Zellic

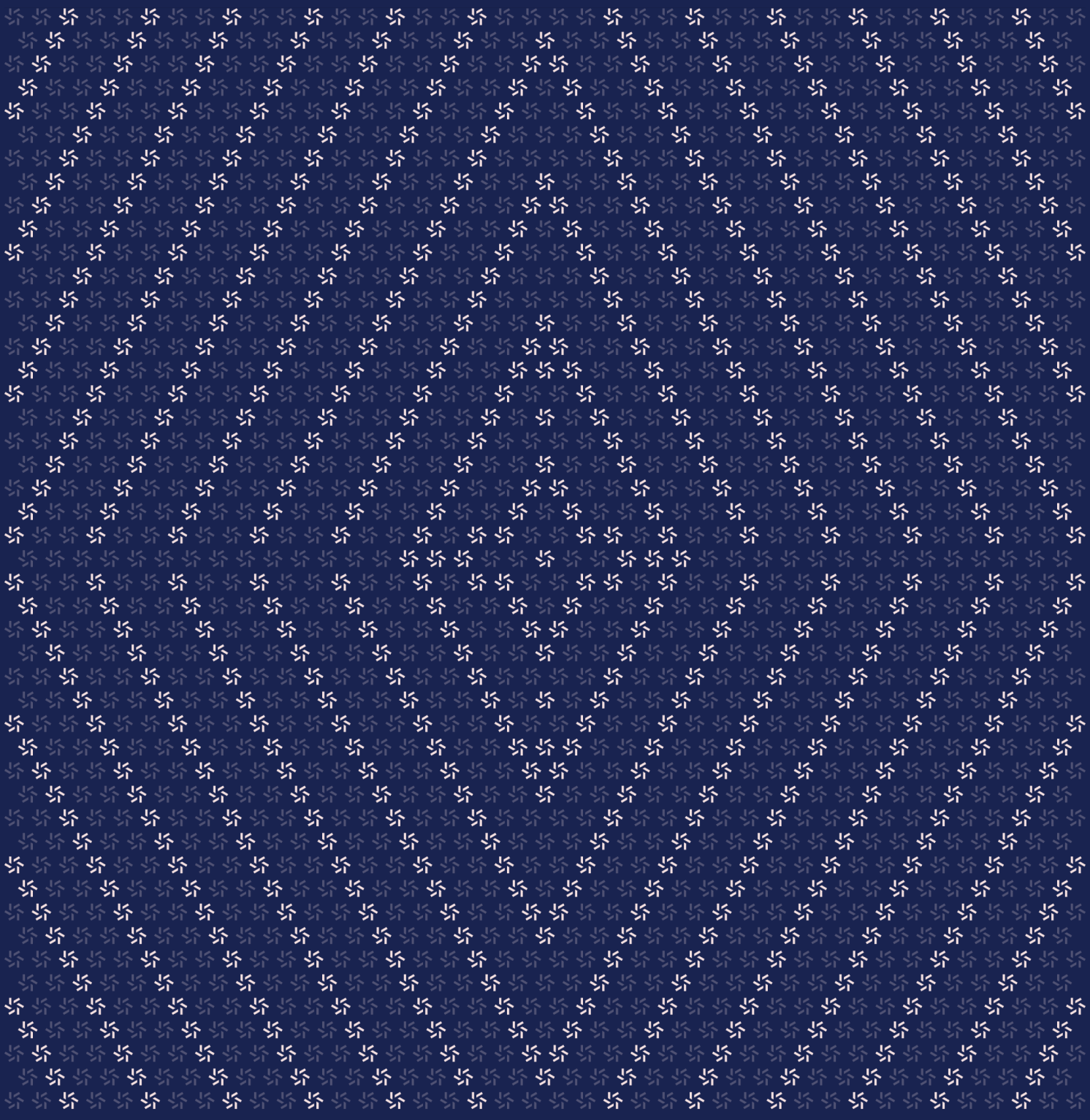
Prepared for
Matteo Lunghi
Liquid Labs, Inc.

Prepared by
Katerina Belotskaia
Juchang Lee
Zellic

March 7, 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 February 24th to February 28th, 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 authorization and access properly validated?
- Does the bonding process ensure safe usage without interruptions?
- Is the transition from the bonding curve to the AMM properly performed?
- Is there any risk of fund loss for users or contracts?

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 six findings. One critical issue was found. One was of high impact, one was of medium impact, two were of low impact, and the remaining finding was informational in nature.

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	1
Medium	1
Low	2
Informational	1

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. The current packages being audited of this monorepo are router/ and launcher/.

