price`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the new `Order` object.
  - **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, args.amountIsBase) -> BookLib.getBestAsk(ds)`
  - **What is controllable?** N/A.

    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the best minimum price `askTree.minimum()`.
    - **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, args.amountIsBase) -> OrderLib.isExpired(bestAskOrder)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns true when the `cancelTimestamp` is not null and is less than the current `block.timestamp`; otherwise, it returns false.
    - **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, args.amountIsBase) -> this._removeExpiredAsk(ds, bestAskOrder) -> TransientMakerData.addBaseToken(order.owner, baseTokenAmount)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here. The `baseAmount` will be saved for the owner of the expired order in the `TransientMakerData` memory.
- `this._processFillBidOrder(ds, account, newOrder, args) -> this._matchIncomingBid(ds, newOrder, args.amountIsBase) -> this._removeExpiredAsk(ds, bestAskOrder) -> BookLib.removeOrderFromBook(ds, order)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** The `metadata.quoteTokenOpenInterest` and `metadata.baseTokenOpenInterest` will be decremented — depends on the side of the order, and `order.id` will be deleted from the `orders` list. Also, `bidTree` and `askTree` will be updated in addition to `orders.nextOrderId` and `orders.prevOrderId`.
- `this._processFillBidOrder(ds, account, newOrder, args) -> this._matchIncomingBid(ds, newOrder, args.amountIsBase) -> this._matchIncomingOrder(ds, bestAskOrder, incomingOrder, incomingOrder.amount, amountIsBase) -> FixedPointMathLib.min(matchedBase, incomingOrder.amount)`
    - **What is controllable?** `incomingOrder.amount`.

    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the minimum value between `matchedBase` and `incomingOrder.amount`.
    - **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, args.amountIsBase) -> this._matchIncomingOrder(ds, bestAskOrder, incomingOrder, incomingOrder.amount, amountIsBase) -> BookLib.getQuoteTokenAmount(ds, matchedPrice, matchData.baseDelta)`
    - **What is controllable?** If the provided `incomingOrder.amount` is less than `matchedBase`, the `matchData.baseDelta` will be equal to the `incomingOrder.amount` and fully controlled by the caller.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the quote-tokens amount calculated using the provided `matchedPrice` and `matchData.baseDelta` amount. The result can be rounded down to zero if `matchData.baseDelta * matchedPrice` is less than `config.baseSize`.
    - **What happens if it reverts, reenters or does other unusual control flow?** Can revert as a result of overflow during `matchData.baseDelta * matchedPrice` calculation if `matchedPrice` or `matchData.baseDelta` is too large.
- `this._processFillBidOrder(ds, account, newOrder, args) -> this._matchIncomingBid(ds, newOrder, args.amountIsBase) -> this._matchIncomingOrder(ds, bestAskOrder, incomingOrder, incomingOrder.amount, amountIsBase) -> FixedPointMathLib.min(matchedBase, BookLib.getBaseTokenAmount(ds, matchedPrice, incomingOrder.amount))`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the minimum amount between the `matchedBase` and the result of the `getBaseTokenAmount` function execution, which calculates the base-token amount using the provided-by-the-caller `incomingOrder.amount` and the `matchedPrice`.
    - **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, args.amountIsBase) -> this._matchIncomingOrder(ds, bestAskOrder, incomingOrder, incomingOrder.amount, amountIsBase) -> BookLib.getBaseTokenAmount(ds, matchedPrice, incomingOrder.amount)`
    - **What is controllable?** `incomingOrder.amount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the base-token amount, which is calculated using the

provided-by-the-caller `incomingOrder.amount` and the `matchedPrice`. Can return zero if `quoteAmount * self.config.baseSize` is less than `price`.
  - **What happens if it reverts, reenters or does other unusual control flow?** Can revert as a result of overflow during `quoteAmount * self.config.baseSize` calculation if `quoteAmount` is too large.
- `this._processFillBidOrder(ds, account, newOrder, args) -> this._matchIncomingBid(ds, newOrder, args.amountIsBase) -> this._matchIncomingOrder(ds, bestAskOrder, incomingOrder, incomingOrder.amount, amountIsBase) -> BookLib.getQuoteTokenAmount(ds, matchedPrice, matchData.baseDelta)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the quote-tokens amount calculated using the provided `matchedPrice` and `matchData.baseDelta` amount. The result can be rounded down to zero if `matchData.baseDelta * matchedPrice` is less than `config.baseSize`.
  - **What happens if it reverts, reenters or does other unusual control flow?** Can revert as a result of overflow during `matchData.baseDelta * matchedPrice` calculation if `matchedPrice` or `matchData.baseDelta` is too large.
- `this._processFillBidOrder(ds, account, newOrder, args) -> this._matchIncomingBid(ds, newOrder, args.amountIsBase) -> this._matchIncomingOrder(ds, bestAskOrder, incomingOrder, incomingOrder.amount, amountIsBase) -> TransientMakerData.addQuoteToken(matchedOrder.owner, matchData.quoteDelta)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here. The `matchData.quoteDelta` will be saved for the owner of the matched order in the `TransientMakerData` memory.
- `this._processFillBidOrder(ds, account, newOrder, args) -> this._matchIncomingBid(ds, newOrder, args.amountIsBase) -> this._matchIncomingOrder(ds, bestAskOrder, incomingOrder, incomingOrder.amount, amountIsBase) -> TransientMakerData.addBaseToken(matchedOrder.owner, matchData.baseDelta)`
  - **What is controllable?** `matchData.baseDelta` can be controlled by the caller, if this amount is less than the `matchedOrder.amount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here. The `matchData.baseDelta` will be saved for the

