Report

v. 1.0

Customer

Uniswap

Smart Contract Audit

Token Launcher

14th November 2025

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	14.11.25	A. Zveryanskaya	Initial Draft
0.2	14.11.25	A. Zveryanskaya	Minor revision
1.0	14.11.25	A. Zveryanskaya	Release

2 Summary

This document presents the results of a smart contract audit performed by ABDK Consulting for Uniswap on the `Token Launcher` system. The audit was conducted from 20th October to 14th November 2025 by Mikhail Vladimirov and Dmitry Khovratovich. The goal of the audit was to assess the security, correctness, and code quality of the smart contract system, with particular focus on token creation mechanisms, distribution strategies, and integration with Uniswap v4 protocol.

Our conclusion is that the system demonstrates a mature implementation with solid code quality. The codebase implements token factories, distribution strategies including Liquidity Bootstrapping Pool mechanisms and Merkle-based claims, as well as libraries for tick calculations and token pricing within the Uniswap v4 ecosystem. The system integrates with Permit2 for enhanced token approvals and provides flexible distribution mechanisms for newly created tokens.

Overall, the system exhibits a well-structured architecture with comprehensive functionality for token launching and distribution. The absence of critical or major vulnerabilities indicates that the smart contract system has been developed with attention to security best practices.

It is important to note that a security audit is not a guarantee of absolute security but rather a snapshot of the system's security posture at a specific point in time. While we strive to uncover all potential issues, the evolving nature of blockchain technology and DeFi means new vulnerabilities can emerge.

3 System overview

This section provides a high-level overview of the system designed to facilitate the creation and distribution of tokens within the Uniswap v4 ecosystem. The system enables users to launch new tokens, configure their initial distribution mechanisms, and establish liquidity positions on decentralized exchanges. It serves as a comprehensive platform for token deployment, combining token factory capabilities with sophisticated distribution strategies and automated liquidity management.

We were asked to review:

- Original Code
- Code with Fixes

TokenLauncher files:

/			
	TokenLauncher.sol	Permit2Forwarder.sol	Multicall.sol
utils/			
	HookBasic.sol		
types/			
	PositionTypes.sol	MigratorParams.sol	MigrationData.sol
	Distribution.sol		
libraries/			
	TokenPricing.sol	TickCalculations.sol	StrategyPlanner.sol
	ParamsBuilder.sol	ActionsBuilder.sol	
interfaces/			
	ITokenLauncher.sol	IPermit2Forwarder.sol	IMulticall.sol
	ILBPStrategyBasic.sol	IDistributionStrategy.sol	IDistributionContract.sol
token-factories/uerc20-factory/factories/			
	USUPERC20Factory.sol	UERC20Factory.sol	

distributionStrategies/			
	MerkleClaimFactory.sol	LBPStrategyBasic Factory.sol	MerkleClaim.sol
	LBPStrategyBasic.sol		
token-factories/uerc20-factory/tokens/			
	USUPERC20.sol	UERC20.sol	BaseUERC20.sol
token-factories/uerc20-factory/libraries/			
	UERC20Metadata Library.sol		
token-factories/uerc20-factory/interfaces/			
	IUSUPERC20Factory.sol	IUSUPERC20Factory.sol	ITokenFactory.sol

At the core of the system is the `TokenLauncher` contract, which orchestrates the entire token creation and distribution workflow. This central component coordinates interactions between multiple subsystems, managing the lifecycle from initial token deployment through distribution execution. The launcher integrates with external permission management systems to enable gasless approvals, providing a seamless user experience for token operations. It implements a multicall pattern that allows batching multiple operations into a single transaction, improving efficiency and reducing transaction costs.

The token creation functionality is handled by specialized factory contracts, specifically `UERC20Factory` and `USUPERC20Factory`. These factories enable deterministic deployment of token contracts with customizable parameters including name, symbol, total supply, and extended metadata. The token implementations extend a base contract that inherits from an external token library, providing standard functionality for transfers, approvals, and balance tracking. The metadata library supports rich token information including descriptions, website URLs, and image references, encoded in a structured format for compatibility with decentralized applications and wallets.