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	`040392bba2c4fcb7a61969c4d8edd54ad10eb683`

Programs	`MegaRouterFacet`
	`BondingCurve`
	`ApproxCurveVaryingSteps`
	`LaunchToken`
	`Launchpad`

2.4. Project Overview

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

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io ↗

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Katerina Belotskaia
Engineer
kate@zellic.io ↗

Juchang Lee
Engineer
lee@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

February 24, 2025	Kick-off call
February 24, 2025	Start of primary review period
February 28, 2025	End of primary review period

3. Detailed Findings

3.1. Attacker can manipulate the initial price in the Uniswap pool

Target	Launchpad		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	High	**Impact**	Critical

Description

During the buy phase, if the full `BONDING_SUPPLY` amount of the launch tokens is sold, the remaining liquidity (both LaunchToken and USDC) related to this token are moved to the Uniswap pool using the `router.addLiquidity` function.

```
function buy(address token, address recipient, uint256 amountOutBase,
    uint256 worstAmountInQuote)
    external
    nonReentrant
    returns (uint256 amountOutBaseActual, uint256 amountInQuoteActual)
{
    [...]
    if (nextAmountSold >= BONDING_SUPPLY) {
        data.active = false;
    [...]
    }

    [...]

    if (!data.active) {
        uint256 tokensToLock = TOTAL_SUPPLY - BONDING_SUPPLY;

        token.safeApprove(address(router), tokensToLock);
        address(quoteAsset).safeApprove(address(router),
    data.quoteBoughtByCurve);

        // slither-disable-next-line unused-return
        router.addLiquidity(
            token, address(quoteAsset), tokensToLock, data.quoteBoughtByCurve,
    0, 0, address(this), block.timestamp
        );
    [...]
    }
}
```

The `addLiquidity` function first checks if a pool already exists. If not, a new pool is created. Otherwise, this step is skipped.

Then, the function checks the current reserve amounts. If they are zero, the desired amounts of `tokenA` and `tokenB` are added to the pool as they are. Otherwise, the input amounts of tokens are adjusted based on the provided desired amounts.

First, the provided `amountADesired` is used. If the calculated `amountBOptimal` is less than or equal to `amountBDesired`, then `amountBOptimal` is used instead of `amountBDesired`. Otherwise, `amountAOptimal` is calculated using `amountBDesired`, and the resulting amount must be less than or equal to `amountADesired`.

The optimal amount is determined by the `quote` function of the UniswapV2Library, which uses the current reserve amounts `reserveA` and `reserveB` previously added to the pool. These values define the token ratio at which additional liquidity can be added. As a result, the `reserveA` and `reserveB` values dictate the current token price in the pool.

Consequently, the Launchpad contract will not provide the initially desired liquidity amounts (`tokensToLock` as LaunchToken and `data.quoteBoughtByCurve` as USDC). Instead, the liquidity provided will be either `tokensToLock` and `amountBOptimal` or `amountAOptimal` and `data.quoteBoughtByCurve`, depending on the current token ratio.

```
function _addLiquidity(
    [...]
) private returns (uint amountA, uint amountB) {
    // create the pair if it doesn't exist yet
    if (IUniswapV2Factory(factory).getPair(tokenA, tokenB) == address(0)) {
        IUniswapV2Factory(factory).createPair(tokenA, tokenB);
    }
    (uint reserveA, uint reserveB) = UniswapV2Library.getReserves(factory,
    tokenA, tokenB);
    if (reserveA == 0 && reserveB == 0) {
        (amountA, amountB) = (amountADesired, amountBDesired);
    } else {
        uint amountBOptimal = UniswapV2Library.quote(amountADesired, reserveA,
    reserveB);
        if (amountBOptimal <= amountBDesired) {
            require(amountBOptimal >= amountBMin, 'UniswapV2Router:
    INSUFFICIENT_B_AMOUNT');
            (amountA, amountB) = (amountADesired, amountBOptimal);
        } else {
            uint amountAOptimal = UniswapV2Library.quote(amountBDesired,
    reserveB, reserveA);
            assert(amountAOptimal <= amountADesired);
            require(amountAOptimal >= amountAMin, 'UniswapV2Router:
    INSUFFICIENT_A_AMOUNT');
            (amountA, amountB) = (amountAOptimal, amountBDesired);
        }
```

```
    }
}
```

To reach this state, `reserveA` and `reserveB` should be set to nonzero values, meaning liquidity must already be added to the pool before the Launchpad triggers the `addLiquidity` function.

The USDC token can be transferred to the pool without any restrictions. However, LaunchToken transfers are locked until the full `BONDING_SUPPLY` amount is sold. Despite this, tokens can still be transferred from the Launchpad contract during the execution of the `buy` function. As a result, a malicious user could set the Uniswap pool address as the `recipient` and provide the necessary liquidity to the pool, manipulating the token ratio to their advantage.

```
function _beforeTokenTransfer(address from, address to, uint256 /*amount*/ )
    internal view override {
    if (!unlocked && from != launcher && to != launcher) {
        revert TransfersDisabledWhileBonding();
    }
}
```

Impact

An attacker can manipulate the initial liquidity setup of the Uniswap V2 pool for LaunchToken and, consequently, control the token price before the Launchpad contract executes `addLiquidity`.

