OpenZeppelin | security

Uniswap Liquidity Launcher Audit

Uniswap

January 23, 2026

Table of Contents

Table of Contents 2

Summary 3

Scope 4

System Overview 6

Liquidity Launcher 6

Security Model and Privileged Roles 7

Medium Severity 8

M-01 collectFees Can Be Maliciously Front-Run 8

Low Severity 8

L-01 Invalid Position Creation Can Be Attempted When The Current Tick Is Too Close to MIN or MAX Tick 8

L-02 getAddress Can Return CREATE2 Addresses That Can Never Be Deployed 9

L-03 Missing Docstrings or Incomplete Docstrings 10

L-04 Interpretation of Block Number Varies Across Networks 10

Notes & Additional Information 11

N-01 Missing Zero-Address Check for Underlying Token 11

N-02 Unused Error 11

N-03 Unnecessary Cast 11

N-04 Possible Duplicate Event Emission 12

N-05 Misleading or Incorrect Comments 12

Conclusion 13

Summary

Type	DeFi
Timeline	From 2025-12-22 To 2026-01-14
Languages	Solidity

Total Issues	10 (8 resolved, 1 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	4 (3 resolved, 1 partially resolved)
Notes & Additional Information	5 (4 resolved)
Client Reported Issues	0 (0 resolved)

Scope

OpenZeppelin audited Uniswap/liquidity-launcher/ repository - all the changes made between commits bde110c and 412df26.

In scope were the following files:

```
contracts
└── src
    ├── LiquidityLauncher.sol
    ├── Multicall.sol
    ├── Permit2Forwarder.sol
    │
    ├── factories
    │   ├── StrategyFactory.sol
    │   │
    │   ├── lbp
    │   │   ├── AdvancedLBPStrategyFactory.sol
    │   │   ├── FullRangeLBPStrategyFactory.sol
    │   │   └── GovernedLBPStrategyFactory.sol
    │   │
    │   └── periphery
    │       └── MerkleClaimFactory.sol
    │
    ├── libraries
    │   ├── ActionsBuilder.sol
    │   ├── DynamicArray.sol
    │   ├── ParamsBuilder.sol
    │   ├── StrategyPlanner.sol
    │   ├── TickCalculations.sol
    │   ├── TokenDistribution.sol
    │   ├── TokenPricing.sol
    │
    ├── periphery
    │   ├── BuybackAndBurnPositionRecipient.sol
    │   ├── PositionFeesForwarder.sol
    │   ├── TimelockedPositionRecipient.sol
    │   │
    │   └── hooks
    │       └── SelfInitializerHook.sol
    │
    ├── strategies
    │   ├── MerkleClaim.sol
    │   │
    │   └── lbp
    │       ├── AdvancedLBPStrategy.sol
    │       ├── FullRangeLBPStrategy.sol
    │       ├── GovernedLBPStrategy.sol
    │       ├── LBPStrategyBase.sol
    │       └── VirtualGovernedLBPStrategy.sol
    │
    └── types
        ├── Distribution.sol
        ├── MigrationData.sol
        ├── MigratorParameters.sol
        └── PositionTypes.sol
```

System Overview

Liquidity Launcher

The liquidity launcher bootstraps liquidity by automatically deploying a Uniswap v4 pool and adding liquidity for a new token paired with a base currency at a fairly determined price. After a continuous clearing auction, a Uniswap v4 pool is launched with the initial tick at the final clearing price of the auction. Both a full-range and single-sided liquidity position may be added to fully utilize the available funds.

Tokens can be launched and distributed through the `LiquidityLauncher` contract. Liquidity will only be added for tokens where the auction was successful. Given a successful auction, after a configured block delay, anyone can call the `migrate` function to initialize the Uniswap pool at the discovered price.

Changes to the Liquidity Launcher

The newly introduced contracts lock liquidity into the contract until a timelock period has passed. After the timelock, liquidity can be removed from the liquidity launcher by the Operator as NFT permissions are granted to the operator. For the `BuybackAndBurnPositionRecipient` contract, any user can call `collectFees` to burn a set amount of tokens in exchange for the currency portion of the fees collected. The remainder of the fees are sent to a burn address. Meanwhile, the `PositionFeesForwarder` contract forwards fees to the fee recipient.