owner of the matched order in the `TransientMakerData` memory.

- `this._processFillBidOrder(ds, account, newOrder, args) -> this._matchIncomingBid(ds, newOrder, args.amountIsBase) -> this._matchIncomingOrder(ds, bestAskOrder, incomingOrder, incomingOrder.amount, amountIsBase) -> BookLib.removeOrderFromBook(ds, matchedOrder)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** The `metadata.quoteTokenOpenInterest` and `metadata.baseTokenOpenInterest` will be decremented — depends on the side of the order, and `order.id` will be deleted from the `orders` list. Also, `bidTree` and `askTree` will be updated in addition to `orders.nextOrderId` and `orders.prevOrderId`.
- `this._processFillBidOrder(ds, account, newOrder, args) -> this._settleIncomingOrder(ds, account, Side.BUY, args.settlement, result.totalQuoteTokenSent, result.totalBaseTokenReceived) -> TransientMakerData.getMakerCreditsAndClearStorage()`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the current state of the maker's credits, including the expired orders and matched orders.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._processFillBidOrder(ds, account, newOrder, args) -> this._settleIncomingOrder(ds, account, Side.BUY, args.settlement, result.totalQuoteTokenSent, result.totalBaseTokenReceived) -> factory.settleIncomingOrder(settleParams)`
    - **What is controllable?** `settleParams.taker`, `settleParams.side`, and `settleParams.settlement` are controlled by the caller.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the `takerFee` amount, which is used for events and as a part of the returned `PostFillOrderResult`.
    - **What happens if it reverts, reenters or does other unusual control flow?** If `settlement` is set to `INSTANT`, the function might revert during `safeTransferFrom` if the taker does not have enough balance or has not approved a sufficient allowance. Also, it reverts if the balance of the factory contract is not enough to transfer tokens to the taker. If `settlement` is set to `ACCOUNT`, the function might revert during the `debitAccount` function call if the taker does not have enough internal token balance.
- `this._processFillAskOrder(ds, account, newOrder, args) -> this._matchIncomingAsk(ds, newOrder, args.amountIsBase) ->`

`BookLib.getBestBid(ds)`

- **What is controllable?** N/A.
- **If the return value is controllable, how is it used and how can it go wrong?** Returns the best maximum price `bidTree.minimum()`.
- **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.

- `this._processFillAskOrder(ds, account, newOrder, args) -> this._matchIncomingAsk(ds, newOrder, args.amountIsBase) -> OrderLib.isExpired(bestBidOrder)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns true when the `cancelTimestamp` is not null and is less than the current `block.timestamp`; otherwise, it returns false.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._processFillAskOrder(ds, account, newOrder, args) -> this._matchIncomingAsk(ds, newOrder, args.amountIsBase) -> this._removeExpiredBid(ds, bestBidOrder) -> BookLib.getQuoteTokenAmount(ds, order.price, order.amount)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the quote-tokens amount calculated using the expired `order.price` and `order.amount`. The result can be rounded down to zero if `order.amount * order.price` is less than `config.baseSize`.
    - **What happens if it reverts, reenters or does other unusual control flow?** Can revert as a result of overflow during `order.amount * order.price` calculation if `order.price` or `order.amount` is too large.
- `this._processFillAskOrder(ds, account, newOrder, args) -> this._matchIncomingAsk(ds, newOrder, args.amountIsBase) -> this._removeExpiredBid(ds, bestBidOrder) -> TransientMakerData.addQuoteToken(order.owner, quoteTokenAmount)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._processFillAskOrder(ds, account, newOrder, args) -> this._matchIncomingAsk(ds, newOrder, args.amountIsBase) -> this._removeExpiredBid(ds, bestBidOrder) -> BookLib.removeOrderFromBook(ds, order)`
    - **What is controllable?** N/A.

- **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
- **What happens if it reverts, reenters or does other unusual control flow?** The `metadata.quoteTokenOpenInterest` and `metadata.baseTokenOpenInterest` will be decremented — depends on the side of the order, and `order.id` will be deleted from the `orders` list. Also, `bidTree` and `askTree` will be updated in addition to `orders.nextOrderId` and `orders.prevOrderId`.