By initializing a new Uniswap V2 pair for LaunchToken and USDC, the attacker can predefine the price by buying the required amount of LaunchToken and transferring it directly to the pool, setting the pool address as the `recipient`. Then, the attacker can add USDC liquidity at a desired price and call `UniswapV2Pair.mint` to lock in the reserves (`reserveA` and `reserveB`).

Once the full `BONDING_SUPPLY` is sold, the Launchpad contract calls `addLiquidity`. However, because the reserves have already been manipulated, the function will not add the full intended liquidity but will instead adjust the amounts to maintain the attacker's predefined price.

As a result, the attacker can immediately sell their LaunchToken, extracting more USDC than they would have under normal market conditions.

Recommendations

We recommend restricting the transfer of LaunchToken during the buy phase to the precalculated Uniswap pair address.

Remediation

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

- 801bea8d ↗
- ca3e56d0 ↗

Liquid Labs, Inc. provided the following response:

> The first commit prevented donations by computing the pool address and preventing it from being the recipient field in launchpad buys (where you receive the launch token). It also added a stored pool address to try and prevent not just base token donations but donating quote to other pre bonding pools. However, we realized this is not necessary, as anyone with quote (like usdc) can just donate regardless. The second commit removes the futile attempt to prevent single sided quote token donations

3.2. Purchase token with zero USDC repeat

Target	Launchpad		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	High	**Impact**	High

Description

Function `_getAverageCostInY` performs the actual trade on the curve for an amount of tokens, returning the cost in USDC scaled.

First, `remainingDistance`, `isForward`, and `stepRemaining` are defined as below:

```
uint256 remainingDistance = fMath.dist(x_0, x_1) // x_1 - x_0. influenced by
    the buy amount.
bool isForward = x_1 >= x_0; // In buy process, most case follow: x_1 >= x_0
uint256 finalX = lastStep.x_rel;
stepRemaining = finalX > 0 ? (isForward ? currentStep.dx - finalX : finalX) :
    currentStep.dx;
```

When the buy amount is sufficiently small, `remainingDistance` becomes less than `stepRemaining`. In this case, the execution will pass through the conditional statement below,

```
if (remainingDistance <= stepRemaining) {
    // Final partial step
    uint256 y = _calculateY(currentStep, remainingDistance);
    totalY += y;
    // [...]
    if (remainingDistance == stepRemaining) {
        finalIdx = isForward ? uint16(finalIdx + 1) : finalIdx > 0 ?
    uint16(finalIdx - 1) : 0;
    }
    // [...]
    break;
}
```

and the `_calculateY` return value will be `totalY`, the average value Y.

This value will be divided with `PRECISION_MULTIPLIER` and `QUOTE_SCALING`. Consequently, if the value of Y at this point is less than `PRECISION_MULTIPLIER * QUOTE_SCALING`, it will be rounded down to 0. This enables the purchase of tokens for zero USDC.

During this process, `finalIdx` is only updated when `remainingDistance` equals `stepRemaining`. Consequently, if these values are inconsistent, `finalIdx` remains unchanged.

Subsequently, `newLastStep` is returned as shown below. This return value will subsequently alter `currentStep`.

```
newLastStep = LastStepData({
    i: finalIdx,
    x_rel: remainingDistance < stepRemaining
        ? (isForward ? remainingDistance : stepRemaining - remainingDistance)
        : 0,
    x_abs: uint256(int256(lastStep.x_abs) + totalStepDeltaX)
});
```

When `remainingDistance` is less than `stepRemaining`, during a buy operation, since `isForward` is true, `x_rel` becomes `stepRemaining`.

If the same `amount` is subsequently bought again, the value of `remainingDistance` remains unchanged, while `stepRemaining` becomes `currentStep.dx - lastStep.x_rel(=remainingDistance)`.

In this case as well, if `remainingDistance` is less than `stepRemaining`, the above process repeats, potentially allowing for infinite repetition of this cycle.

Impact

To execute this attack, the quantity of LaunchToken that can be purchased at once is approximately 1e17, which is significantly small compared to the total supply of 1e27. However, this method enables unlimited initial purchases at zero USDC. This vulnerability has substantial potential impact if the token price increases due to factors such as listing on Uniswap pools or similar events.

Recommendations

Either enhance the division policy to prevent losses or implement precise step updates. While blocking purchases at zero USDC is another potential solution.

Remediation

This issue has been acknowledged by Liquid Labs, Inc., and a fix was implemented in commit df320180 ↗.