Several strategy contracts have been introduced that are deployed via the `StrategyFactory` using CREATE2. The `FullRangeLBPStrategy` contract deploys a full-range position, The `AdvancedLBPStrategy` creates both a full-range position and a single-sided position to achieve optimal capital efficiency. The `DynamicArray` library was also introduced, which provides a gas-efficient way to build dynamic byte arrays in Solidity by pre-allocating a fixed-size array in memory and tracking the actual length in transient storage.

Security Model and Privileged Roles

Throughout the in-scope codebase, the following assumptions and privileged roles were identified:

- The liquidity launcher is not compatible with fee on transfer tokens or rebasing tokens.
- Token creation and distribution should be done atomically to prevent attackers from taking the tokens which are left inside the liquidity launcher contract.
- The `operator` can freely remove liquidity from the position recipient contracts after the timelock period has passed. Previously, there was no timelock and the operator had immediate control over the liquidity position.

Medium Severity

M-01 `collectFees` Can Be Maliciously Front-Run

When calling the `collectFees` function of the `BuybackAndBurnPositionRecipient` contract, the caller loses `minTokenBurnAmount` of tokens in exchange for the currency fees currently accrued in the liquidity position. However, this is only profitable for the caller if the value of the collected currency exceeds the value of the tokens they burn. If a user sends a `collectFees` transaction expecting a reward, a malicious frontrunner could submit the same transaction before them, causing the user to still burn tokens without receiving any fees. This does not cause the front-runner to lose anything as they get the currency-for-token exchange rate that the original user was trying to receive.

Consider adding front-running protection in the form of a "minimum fees collected" parameter. This is so that fee collectors can ensure that they always receive the expected amount of currency in exchange for the burned tokens.

Update: *Resolved in pull request #107 at commit `68163f9`.*

Low Severity

L-01 Invalid Position Creation Can Be Attempted When The Current Tick Is Too Close to `MIN` or `MAX` Tick

The `getRightSideBounds` function contains a check which returns null values if the current tick is too close to the max tick. However, this check is not strict enough as the function will still attempt to create positions with invalid bounds when the current tick is too high. It only restricts the lower tick to be lower than the maximum tick, but it may attempt to create a position where the lower tick and upper tick are both the maximum tick.

Uniswap v4's maximum tick is `887272`. With this in mind, consider a scenario where the tick spacing is `100` and the current tick is `887170`:

- `TickMath.MAX_TICK - initialTick` is `102`, which is greater than the tick spacing, passing the check.
- The `lowerTick`, will be rounded up from the current tick to the next viable tick spacing which would be `887270`.
- The `upperTick` will be the `maxUsableTick`, which is also `887270`. This is the same as the lower tick, which is an invalid liquidity position as the ticks must be at least one `tickSpacing` apart.

This issue also applies to the `getLeftSideBounds` function where the acceptable tick can be too close to the minimum tick boundary.

Consider making the difference between the current and `MAX` tick to be at least 2 tick spacings.

Update: *Resolved in PR 116 at commit 2253605.*

L-02 `getAddress` Can Return CREATE2 Addresses That Can Never Be Deployed

The `_validateParamsAndReturnDeployedBytecode` function of the `MerkleClaim` contract only ABI-encodes constructor arguments while performing no validation. This contradicts the comment which states that the function “MUST revert if the given params are invalid”. The `getAddress` function of `StrategyFactory` uses the result of `_validateParamsAndReturnDeployedBytecode` and can therefore return addresses for parameters will fail at deployment. A similar mismatch exists in `FullRangeLBPStrategyFactory`, which validates only `totalSupply` but does not enforce `MigratorParameters` invariants that are enforced during `LBPStrategyBase` construction via `_validateMigratorParams`.

Consider changing the `_validateParamsAndReturnDeployedBytecode` function to revert upon incorrect parameters, or documenting the fact that `getAddress` can return invalid addresses.

Update: *Resolved in PR 117 at commit 44d6a93.*

L-03 Missing Docstrings or Incomplete Docstrings

Throughout the codebase, multiple instances of missing docstrings were identified:

- In `ITimelockedPositionRecipient.sol`, the `ITimelockedPositionRecipient` interface
- In `ITimelockedPositionRecipient.sol`, the `timelockBlockNumber` function
- In `ITimelockedPositionRecipient.sol`, the `operator` function
- In `ITimelockedPositionRecipient.sol`, the `positionManager` function
- In `AdvancedLBPStrategy.sol`, the `createOneSidedTokenPosition` state variable
- In `AdvancedLBPStrategy.sol`, the `createOneSidedCurrencyPosition` state variable
- In the `getHookPermissions` function of `GovernedLBPStrategy.sol`, the return value is not documented.