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

This function allows to post a limit buy and sell order for the account. It 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 provides the execution result to the CLOBManager contract for token settlement and applies fees.

Inputs

- `account`
  - **Control**: Full control.
  - **Constraints**: Should be `msg.sender` itself or an approved operator for the `account`.
  - **Impact**: The account on whose behalf the order is being filled.
- `args`
  - **Control**: Full control.
  - **Constraints**: The `cancelTimestamp` should be more than the `block.timestamp`. The `amountInBaseLots` should be more than `settings.minLimitOrderAmountInBase`. Also, `price % tickSize == 0 && args.price != 0`.
  - **Impact**: Contains the information about the order — `amountInBase`, `price`, `cancelTimestamp`, `side`, `limitOrderType`, and `settlement`.

Branches and code coverage

Intended branches

- The `BUY` order was fully executed and was not posted.
  - [ ] Test coverage
- The `BUY` order was partly executed and was posted.
  - [x] 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
- `price % tickSize != 0`.
    - ☑ Negative test
- `args.price == 0`.
    - ☑ Negative test
- If there is a `BUY` order and `settlement` is `ACCOUNT` — the quote internal-account balance in CLOBManager is insufficient.
    - ☑ Negative test
- If there is a `SELL` order and `settlement` is `ACCOUNT` — the base internal-account balance in CLOBManager is insufficient.
    - ☐ Negative test
- If there is a `BUY` order and `settlement` is `INSTANT` — the quote account balance is insufficient.
    - ☑ Negative test
- If there is a `SELL` order and `settlement` is `INSTANT` — the base account balance is insufficient.
    - ☐ Negative test

Function call analysis

- `BookLib.assertLimitPriceInBounds(ds, args.price)`
    - **What is controllable?** `args.priceLimit`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if `priceLimit` is zero or `price % tickSize != 0`.
- `BookLib.assertLimitOrderAmountInBounds(ds, args.amountInBase)`
    - **What is controllable?** `args.amountInBase`.

- **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
- **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the `amountInBase` is less than `minLimitOrderAmountInBase`.

- `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?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the number of orders placed in this transaction exceeds the `maxLimitsPerTx` or if the limit is exceeded for the `caller`.
- `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 the next order ID.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `OrderLib.toOrder(args, orderId, account)`
    - **What is controllable?** `args.side`, `args.amountInBase`, and `args.priceLimit`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the new `Order` object.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `OrderLib.isExpired(newOrder)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns true when the `cancelTimestamp` is not null and is less than the current `block.timestamp`; otherwise, it returns false.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) -> this._matchIncomingBid(ds, newOrder, true)`
    - **What is controllable?** `newOrder`.
    - **If the return value is controllable, how is it used and how can it go wrong?** If there are not any matches, this function returns zero `totalQuoteTokenSent` and zero `totalBaseTokenReceived`.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if only one of `totalQuoteTokenSent` or `totalBaseTokenReceived` is zero.

- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) -> this._matchIncomingBid(ds, newOrder, true) -> BookLib.getBestAsk(ds)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the best minimum price `askTree.minimum()`.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) -> this._matchIncomingBid(ds, newOrder, true) -> OrderLib.isExpired(bestAskOrder)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns true when the `cancelTimestamp` is not null and is less than the current `block.timestamp`; otherwise, it returns false.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) -> this._matchIncomingBid(ds, newOrder, true) -> this._removeExpiredAsk(ds, bestAskOrder) -> TransientMakerData.addBaseToken(order.owner, baseTokenAmount)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here. The `baseAmount` will be saved for the owner of the expired order in the `TransientMakerData` memory.
- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) -> this._matchIncomingBid(ds, newOrder, true) -> this._removeExpiredAsk(ds, bestAskOrder) -> BookLib.removeOrderFromBook(ds, order)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** The `metadata.quoteTokenOpenInterest` and `metadata.baseTokenOpenInterest` will be decremented — depends on the side of the order, and `order.id` will be deleted from the `orders` list. Also, `bidTree` and `askTree` will be updated in addition to `orders.nextOrderId` and `orders.prevOrderId`.

- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) -> this._matchIncomingBid(ds, newOrder, true) -> this._matchIncomingOrder(ds, bestAskOrder, incomingOrder, incomingOrder.amount, amountIsBase) -> FixedPointMathLib.min(matchedBase, incomingOrder.amount)`
    - **What is controllable?** `incomingOrder.amount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the minimum value between `matchedBase` and `incomingOrder.amount`.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) -> this._matchIncomingBid(ds, newOrder, true) -> this._matchIncomingOrder(ds, bestAskOrder, incomingOrder, incomingOrder.amount, amountIsBase) -> BookLib.getQuoteTokenAmount(ds, matchedPrice, matchData.baseDelta)`
    - **What is controllable?** If the provided `incomingOrder.amount` is less than `matchedBase`, the `matchData.baseDelta` will be equal to the `incomingOrder.amount` and fully controlled by the caller.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the quote-tokens amount calculated using the provided `matchedPrice` and `matchData.baseDelta` amount. The result can be rounded down to zero if `matchData.baseDelta * matchedPrice` is less than `config.baseSize`.
    - **What happens if it reverts, reenters or does other unusual control flow?** Can revert as a result of overflow during `matchData.baseDelta * matchedPrice` calculation if `matchedPrice` or `matchData.baseDelta` is too large.
- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) -> this._matchIncomingBid(ds, newOrder, true) -> this._matchIncomingOrder(ds, bestAskOrder, incomingOrder, incomingOrder.amount, amountIsBase) -> FixedPointMathLib.min(matchedBase, BookLib.getBaseTokenAmount(ds, matchedPrice, incomingOrder.amount))`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the minimum amount between the `matchedBase` and the result of the `getBaseTokenAmount` function execution, which calculates the base-token amount using the provided-by-the-caller `incomingOrder.amount` and the `matchedPrice`.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.

- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) -> this._matchIncomingBid(ds, newOrder, true) -> this._matchIncomingOrder(ds, bestAskOrder, incomingOrder, incomingOrder.amount, amountIsBase) -> BookLib.getBaseTokenAmount(ds, matchedPrice, incomingOrder.amount)`
    - **What is controllable?** `incomingOrder.amount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the base-token amount, which is calculated using the provided-by-the-caller `incomingOrder.amount` and the `matchedPrice`. Can return zero if `quoteAmount * self.config.baseSize` is less than `price`.
    - **What happens if it reverts, reenters or does other unusual control flow?** Can revert as a result of overflow during `quoteAmount * self.config.baseSize` calculation if `quoteAmount` is too large.
- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) -> this._matchIncomingBid(ds, newOrder, true) -> this._matchIncomingOrder(ds, bestAskOrder, incomingOrder, incomingOrder.amount, amountIsBase) -> BookLib.getQuoteTokenAmount(ds, matchedPrice, matchData.baseDelta)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the quote-tokens amount calculated using the provided `matchedPrice` and `matchData.baseDelta` amount. The result can be rounded down to zero if `matchData.baseDelta * matchedPrice` is less than `config.baseSize`.
    - **What happens if it reverts, reenters or does other unusual control flow?** Can revert as a result of overflow during `matchData.baseDelta * matchedPrice` calculation if `matchedPrice` or `matchData.baseDelta` is too large.
- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) -> this._matchIncomingBid(ds, newOrder, true) -> this._matchIncomingOrder(ds, bestAskOrder, incomingOrder, incomingOrder.amount, amountIsBase) -> TransientMakerData.addQuoteToken(matchedOrder.owner, matchData.quoteDelta)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here. The `matchData.quoteDelta` will be saved for the owner of the matched order in the `TransientMakerData` memory.

- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) -> this._matchIncomingBid(ds, newOrder, true) -> this._matchIncomingOrder(ds, bestAskOrder, incomingOrder, incomingOrder.amount, amountIsBase) -> TransientMakerData.addBaseToken(matchedOrder.owner, matchData.baseDelta)`
    - **What is controllable?** `matchData.baseDelta` can be controlled by the caller, if this amount is less than the `matchedOrder.amount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here. The `matchData.baseDelta` will be saved for the owner of the matched order in the `TransientMakerData` memory.
- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) -> this._matchIncomingBid(ds, newOrder, true) -> this._matchIncomingOrder(ds, bestAskOrder, incomingOrder, incomingOrder.amount, amountIsBase) -> BookLib.removeOrderFromBook(ds, matchedOrder)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** The `metadata.quoteTokenOpenInterest` and `metadata.baseTokenOpenInterest` will be decremented — depends on the side of the order, and `order.id` will be deleted from the `orders` list. Also, `bidTree` and `askTree` will be updated in addition to `orders.nextOrderId` and `orders.prevOrderId`.
- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) -> this._removeNonCompetitiveOrder(ds, ds.orders[ds.bidLimits[minBidPrice].tailOrder]) -> factory.creditAccount(order.owner, address(ds.config.quoteToken), quoteRefunded)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** This function increases the internal account balance using the specified `quoteRefunded` amount. But there is no verification that the actual factory balance is sufficient to replenish the account for this amount.
- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) ->`

`this._removeNonCompetitiveOrder(ds, ds.orders[ds.bidLimits[minBidPrice].tailOrder]) -> ds.removeOrderFromBook(order)`

- **What is controllable?** N/A.
- **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
- **What happens if it reverts, reenters or does other unusual control flow?** The `metadata.quoteTokenOpenInterest` and `metadata.baseTokenOpenInterest` will be decremented — depends on the side of the order, and `order.id` will be deleted from the `orders` list. Also, `bidTree` and `askTree` will be updated in addition to `orders.nextOrderId` and `orders.prevOrderId`.

- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) -> ds.addOrderToBook(newOrder);`
    - **What is controllable?** `newOrder` is partly controlled by the caller.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Updates the book with a new order. Increases the `quoteTokenOpenInterest` with the order amount since the side is `BUY`.
- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) -> ds.getQuoteTokenAmount(newOrder.price, newOrder.amount);`
    - **What is controllable?** `newOrder.price` and `newOrder.amount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the quote-tokens amount calculated using the provided `matchedPrice` and `matchData.baseDelta` amount. The result can be rounded down to zero if `matchData.baseDelta * matchedPrice` is less than `config.baseSize`.
    - **What happens if it reverts, reenters or does other unusual control flow?** Can revert as a result of overflow during `matchData.baseDelta * matchedPrice` calculation if `matchedPrice` or `matchData.baseDelta` is too large.
- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._settleIncomingOrder(ds, account, Side.BUY, args.settlement, quoteTokenAmountSent + postAmount,baseTokenAmountReceived) -> TransientMakerData.getMakerCreditsAndClearStorage()`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the current state of the maker's credits including the expired orders and matched orders.

    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._settleIncomingOrder(ds, account, Side.BUY, args.settlement, quoteTokenAmountSent + postAmount,baseTokenAmountReceived) -> factory.settleIncomingOrder(settleParams)`
    - **What is controllable?** `settleParams.settlement` is controlled by the caller.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the `takerFee` amount, which is used for events and as a part of the returned `PostFillOrderResult`.
    - **What happens if it reverts, reenters or does other unusual control flow?** If `settlement` is set to `INSTANT`, the function might revert during `safeTransferFrom` if the taker does not have enough balance or has not approved a sufficient allowance. Also, it reverts if the balance of the factory contract is not enough to transfer tokens to the taker. If `settlement` is set to `ACCOUNT`, the function might revert during the `debitAccount` function call if the taker does not have enough internal token balance.
- `this._processLimitAskOrder(ds, account, newOrder, args) -> this._executeAskLimitOrder(ds, newOrder, args.limitOrderType) -> CLOB._executeAskLimitOrder(ds, account, newOrder, args) -> _executeAskLimitOrder(ds, newOrder, args.limitOrderType)`
    - **What is controllable?** `newOrder`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the final posted amount for the new order — also, the `quoteToken` amount should be received by the account and `baseToken` should be provided by the account.
    - **What happens if it reverts, reenters or does other unusual control flow?** Can revert if the order type is `POST_ONLY` but the specified order `price` is less than the best price.
- `this._processLimitAskOrder(ds, account, newOrder, args) -> this._executeAskLimitOrder(ds, newOrder, args.limitOrderType) -> CLOB._executeAskLimitOrder(ds, account, newOrder, args) -> _executeAskLimitOrder(ds, newOrder, args.limitOrderType) -> _matchIncomingAsk(ds, newOrder, true) -> BookLib.getBestBid(ds)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the best maximum price `bidTree.minimum()`.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._processLimitAskOrder(ds, account, newOrder, args) -> this._executeAskLimitOrder(ds, newOrder, args.limitOrderType) -> CLOB._executeAskLimitOrder(ds, account, newOrder, args) -> _executeAskLimitOrder(ds, newOrder, args.limitOrderType) ->`

`_matchIncomingAsk(ds, newOrder, true) -> OrderLib.isExpired(bestBidOrder)`

- **What is controllable?** N/A.
- **If the return value is controllable, how is it used and how can it go wrong?** Returns true when the `cancelTimestamp` is not null and is less than the current `block.timestamp`; otherwise, it returns false.
- **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.

- `this._processLimitAskOrder(ds, account, newOrder, args) -> this._executeAskLimitOrder(ds, newOrder, args.limitOrderType) -> CLOB._executeAskLimitOrder(ds, account, newOrder, args) -> _executeAskLimitOrder(ds, newOrder, args.limitOrderType) -> _matchIncomingAsk(ds, newOrder, true) -> this._removeExpiredBid(ds, bestBidOrder) -> BookLib.getQuoteTokenAmount(ds, order.price, order.amount)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the quote-tokens amount calculated using the expired `order.price` and `order.amount`. The result can be rounded down to zero if `order.amount * order.price` is less than `config.baseSize`.
  - **What happens if it reverts, reenters or does other unusual control flow?** Can revert as a result of overflow during `order.amount * order.price` calculation if `order.price` or `order.amount` is too large.
- `this._processLimitAskOrder(ds, account, newOrder, args) -> this._executeAskLimitOrder(ds, newOrder, args.limitOrderType) -> CLOB._executeAskLimitOrder(ds, account, newOrder, args) -> _executeAskLimitOrder(ds, newOrder, args.limitOrderType) -> _matchIncomingAsk(ds, newOrder, true) -> this._removeExpiredBid(ds, bestBidOrder) -> TransientMakerData.addQuoteToken(order.owner, quoteTokenAmount)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** The `quoteTokenAmount` will be saved for the owner of the expired order in the `TransientMakerData` memory.
- `this._processLimitAskOrder(ds, account, newOrder, args) -> this._executeAskLimitOrder(ds, newOrder, args.limitOrderType) -> CLOB._executeAskLimitOrder(ds, account, newOrder, args) -> _executeAskLimitOrder(ds, newOrder, args.limitOrderType) -> _matchIncomingAsk(ds, newOrder, true) -> this._removeExpiredBid(ds, bestBidOrder) -> BookLib.removeOrderFromBook(ds, order)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?**