Liquid Labs, Inc. provided the following response:

The commit goes with Zellic's second recommendation of cleaning the amountIn using (amountIn / 1e18) * 1e18. This prevents buys that are small enough from costing 0. However, we decided for the next audit, which covers SimpleLaunchpad and SimpleBondingCurve (less precision loss prone implementations of this curve) to go with Zellic's first recommendation, which is to revert if the cost is 0

3.3. Remove launchpadSell function from MegaRouterFacet

Target	MegaRouterFacet		
Category	Business Logic	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

The `launchpadSell` function allows a user to sell a specified amount of `LaunchToken` through the trusted Launchpad contract. It transfers `amountInBase` `LaunchToken` from the caller to the contract, approves the Launchpad contract to spend them, and then executes the sale via the `sell` function.

```
function launchpadSell(address token, uint256 amountInBase,
    uint256 worstAmountOutQuote)
    external
    returns (uint256 baseSpent, uint256 quoteBought)
{
    token.safeTransferFrom(msg.sender, address(this), amountInBase);

    token.safeApprove(address(launchpad), amountInBase);

    return launchpad.sell(token, msg.sender, amountInBase,
    worstAmountOutQuote);
}
```

The problem is that `LaunchToken` are locked for transfers until the `BONDING_SUPPLY` amount of tokens is sold. Tokens can only be transferred to or from the `Launchpad` contract.

Impact

The caller cannot provide the tokens to this contract, making the function unusable. And after the `BONDING_SUPPLY` amount of tokens is sold, all remaining tokens will be moved to the UniswapV2 pool, making this function unusable as well.

The caller cannot provide the tokens to this contract, making the function unusable during the bonding curve phase. Additionally, once the `BONDING_SUPPLY` amount of tokens is sold, all remaining tokens will be moved to the UniswapV2 pool, causing this function to remain unusable indefinitely.

Recommendations

Since the `launchpadSell` function cannot be used at any point due to the transfer restrictions on `LaunchToken`, we recommend removing this function entirely from the contract.

Remediation

This issue has been acknowledged by Liquid Labs, Inc., and a fix was implemented in commit `df320180` ↗.

Liquid Labs, Inc. provided the following response:

> This commit removed launchpad.Sell from the router, however for the next audit, we decided to beef up the pre-graduation transfer guards in LaunchToken to allow the router to transfer and sell

3.4. MegaRouterFacet does not validate that `clob` is trusted

Target	MegaRouterFacet		
Category	Business Logic	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The MegaRouterFacet contract provides functions that act as a wrapper for multicalls, for example, `clobCancel` or `clobPostLimitOrder`. These functions accept an arbitrary `clob` contract address provided by the caller and do not perform validation to ensure that this contract was created using the trusted clobFactory contract.

```
/// @notice A clob cancel wrapper for multicalls
function clobCancel(ICLOB clob, ICLOB.CancelArgs calldata args)
    external override {
    clob.cancel(msg.sender, args);
}

/// @notice A clob post limit order wrapper for multicalls
function clobPostLimitOrder(ICLOB clob, ICLOB.PostLimitOrderArgs memory args)
    external override {
    args.settlement = ICLOB.Settlement.ACCOUNT;
    clob.postLimitOrder(msg.sender, args);
}
```

Impact

Since the functions accept an arbitrary `clob` contract address without validation, users can call any contract, including malicious or unintended ones. This could lead to unauthorized interactions and potential security vulnerabilities, as the contract provided by the caller can execute arbitrary logic.

Recommendations

Add verification that clob market has been created using the trusted clobFactory.

Remediation

This issue has been acknowledged by Liquid Labs, Inc., and a fix was implemented in commit 3f807fdf.

Liquid Labs, Inc. provided the following response:

> This commit adds a validity check to functions that take an arbitrary clob address in the router, ensuring that it is a clob created by the clob factory (immutable).

3.5. The Uniswap router contract still retains an approval after an unsuccessful swap

Target	Launchpad		
Category	Business Logic	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The `buy` function of the Launchpad contract will execute `_swapRemainingQuote` to attempt swapping the remaining USDC tokens if the available LaunchToken supply, before reaching `BONDING_SUPPLY`, was insufficient. After moving liquidity to the Uniswap pair contract, the Uniswap router will be assigned the remaining USDC tokens, and `router.swapExactTokensForTokens` will be executed within a try-catch block. If the swap execution reverts, the remaining USDC tokens will be transferred back to the user. However, in this case, the router will still retain approval for the transfer of these tokens.

```
function _swapRemainingQuote(address token, address recipient,
    uint256 remainingQuote)
    internal
    returns (uint256, uint256)
{
    // Transfer the remaining quote from the user
    address(quoteAsset).safeTransferFrom(msg.sender, address(this),
    remainingQuote);

    // Prepare swap path
    address[] memory path = new address[](2);
    path[0] = address(quoteAsset);
    path[1] = token;

    // Approve router to spend remaining quote
    address(quoteAsset).safeApprove(address(router), remainingQuote);

    try router.swapExactTokensForTokens(
        remainingQuote,
        1, // Accept any amount (user has already authorized
    worstAmountInQuote)
        path,
        recipient,
        block.timestamp + 300
    ) returns (uint256[] memory amounts) {
```

```
            // Return the tokens received and quote used
            return (amounts[1], remainingQuote);
        } catch {
            // If swap fails, return the additional quote tokens to the user
            address(quoteAsset).safeTransfer(msg.sender, remainingQuote);
            return (0, 0);
        }
    }
```

Impact

If the `swapExactTokensForTokens` execution reverts, the Uniswap router will still retain approval for transferring USDC tokens from the Launchpad contract. This creates a security risk, as the approved router contract could potentially be used to swap and transfer USDC tokens without additional authorization, leading to a potential loss of funds from the Launchpad contract.