Distribution strategies form a critical component of the system architecture. The `LBPStrategyBasic` implements a Liquidity Bootstrapping Pool mechanism that enables price discovery through a controlled token sale process. This strategy manages the transition from an initial auction phase to a stable liquidity pool on Uniswap v4. The strategy coordinates with an external auction factory to create auction contracts, allocates a specified percentage of the token supply to the auction, and prepares the remaining tokens for liquidity provision. Upon completion, it facilitates migration to a Uniswap v4 pool by calculating appropriate tick ranges, determining liquidity amounts, and executing the necessary actions to establish trading pairs.

An alternative distribution approach is provided through `MerkleClaimFactory`, which creates contracts that allow users to claim tokens based on cryptographic proofs. This mechanism leverages merkle tree data structures to efficiently verify eligibility for token allocations without requiring on-chain storage of all recipient addresses. The claim contracts incorporate time constraints to define distribution windows and provide administrative functions for managing unclaimed tokens after the distribution period ends.

The system includes several computational libraries that handle complex mathematical operations required for integration with Uniswap v4. The `TickCalculations` library

performs tick arithmetic, converting between price points and tick indices, calculating maximum liquidity per tick based on spacing parameters, and ensuring that position boundaries align with valid tick multiples. The `TokenPricing` library handles price conversions between different representations, managing the fixed-point arithmetic required for accurate price quotations across varying token decimals and pool configurations.

Liquidity position management is facilitated through the `StrategyPlanner` library, which determines optimal position parameters based on available token amounts and desired distribution patterns. The planner supports multiple position types including full range positions that provide liquidity across the entire price spectrum, and one-sided positions that concentrate capital at specific price points. When a requested position exceeds the maximum liquidity constraints for a given tick range, the system implements fallback logic to ensure transaction success by adjusting to alternative position configurations.

The `ActionsBuilder` and `ParamsBuilder` libraries work in tandem to construct the action sequences and parameter sets required for interacting with the Uniswap v4 pool manager. These libraries encode operations such as settling token transfers, minting liquidity positions based on token deltas, and clearing or withdrawing excess amounts. The structured approach ensures that complex multi-step operations are executed atomically and correctly.

Integration with external systems is a fundamental aspect of the architecture. The system interfaces with the Uniswap v4 pool manager for all liquidity operations, utilizing the protocol's hook mechanism to enable custom logic during pool interactions. The `HookBasic` utility provides a foundation for implementing hook callbacks that can execute before or after swaps, liquidity modifications, and other pool events. Additionally, the system integrates with a permission management protocol to enable token approvals without requiring separate approval transactions, streamlining the user interaction flow. The use of external token standards and merkle distribution libraries demonstrates the system's commitment to leveraging proven, audited components where appropriate while focusing development efforts on the unique token launching and distribution logic.

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check whether the code actually does what it is supposed to do, whether the algorithms are optimal and correct, and whether proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Recommendations** cover code style, best practices and general improvements.

5 Our findings

We provided the client with some recommendations.

Moderate	Info	Fixed
	10	2

6 Moderate Issues

CVF-1 INFO

- **Category** Unclear behavior
- **Source** StrategyPlanner.sol

Description Here we silently fallback to the full range position plan, which makes contract behavior less predictable.

Recommendation Revert in case one side position cannot be minted.

Client Comment *We think this is more secure since migration won't be totally bricked then - at least a full range position can still be created.*

```
91 if (baseParams.liquidity + newLiquidity > baseParams.poolTickSpacing
   ↪ .tickSpacingToMaxLiquidityPerTick()) {
    return (existingActions, ParamsBuilder.truncateParams(
       ↪ existingParams));
```

CVF-2 INFO

- **Category** Unclear behavior
- **Source** MerkleClaimFactory.sol

Description The requirements for the salt value are not listed.