    This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** The `metadata.quoteTokenOpenInterest` and `metadata.baseTokenOpenInterest` will be decremented — depends on the side of the order, and `order.id` will be deleted from the `orders` list. Also, `bidTree` and `askTree` will be updated in addition to `orders.nextOrderId` and `orders.prevOrderId`.
- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) -> _removeNonCompetitiveOrder(ds, ds.orders[ds.askLimits[maxAskPrice].tailOrder]) -> factory.creditAccount(order.owner, address(ds.config.baseToken), baseRefunded)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** This function increases the internal account balance using the specified `baseRefunded` amount. But there is no verification that the actual factory balance is sufficient to replenish the account for this amount.
- `this._processLimitBidOrder(ds, account, newOrder, args) -> this._executeBidLimitOrder(ds, newOrder, args.limitOrderType) -> _removeNonCompetitiveOrder(ds, ds.orders[ds.askLimits[maxAskPrice].tailOrder]) -> ds.removeOrderFromBook(order)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** The `metadata.quoteTokenOpenInterest` and `metadata.baseTokenOpenInterest` will be decremented — depends on the side of the order, and `order.id` will be deleted from the `orders` list. Also, `bidTree` and `askTree` will be updated in addition to `orders.nextOrderId` and `orders.prevOrderId`.
- `this._processLimitAskOrder(ds, account, newOrder, args) -> this._executeAskLimitOrder(ds, newOrder, args.limitOrderType) -> CLOB._executeAskLimitOrder(ds, account, newOrder, args) -> _executeAskLimitOrder(ds, newOrder, args.limitOrderType) -> ds.addOrderToBook(newOrder)`
  - **What is controllable?** `newOrder` is partly controlled by the caller.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?**

Updates the book with a new order. Increases the `baseTokenOpenInterest` with the order amount since the side is `SELL`.

5.3. Module: GTERouter.sol

Function: `executeRoute(address tokenIn, uint256 amountIn, uint256 amountOutMin, uint256 deadline, bytes[] hops, ICLOB.Settlement settlement)`

This function allows to execute several actions, `executeClobPostFillOrder` or `executeUniV2SwapExactTokensForTokens`.

Inputs

- `tokenIn`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: The address of the tokens that will be provided by the caller to execute the desired action.
- `amountIn`
  - **Control**: Full control.
  - **Constraints**: The caller should own a sufficient amount of tokens to provide them for the call.
  - **Impact**: The amount of the `tokenIn` token will be provided for this call or will be used from the caller account in the clobFactory.
- `amountOutMin`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: This amount is used for a slippage check at the end of the call.
- `deadline`
  - **Control**: Full control.
  - **Constraints**: The `block.timestamp` cannot be more than the `deadline`.
  - **Impact**: The deadline for this call execution.
- `hops`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: The hop-specific data.
- `settlement`

    - **Control**: Full control.
    - **Constraints**: `INSTANT` or `ACCOUNT`.
    - **Impact**: Determines if settlement occurs with the caller's wallet or their account in clobFactory.

Branches and code coverage

Intended branches

- `settlement` is `ICLOB.Settlement.INSTANT`, and `hops[0][0:4]` is `this.executeClobPostFillOrder.selector`.
    - ☐ Test coverage
- `settlement` is `ICLOB.Settlement.ACCOUNT`, and `hops[0][0:4]` is `this.executeClobPostFillOrder.selector`.
    - ☐ Test coverage
- `settlement` is `ICLOB.Settlement.INSTANT`, and `hops[0][0:4]` is `this.executeUniV2SwapExactTokensForTokens.selector`.
    - ☐ Test coverage
- `settlement` is `ICLOB.Settlement.ACCOUNT`, and `hops[0][0:4]` is `this.executeUniV2SwapExactTokensForTokens.selector`.
    - ☐ Test coverage

Negative behavior

- An unsupported function selector is provided for the execution.
    - ☑ Negative test
- The result is less than `amountOutMin`.
    - ☐ Negative test
- Invalid CLOB address
    - ☑ Negative test

Function call analysis

- `this._handleWrap(tokenIn, amountIn, settlement)`
    - **What is controllable?** `tokenIn`, `amountIn`, and `settlement`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns `isWrapping` equal to `true`, in the case when the caller provides nonzero `msg.value` and `token` is zero, it means that provided `msg.value` has been deposited to the `weth` contract and `weth` should be used as an input-token address. Otherwise, it will return the `token` itself and `false`.