The current router implementation does not allow setting a custom sender address and only uses `msg.sender`. However, if the router were to support specifying an arbitrary sender address in the future, the severity of this issue would increase significantly, as it could enable an attacker to directly swap tokens from the Launchpad contract.

Recommendations

We recommend revoking the approval to prevent a scenario where USDC tokens from the Launchpad could be used through the router contract after an unsuccessful swap.

Remediation

This issue has been acknowledged by Liquid Labs, Inc., and a fix was implemented in commit `15eadc2b` ↗.

Liquid Labs, Inc. provided the following response:

> This commit (same as 3.3) also addresses 3.5 by revoking token approvals to the uniswap router if the graduation slippage spillover swap fails.

3.6. Usage of `msg.sender.transfer()` function

Target	Launchpad		
Category	Business Logic	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The `pullFees` calls the `transfer` function to send the full contract balance of Ether to the `msg.sender` account.

Impact

The `transfer` function uses a hardcoded amount of GAS and will fail if GAS costs increase in the future, so it is no longer recommended for use.

Recommendations

As best practice ↗, consider using the `msg.sender.call.value(value)("")` function:

```
(bool success, ) = msg.sender.call.value(amounts[1].sub(feeAmount))("");
require(success, "Transfer failed.");
```

Remediation

This issue has been acknowledged by Liquid Labs, Inc., and a fix was implemented in commit 06303e34 ↗.

Liquid Labs, Inc. provided the following response:

> This commit uses a call to pull eth fees from the launchpad instead of transfer to prevent out of gas exceptions

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. The `executeRoute` does not revert on unsupported function selector

The `executeRoute` function processes hops data, which specifies the necessary details for executing multiple functions in sequence. However, when an unsupported function selector is provided, the function does not revert but instead proceeds to the next hop.

We recommend modifying the `executeRoute` function to explicitly revert when encountering an unsupported function selector. This will ensure that any invalid input is properly handled, maintaining the expected execution flow.

```
function executeRoute(
    [...]
) external override {
    RouteMetadata memory route;

    if (settlement == ICLOB.Settlement.INSTANT && bytes4(hops[0][0:4]) ==
    this.executeClobPostFillOrder.selector) {
        clobFactory.deposit(msg.sender, tokenIn, amountIn, false);
    } else if (settlement == ICLOB.Settlement.INSTANT) {
        tokenIn.safeTransferFrom(msg.sender, address(this), amountIn);
    } else if (bytes4(hops[0][0:4]) ==
    this.executeUniV2SwapExactTokensForTokens.selector) {
        clobFactory.withdraw(msg.sender, tokenIn, amountIn, true);
    }

    [...]

    for (uint256 i = 0; i < hops.length; i++) {
        bytes4 currSelector = bytes4(hops[i][0:4]);
        route.nextSelector = i == hops.length - 1 ? bytes4(0) : bytes4(hops[i
    + 1][0:4]);

        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]);
        }
```

```
            route.prevSelector = currSelector;
        }

        [...]
    }
```

This issue has been acknowledged by Liquid Labs, Inc., and a fix was implemented in commit 2bc39fcf ↗.

Liquid Labs, Inc. provided the following response:

> This commit adds a revert to route dispatching if the selector is not recognized

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: Launchpad.sol

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

This function allows buying `amountOutBase` tokens, but the amount of USDC to be spent is limited by `worstAmountInQuote`. After selling the `BONDING_SUPPLY` amount of LaunchToken, the remaining liquidity will be transferred to the Uniswap pool, and the current function will become unavailable for this `token`.

Inputs

- `token`
  - **Control**: Full control.
  - **Constraints**: The status `active` of this token should be `true`, and `launches` should contain the `token` info.
  - **Impact**: The address of the LaunchToken to be bought.
- `recipient`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: The address of the recipient of LaunchToken.
- `amountOutBase`
  - **Control**: Full control.
  - **Constraints**: The total sold amount of LaunchToken cannot exceed `BONDING_SUPPLY`.
  - **Impact**: The desired amount of LaunchToken to be bought.
- `worstAmountInQuote`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: The limited amount of USDC to be spent.

Branches and code coverage

Intended branches

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

Negative behavior

- The `token` status is not active, because it doesn't exists
  - ☑ Negative test
- The `token` status is not active, because liquidity was transferred to the Uniswap pool.
  - ☑ Negative test
- `worstAmountInQuote` is less than `amountInQuote`.
  - ☑ 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.
- `data.bondingCurve.getAverageCostInY(token, this.baseToX(data.baseSoldFromCurve), this.baseToX(nextAmountSold))`
  - **What is controllable?** N/A.
  - **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 than expected; otherwise, the resulting amount can be more expensive than expected but not more than `worstAmountInQuote`.