Recommendation Provide explicit requirements such as randomness/entropy, uniqueness, length, etc.

Client Comment *Won't fix because not using this at deployment time.*

```
17 /// @param salt The salt for deterministic deployment
```

```
37 /// @param salt The salt for deterministic deployment
```

CVF-3 INFO

- **Category** Procedural
- **Source** BaseUERC20.sol

Recommendation While Solidity doesn't support immutable variables of type string, it is possible to store short strings as immutable "bytes32" variables. See the following gist for details: https://gist.github.com/3sGgpQ8H/567354534170905e047b299286697e19

Client Comment *Not changing anything in factories folder.*

```
26 string internal _name;
   string internal _symbol;
```

CVF-4 INFO

- **Category** Suboptimal
- **Source** UERC20MetadataLibrary.sol

Description The data is copied several times.
Recommendation Refactor the code to produce JSON in one pass without copying.
Client Comment *Not changing anything in factories folder.*

```
30 bytes memory json = abi.encodePacked("{");

34     json = abi.encodePacked(json, '"description":"', metadata.
          ↪ description.escapeJSON(), '"');

39     json = abi.encodePacked(json, '"website":"', metadata.website.
          ↪ escapeJSON(), '"');

44     json = abi.encodePacked(json, '"image":"', metadata.image.
          ↪ escapeJSON(), '"');

47 return abi.encodePacked(json, "}");
```

CVF-5 INFO

- **Category** Suboptimal
- **Source** TickCalculations.sol

Description This could be optimized, taking into account that MIN_TICK = -MAX_TICK. So, in case MAX_TICK % tickSpacing is zero, numTicks = MAX_TICK / tickSpacing * 2 + 1. Otherwise, numTicks = MAX_TICK / tockSpacing * 2 + 2.
Recommendation Optimize this calculation.
Client Comment *Taken right from v4 so won't fix.*

```
19 let minTick := sub(sdiv(MIN_TICK, tickSpacing), slt(smod(MIN_TICK,
      ↪ tickSpacing), 0))
20 let maxTick := sdiv(MAX_TICK, tickSpacing)
   let numTicks := add(sub(maxTick, minTick), 1)
```

CVF-6 FIXED

- **Category** Suboptimal
- **Source** TickCalculations.sol

Recommendation This calculation seems subotimal. It could be optimized like this: unchecked { int24 reminder = tick % tickSpacing; return reminder >= 0 ? tick - reminder : tick - reminder - tickSpace; }

```
31 int24 compressed = tick / tickSpacing;
   if (tick < 0 && tick % tickSpacing != 0) compressed--;
   return compressed * tickSpacing;
```

CVF-7 FIXED

- **Category** Suboptimal
- **Source** TickCalculations.sol

Recommendation This calculation seems suboptimal. It could be optimized like this: unchecked { int24 reminder = tick % tickSpacing; return reminder >= 0 ? tick + tickSpacing - reminder : tick - reminder; }

```
41 int24 compressed = tick / tickSpacing;
   if (tick % tickSpacing != 0) {
       if (tick >= 0) {
           compressed++; // For positive ticks, add 1 to round up
       }
       // For negative ticks, integer division already rounds towards
          ↪ zero (up)
   } else {
       compressed++; // If already a multiple, go to next one
   }
50 return compressed * tickSpacing;
```

CVF-8 INFO

- **Category** Suboptimal
- **Source** TokenPricing.sol

Recommendation The precision could be improved by using larger numerator when inverting the price, such as 2^255, or even 2^256-1. Something like this: if (currencyIs-Currency0) { price = ~uint (0) / price; require (price » 224 == 0); priceX192 = price « 32; } else { require (price » 160 == 0); priceX192 = price « 96; }

```
35     price = (1 << (FixedPoint96.RESOLUTION * 2)) / price;
```

```
44 priceX192 = price << FixedPoint96.RESOLUTION;
```

CVF-9 INFO

- **Category** Procedural
- **Source** IMulticall.sol

Recommendation In case one of the calls reverts, this function should revert either with an error message, that includes the index of the failed call as well as the error message received from it. This error should be declared in this interface.