    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the caller has provided nonzero `msg.value` and nonzero `token` address and also if `msg.value` is not equal to the `amountIn` or `settlement` is `ACCOUNT`.
- `this._handleWrap(tokenIn, amountIn, settlement) -> this.weth.deposit{value: msg.value}`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `SafeTransferLib.safeApprove(address(this.weth), address(this.clobFactory), amountIn)`
    - **What is controllable?** `amountIn`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here. This function is used for the case `isWrapping == true`, when the funds inside the `deposit` function will be transferred from the router contract instead of the caller.
- `this.clobFactory.deposit(msg.sender, tokenIn, amountIn, isWrapping)`
    - **What is controllable?** `tokenIn` and `amountIn`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** The caller should set up the router contract address as an allowed operator to execute this function.
- `SafeTransferLib.safeTransferFrom(tokenIn, msg.sender, address(this), amountIn)`
    - **What is controllable?** `tokenIn` and `amountIn`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the allowance is not enough.
- `this.clobFactory.withdraw(msg.sender, tokenIn, amountIn, True)`
    - **What is controllable?** `tokenIn` and `amountIn`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the router is not an allowed operator or if the balance of the caller is not enough to withdraw `amountIn` tokens.

- `this._executeClobPostFillOrder(route, hops[i]) -> route.nextTokenIn.safeApprove(address(clobFactory), route.prevAmountOut)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** This function is called in the case when this is not a first execution and the previous has been `executeUniV2SwapExactTokensForTokens`, so the result of the previous swap should be deposited to the clobFactory. There are no problems here.
- `this._executeClobPostFillOrder(route, hops[i]) -> clobFactory.deposit(msg.sender, route.nextTokenIn, route.prevAmountOut, true);`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** This function is called to deposit to the clobFactory the result of the previous swap. There are no problems here.
- `this._executeClobPostFillOrder(route, hops[i]) -> market.postFillOrder(msg.sender, args);`
  - **What is controllable?** `args` (excluding `amount` and `settlement`).
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the result of the order fill.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the order type is `FILL_OR_KILL`, but it was not fully filled, or if the provided price is incorrect.
- `this._executeClobPostFillOrder(route, hops[i]) -> clobFactory.withdraw(msg.sender, tokenOut, amountOutFilled, true)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Withdraws funds from the clobFactory on behalf of the router to continue execution for the `executeUniV2SwapExactTokensForTokens` case.
- `this._executeClobPostFillOrder(route, hops[i]) -> clobFactory.withdraw(msg.sender, tokenOut, amountOutFilled, false)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?**

Withdraws the resulting tokens to the caller from the clobFactory at the final step of the execution, in the case `Settlement == INSTANT`.

- `this._executeUniV2SwapExactTokensForTokens(route, hops[i]) -> SafeTransferLib.safeApprove(path[0], address(this.uniV2Router), amountIn)`
    - **What is controllable?** `path[0]` and `amountIn`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._executeUniV2SwapExactTokensForTokens(route, hops[i]) -> this.uniV2Router.swapExactTokensForTokens(amountIn, amountOutMin, path, recipient, block.timestamp)`
    - **What is controllable?** `amountOutMin`, `path`, and `amountIn` (only in the case of the first execution).
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the results of the swap.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the result of the swap is less than `amountOutMin`.
- `this._handleUnwrap(route.prevAmountOut) -> weth.withdraw(amount)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the balance of the router is less than the requested `amount`.

5.4. Module: SimpleLaunchpad.sol

Function: `buy(address account, address token, address recipient, uint256 amountOutBase, uint256 maxAmountInQuote)`

This function allows any caller or trusted operator to buy the specified `amountOutBase` amount of the LaunchToken. The `maxAmountInQuote` specifies the maximum amount of quote tokens that can be spent.

Inputs

- `account`
    - **Control**: Full control.
    - **Constraints**: The caller should be the specified `account` itself or an approved operator by this `account`.
    - **Impact**: The account address on whose behalf the purchase is made.

- `token`
  - **Control**: Full control.
  - **Constraints**: `_launches` contains data related to this `token`.
  - **Impact**: The address of the trusted LaunchToken, created using the `launch` function.
- `recipient`
  - **Control**: Full control.
  - **Constraints**: Cannot be equal to the address of the Uniswap pool related to this LaunchToken and quote token.
  - **Impact**: The recipient of the purchased tokens.
- `amountOutBase`
  - **Control**: Full control.
  - **Constraints**: The total amount of LaunchToken tokens bought cannot exceed the `BONDING_SUPPLY` limit.
  - **Impact**: Amount of the LaunchToken to be bought.
- `maxAmountInQuote`
  - **Control**: Full control.
  - **Constraints**: The resulting `amountInQuoteActual` should be less than or equal to the `maxAmountInQuote`.
  - **Impact**: The maximum quote-tokens amount that can be spent.