    - **What happens if it reverts, reenters or does other unusual control flow?** It has a `nonReentrant` modifier.
- `SafeTransferLib.safeTransfer(token, recipient, amountOutBase)`
    - **What is controllable?** `recipient` and `amountOutBase`.
    - **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(this.quoteAsset), 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(this.router), 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(this.quoteAsset), address(this.router), 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.router.addLiquidity(token, address(this.quoteAsset), tokensToLock, data.quoteBoughtByCurve, 0, 0, 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. Therefore, only the portion of liquidity that matches the current rate will be added, rather than the full amount.
- `this._swapRemainingQuote(token, recipient, remainingQuote) -> SafeTransferLib.safeTransferFrom(address(this.quoteAsset), msg.sender, address(this), remainingQuote)`

- **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._swapRemainingQuote(token, recipient, remainingQuote) -> SafeTransferLib.safeApprove(address(this.quoteAsset), address(this.router), remainingQuote)`
  - **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._swapRemainingQuote(token, recipient, remainingQuote) -> this.router.swapExactTokensForTokens(remainingQuote, 1, path, recipient, block.timestamp + 300)`
  - **What is controllable?** `recipient`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the output amount of LaunchToken received as a result of the swap. This amount is added to the total LaunchToken received by the recipient.
  - **What happens if it reverts, reenters or does other unusual control flow?** this function can revert if `amountOutMin` is less than the resulting output amount, but since it is called with `amountOutMin` set to `1`, it is unlikely to revert.
- `this._swapRemainingQuote(token, recipient, remainingQuote) -> SafeTransferLib.safeTransfer(address(this.quoteAsset), msg.sender, remainingQuote)`
  - **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)`

Anyone can launch a new LaunchToken instance, but the `LAUNCH_FEE` must be paid.

Inputs

- `name`
  - **Control**: Full control.
  - **Constraints**: No constraints.

- **Impact**: The name for the new LaunchToken.

- `symbol`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: The symbol for the new LaunchToken.
- `mediaURI`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: The media URI for 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 `LAUNCH_FEE`.
  - ☐ Negative test
- `bondingCurve` address is not set up and is equal to zero.
  - ☐ Negative test

Function call analysis

- `LaunchToken(token).mint(Launchpad.TOTAL_SUPPLY)`
  - **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?** N/A.

Function: `sell(address token, address recipient, uint256 amountInBase, uint256 worstAmountOutQuote)`

This function allows selling the previously bought LaunchToken tokens but only during the bonding-curve sale phase.

Inputs

- `token`
  - **Control**: Full control.
  - **Constraints**: The status `active` of this token should be `true`, and `launches` should contain the `token` info.
  - **Impact**: The address of the LaunchToken to be sold.
- `recipient`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: The address of the recipient of USDC.
- `amountInBase`
  - **Control**: Full control.
  - **Constraints**: The caller should own at least `amountInBase` tokens.
  - **Impact**: The desired amount of LaunchToken to be sold.
- `worstAmountOutQuote`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: The minimum amount of USDC to be received as result of selling LaunchToken.

Branches and code coverage

Intended branches

- Tokens have been sold successfully.
  - ☑ Test coverage

Negative behavior

- The caller owns fewer LaunchToken tokens than `amountInBase`.
  - ☐ Negative test
- The `token` status is not active, because it doesn't exists
  - ☑ Negative test
- The `token` status is not active, because liquidity was transferred to the Uniswap pool.
  - ☑ Negative test
- `worstAmountInQuote` is greater than `amountInQuote`.
  - ☑ Negative test

Function call analysis

- `data.bondingCurve.getAverageCostInY(token, this.baseToX(data.baseSoldFromCurve), this.baseToX(nextAmountSold))`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** The returned `amountOutQuote` can be less or more than expected. If `amountOutQuote` is less than expected, the user can sell tokens cheaper than expected but not less than `worstAmountInQuote`; otherwise, the resulting amount can be more than expected, so the user will sell tokens at an inflated price.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `SafeTransferLib.safeTransferFrom(token, msg.sender, address(this), amountInBase)`
    - **What is controllable?** `amountInBase`.
    - **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.safeTransfer(address(this.quoteAsset), recipient, amountOutQuote)`
    - **What is controllable?** `recipient`.
    - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value here.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.

5.2. Module: MegaRouterFacet.sol

Function: `clobCancel(ICLOB clob, ICLOB.CancelArgs args)`

This function allows the cancellation of orders identified by the provided `args.orderIds` array.

Inputs

- `clob`
    - **Control**: Full control.
    - **Constraints**: No constraints.
    - **Impact**: The address of the contract on which the `cancel` function will be called.

- `args`
    - **Control**: Full control.
    - **Constraints**: The caller should be an owner of the provided `args.orderIds`.
    - **Impact**: Contains the `orderIds` and the settlement type, `INSTANT` or `ACCOUNT` — `ACCOUNT` means that funds will be added to the user's balance in the CLOBManager contract, while `INSTANT` means that tokens will be transferred directly to the user.