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

***Update:** Partially Resolved in PR #115 at commit 787ae39. The team stated:*

> *Partially resolved, will not fix others*

L-04 Interpretation of Block Number Varies Across Networks

The `block.number` value carries different implications across various layer 2 (L2) networks. For example, on the Optimism network, `block.number` represents the L2 block number, whereas, on Arbitrum, it corresponds to the L1 block number.

Throughout the codebase, multiple instances of `block.number` being used were identified:

- The `block.number` expression in `TimelockedPositionRecipient.sol`.
- The multiple `block.number` expressions in `LBPStrategyBase.sol`.

Consider using the `Blocknumberish` library for the liquidity launcher.

***Update:** Resolved in PR #103 at commit d27c6a8.*

Notes & Additional Information

N-01 Missing Zero-Address Check for Underlying Token

The constructor of the `VirtualGovernedLBPStrategy` contract retrieves the underlying by calling `IVirtualERC20(_token).UNDERLYING_TOKEN_ADDRESS` and stores it as an immutable value. The `getPoolToken` function returns this value, which determines pool currency selection during migration. However, the constructor does not validate the returned address.

As such, if the underlying token is `address(0)`, Uniswap v4 interprets this as native ETH. Since the strategy does not support native ETH as a pool token, the `migrate` function reverts when attempting to settle without providing ETH value. If the underlying equals the raised currency, pool initialization reverts with `CurrenciesOutOfOrderOrEqual`. Since the underlying token is immutable, a misconfigured deployment is permanently non-functional.

Consider reverting if the underlying token is set to `address(0)`.

Update: *Resolved at commit `2608da3`.*

N-02 Unused Error

The `InvalidActionsLength` error in `ActionsBuilder.sol` is unused.

To improve the overall clarity, intentionality, and readability of the codebase, consider either using or removing any currently unused errors.

Update: *Resolved in pull request #115 at commit a1e54a1.*

N-03 Unnecessary Cast

Within the `MerkleClaim` contract, the `address(_token)` cast is unnecessary.

To improve the overall clarity and intent of the codebase, consider removing any unnecessary casts.

Update: *Resolved in pull request #115 at commit 885864c.*

N-04 Possible Duplicate Event Emission

When a setter function does not check if the value being set is different from the existing one, it becomes possible to set the same value repeatedly, creating a potential for event spamming. Repeated emission of identical events can also confuse off-chain clients.

- Within `MerkleClaimHelpers.sol`, the `_transferOwnership` sets the `_owner` and emits an event without checking if the value has changed.
- Within `GovernedLBPStrategy.sol`, the `approveMigration` sets the `isMigrationApproved` variable and emits duplicate events after the initial occurrence.

Consider adding a check that reverts the transaction if the value being set is identical to the existing one.

Update: *Acknowledged, not resolved. The team stated:*

> *Won't resolve*

N-05 Misleading or Incorrect Comments

Throughout the codebase, multiple instances of misleading or incorrect comments were identified:

- In line-61 of `GovernedLBPStrategy` contract, the comments mention the "before remove liquidity" hook which is never implemented. Consider updating the comment to be consistent with the codebase.
- In line-25 of `ILiquidityLauncher` interface, the comments for the `createToken` function mention distributing tokens, which can mislead developers. Consider adding this explanation separately or at the start of the contract with more details.

Consider addressing the above-listed instances of misleading or incorrect comments to improve the clarity and maintainability of the codebase.

Update: *Resolved in pull request #115 at commit 1b4e2b8.*

Conclusion

The Liqudidity Launcher deploys liquidity pools and distributes new tokens after a continuous clearing auction. The audited changes allow liquidity to be locked for a specified period with fees either being burned or sent to a fee recipient.

One medium-severity issue as well as some low- and note-severity issues were identified. The medium-severity issue addresses potential front-running that could occur during fee collection. Meanwhile, the low- and note-severity issues mainly focus on small code improvements, additional checks, and documentation inaccuracies. The Uniswap Labs team is appreciated for providing a detailed walkthrough of the codebase and promptly answering any questions the audit team had about the codebase.