Branches and code coverage

Intended branches

- `amountOutBaseActual` is less than `BONDING_SUPPLY`.
  - ☑ Test coverage
- `amountOutBaseActual` is greater than `BONDING_SUPPLY` but has been successfully limited by `BONDING_SUPPLY`.
  - ☑ Test coverage
- The remaining tokens are successfully swapped using the Uniswap pool.
  - ☑ Test coverage
- If the swap using the Uniswap pool fails, the remaining tokens are transferred to the caller.
  - ☑ Test coverage

Negative behavior

- The `token` status is not active, because it does not exist.

  - ☑ Negative test
- The `token` status is not active, because liquidity was transferred to the Uniswap pool.
  - ☑ Negative test
- The `maxAmountInQuote` is less than `amountInQuoteActual`.
  - ☑ Negative test

Function call analysis

- `LaunchToken(token).unlock()`
  - **What is controllable?** `token`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value here.
  - **What happens if it reverts, reenters or does other unusual control flow?** This function unlocks the LaunchToken transferring.
- `bondingCurve.buy(token, amountOutBaseActual)`
  - **What is controllable?** `token` and `amountOutBaseActual`.
  - **If the return value is controllable, how is it used and how can it go wrong?** The returned `amountInQuote` can be less or more than expected. If `amountInQuote` is less than expected, the user can buy tokens cheaper; otherwise, the resulting amount can be more expensive than expected but not more than `maxAmountInQuote`.
  - **What happens if it reverts, reenters or does other unusual control flow?** There is a `nonReentrant` modifier.
- `SafeTransferLib.safeTransfer(token, recipient, amountOutBaseActual)`
  - **What is controllable?** `recipient` and `amountOutBaseActual`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value here.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `SafeTransferLib.safeTransferFrom(address(data.quote), msg.sender, address(this), amountInQuote)`
  - **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 here.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `SafeTransferLib.safeApprove(token, address(uniV2Router), tokensToLock)`
  - **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 here.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `SafeTransferLib.safeApprove(address(data.quote), address(uniV2Router), data.quoteBoughtByCurve)`
  - **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 here.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this.uniV2Router.addLiquidity(token, address(this.quote), tokensToLock, data.quoteBoughtByCurve, tokensToLock, data.quoteBoughtByCurve, address(this), block.timestamp)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returned values are not used here.
  - **What happens if it reverts, reenters or does other unusual control flow?** If the pool already has liquidity, the initial price has already been determined. But since before that the `skim` function has been called, this is not the case.
- `this._swapRemaining(d) -> SafeTransferLib.safeTransferFrom(address(data.quote), msg.sender, address(this), data.quoteAmount)`
  - **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 here.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._swapRemaining(d) -> SafeTransferLib.safeApprove(address(this.quote), address(this.uniV2Router), data.quoteAmount)`
  - **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 here.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._swapRemaining(d) -> this.uniV2Router.swapTokensForExactTokens(data.baseAmount, data.quoteAmount, path, data.recipient, block.timestamp + 1)`
  - **What is controllable?** `data.recipient`.
  - **If the return value is controllable, how is it used and how can it go wrong?** The returned values are not used.

  - **What happens if it reverts, reenters or does other unusual control flow?** This function can revert if `amountInMax` is less than the resulting input amount.
- `this._swapRemaining(d) -> SafeTransferLib.safeApprove(address(data.quote), address(uniV2Router), 0)`
  - **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 here.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here. Reset the current approve to `0`.
- `this._swapRemaining(d) -> SafeTransferLib.safeTransfer(address(data.quote), msg.sender, data.quoteAmount)`
  - **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 here.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.

Function: `launch(string name, string symbol, string mediaURI)`

This function allows any caller to create a LaunchToken by paying a `launchFee`.

Inputs

- `name`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: The name of the new LaunchToken.
- `symbol`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: The symbol of the new LaunchToken.
- `mediaURI`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: The `mediaURI` of the new LaunchToken.

Branches and code coverage

Intended branches

- The new LaunchToken has been successfully created, and `TOTAL_SUPPLY` tokens have been minted.
  - ☐ Test coverage

Negative behavior

- `msg.value` is less than `launchFee`.
  - ☑ Negative test
- `msg.value` is greater than the `launchFee`.
  - ☐ Negative test
- `bondingCurve` address is not set up and is equal to zero.
  - ☐ Negative test
- `quoteAsset` address is not set up and is equal to zero.
  - ☐ Negative test

Function call analysis

- `bondingCurve.setReserves(token);`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** The `setReserves` function reverts if the caller is not this contract. But this is not the case, so there are no problems here.
- `LaunchToken(token).mint(TOTAL_SUPPLY);`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.

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 contracts, we discovered nine findings. One critical issue was found. Five were of medium impact and three were of low 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.