Branches and code coverage

Intended branches

- Orders have been successfully closed.
    - ☐ Test coverage

Negative behavior

- The caller is not an owner of the provided `args.orderIds`.
    - ☐ Negative test

Function call analysis

- `clob.cancel(msg.sender, args)`
    - **What is controllable?** `clob` and `args`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Return values are not used here.
    - **What happens if it reverts, reenters or does other unusual control flow?** There is a potential problem because the `clob` contract address is fully controlled by the caller. Reentrancy is possible here. For more detailed information, refer to the `3.5` finding description (3.4. ↗).

Function: `clobDeposit(address token, uint256 amount, bool fromRouter)`

This function allows depositing the specified `amount` of `token` to the CLOBManager contract.

Inputs

- `token`
    - **Control**: Full control.
    - **Constraints**: CLOBManager allows depositing arbitrary tokens.

- **Impact**: The address of the token will be deposited to the CLOBManager contract.

- `amount`
  - **Control**: Full control.
  - **Constraints**: The caller should own enough tokens to transfer them to the CLOBManager contract.
  - **Impact**: The amount of token to be transferred to the CLOBManager contract.
- `fromRouter`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: If `fromRouter` is `true`, then the tokens are transferred to this contract, and approval is granted to the `CLOBManager` contract to use these tokens. Otherwise, tokens will be transferred from the caller account.

Branches and code coverage

Intended branches

- Tokens are successfully deposited when `fromRouter` is true.
  - ☑ Test coverage
- Tokens are successfully deposited when `fromRouter` is false.
  - ☑ Test coverage

Negative behavior

- The amount exceeds the caller's balance of the `token`.
  - ☐ Negative test

Function call analysis

- `SafeTransferLib.safeTransferFrom(token, msg.sender, 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?** 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(this.clobFactory), amount)`
  - **What is controllable?** `token` and `amount`.
  - **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.clobFactory.deposit(msg.sender, token, amount, fromRouter)`
  - **What is controllable?** `token`, `amount`, and `fromRouter`.
  - **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: `clobPostLimitOrder(ICLOB clob, ICLOB.PostLimitOrderArgs args)`

This function allows triggering the `postLimitOrder` function of the specified `clob` contract. The provided order will be executed and, if not fully filled, added to the order book.

Inputs

- `clob`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: The address of the contract on which the `postLimitOrder` function will be called.
- `args`
  - **Control**: Full control.
  - **Constraints**: `settlement` is set up to `ACCOUNT`.
  - **Impact**: Contains `amountInBaseLots`, `priceInTicks`, `cancelTimestamp`, `side`, `limitOrderType`, and `settlement`.

Branches and code coverage

Negative behavior

- The resulting amount of tokens exceeds the caller's balance in the CLOBManager.
  - ☐ Negative test

Function call analysis

- `clob.postLimitOrder(msg.sender, args)`
  - **What is controllable?** `clob` and `args`.

- **If the return value is controllable, how is it used and how can it go wrong?** Return values are not used here.
- **What happens if it reverts, reenters or does other unusual control flow?** There is a potential problem because the `clob` contract address is fully controlled by the caller. Reentrancy is possible here. For more detailed information, refer to the `3.5` finding description (3.4. ↗).

Function: `clobWithdraw(address token, uint256 amount)`

This function allows withdrawing the specified `amount` of `token` from the CLOBManager contract, given that it has been previously deposited. The caller of the function will receive the withdrawn tokens.

Inputs

- `token`
  - **Control**: Full control.
  - **Constraints**: CLOBManager allows to deposit and withdraw arbitrary tokens.
  - **Impact**: The address of the token will be withdrawn from the CLOBManager contract.
- `amount`
  - **Control**: Full control.
  - **Constraints**: There are no constraints here. However, `withdraw` verifies that `amount` cannot be more than the user's balance.
  - **Impact**: The amount of tokens will be withdrawn from the CLOBManager.

Branches and code coverage

Intended branches

- Tokens have been sucessfully withdrawn.
  - ☑ Test coverage

Negative behavior

- The caller has not deposited any tokens.
  - ☐ Negative test
- The amount exceeds the caller's balance in the CLOBManager.
  - ☐ Negative test

Function call analysis

- `this.clobFactory.withdraw(msg.sender, token, amount, False)`
  - **What is controllable?** `token` and `amount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** There are no return values here.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.

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

This function allows to execute several actions, `postFillOrder` or `swapExactTokensForTokens`.

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 the sufficient amount of tokens to provide them for the call.
  - **Impact**: The amount of the `tokenIn` token will be provided for this call.
- `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 CLOBManager.

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

- Unsupported function selector is provided for the execution.
    - ☐ Negative test
- The result is less than `amountOutMin`.
    - ☐ Negative test

Function call analysis

- `this.clobFactory.deposit(msg.sender, tokenIn, amountIn, False)`
    - **What is controllable?** `tokenIn` and `amountIn`.
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the caller has assigned insufficient approval for clobFactory or if the caller's balance is not enough to transfer `amountIn`.

- `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?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the caller has assigned insufficient approval for this contract or if the caller's balance is not enough to transfer `amountIn`.
- `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?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if `msg.sender` balance in the clobFactory is not enough to withdraw.
- `this._executeClobPostFillOrder(route, hops[i]) -> SafeTransferLib.safeApprove(route.nextTokenIn, address(this.clobFactory), route.prevAmountOut)`
    - **What is controllable?** `route.nextTokenIn`.
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._executeClobPostFillOrder(route, hops[i]) -> this.clobFactory.deposit(msg.sender, route.nextTokenIn, route.prevAmountOut, True)`
    - **What is controllable?** `route.nextTokenIn`.
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** This function is executed when it is not the first hop and the previous hop was not `executeClobPostFillOrder.selector`. This means that the contract retains the resulting tokens, which should be deposited into the `clobFactory` to make these funds available for the current hop execution.
- `this._executeClobPostFillOrder(route, hops[i]) -> this._getClobTokenOutAndBaseLotsIn(route.prevAmountOut, market, args) -> market.getQuoteToken()`
    - **What is controllable?** `market` and `args`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the address of the Quote token from the specified `market` contract, which will be used as a `tokenOut` in the case of `SELL`.
    - **What happens if it reverts, reenters or does other unusual control flow?**

There are no problems here.

- `this._executeClobPostFillOrder(route, hops[i]) -> this._getClobTokenOutAndBaseLotsIn(route.prevAmountOut, market, args) -> market.getBaseTokenAmountToBaseLots(amountIn)`
  - **What is controllable?** `market`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns `amount / self.config.baseLotSize`, which can be zero as a result of rounding down if `amount` is less than `baseLotSize`.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._executeClobPostFillOrder(route, hops[i]) -> this._getClobTokenOutAndBaseLotsIn(route.prevAmountOut, market, args) -> market.getBaseToken()`
  - **What is controllable?** `market` and `args`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the address of the BASE token from the specified `market` contract, which will be used as a `tokenOut` in the case of `BUY`.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._executeClobPostFillOrder(route, hops[i]) -> this._getClobTokenOutAndBaseLotsIn(route.prevAmountOut, market, args) -> market.getQuoteTokenAmountToBaseLots(amountIn, args.priceInTicks)`
  - **What is controllable?** `market`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns `(quoteAmount * self.config.baseLotsPerBaseUnit)/ (priceInTicks * self.config.tickSizeInQuoteLotsPerBaseUnit * self.config.quoteLotSize)`, which can be zero as a result of rounding down if `quoteAmount * self.config.baseLotsPerBaseUnit` is less than `priceInTicks * self.config.tickSizeInQuoteLotsPerBaseUnit * self.config.quoteLotSize`.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `this._executeClobPostFillOrder(route, hops[i]) -> market.postFillOrder(msg.sender, args)`
  - **What is controllable?** `market` and `args`.
  - **If the return value is controllable, how is it used and how can it go wrong?** if the resulting `prevAmountOut` is less than `amountOutMin`, the function reverts.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the provided `priceInTicks` is invalid or if `amountInBaseLots` is less than `MIN_FILL_ORDER_AMOUNT_BASE_LOTS`. It also reverts if `fillOrderType` is `FILL_OR_KILL` but the order has not been fully filled.
- `this._executeClobPostFillOrder(route, hops[i]) ->`

`this.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?** N/A.
- **What happens if it reverts, reenters or does other unusual control flow?** This function executes in the middle hop when the next hop is not `executeClobPostFillOrder.selector`.

- `this._executeClobPostFillOrder(route, hops[i]) -> this.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?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** This function executes in the last hop when the settlement is `INSTANT` to transfer resulting tokens to the user.
- `this._executeUniV2SwapExactTokensForTokens(route, hops[i]) -> SafeTransferLib.safeApprove(path[0], address(this.uniV2Router), amountIn)`
  - **What is controllable?** `amountIn`.
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **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` and `path`.
  - **If the return value is controllable, how is it used and how can it go wrong?** The resulting amounts can be less than expected.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.

6. Assessment Results

At the time of our assessment, the reviewed code had not yet been deployed to the MegaETH mainnet.

Recommended Next Steps:

- Deploy to the MegaETH testnet as soon as it becomes available, to validate the intended behavior under real-world conditions.
- Expand test coverage for the MegaRouterFacet components by ensuring all functions are covered and by implementing more complex test scenarios for the Launchpad contract — particularly those involving interactions with `IUniswapV2Router`.
- Consider improving code documentation to support long-term maintainability and onboarding of new contributors.

While the foundation is solid, addressing these recommendations will provide greater confidence ahead of a mainnet deployment.

During our assessment on the scoped GTE contracts, we discovered six findings. One critical issue was found. One was of high impact, one was of medium impact, two were of low impact, and the remaining finding was informational in nature.

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.