Client Comment *Taken right from v4 so won't fix.*

```
10 function multicall(bytes[] calldata data) external returns (bytes[]
      ↪ memory results);
```

CVF-10 INFO

- **Category** Procedural
- **Source** IPermit2Forwarder.sol

Description Returning an error message from inner call as a normal returned value is a bad practice.

Recommendation Revert in case the inner call reverted, and wrap the original error message into a higher-level error.

Client Comment *Taken right from v4 so won't fix.*

```
13 /// @return err the error returned by a reverting permit call, empty
      ↪  if successful
```

CVF-11 INFO

- **Category** Suboptimal
- **Source** Multicall.sol

Description Forwarding the original error message as is makes it harder to know which particular operation failed.

Recommendation Wrap the original error message into a higher-level error and include the index of the failed operation into this error as an additional parameter.

Client Comment *Taken right from v4 so won't fix.*

```
16 // bubble up the revert reason
assembly {
    revert(add(result, 0x20), mload(result))
```

CVF-12 INFO

- **Category** Procedural
- **Source** Permit2Forwarder.sol

Description Returning onhte the revert reason makes it impossible to distinguish successful “permit2” call from a failed one that returned no error.

Recommendation Return the success flag along with the error.

Client Comment *Taken right from v4 so won’t fix.*

```
23 catch (bytes memory reason) {
       err = reason;
```

7 Recommendations

CVF-13 FIXED

- **Category** Procedural
- **Source** HookBasic.sol

Description Specifying a particular compiler version makes it harder migrating to newer versions.

Recommendation Specify as "^0.8.0" or as "^0.8.26" is there is something special regarding this particular version. See also: UERC20.sol, UERC20Factory.sol, USUPERC20Factory.sol, IPermit2Forwarder.sol, ITokenLauncher.sol, LBPStrategyBasicFactory.sol, MerkleClaimFactory.sol, MerkleClaim.sol, LBPStrategyBasic.sol, TickStorage.sol, AuctionStepStorage.sol, BidStorage.sol, PermitSingleForwarder.sol, TokenCurrencyStorage.sol, AuctionFactory.sol, Auction.sol, CheckpointStorage.sol.

```
2 pragma solidity 0.8.26;
```

CVF-14 INFO

- **Category** Procedural
- **Source** HookBasic.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
18 constructor(IPoolManager _poolManager) BaseHook(_poolManager) {}
```

CVF-15 FIXED

- **Category** Procedural
- **Source** MigratorParams.sol

Description The structure name is similar but not exactly equal to the file name. This make harder navigating through code.

Recommendation Make the structure name and the file name equal to each other.

```
6 struct MigratorParameters {
```

CVF-16 INFO

- **Category** Bad datatype
- **Source** MigratorParams.sol

Recommendation The type for this field should be “IERC20”.

```
8 address currency; // the currency that the token will be paired with
    ↪  in the v4 pool (currency that the auction raised funds in)
```

CVF-17 FIXED

- **Category** Documentation
- **Source** MigratorParams.sol

Description The number format for this field is unclear.

Recommendation Explain in a documentation comment.

```
11 uint24 tokenSplitToAuction; // the percentage of the total supply of
    ↪  the token that will be sent to the auction
```

CVF-18 INFO

- **Category** Bad datatype
- **Source** MigratorParams.sol

Recommendation The type for this field should be more specific.

```
12 address auctionFactory; // the Auction factory that will be used to
    ↪ create the auction
```

CVF-19 INFO

- **Category** Bad naming
- **Source** PositionTypes.sol

Description The names of these structures are not directly related to the name of the file they are declared in. This makes it harder navigating through code.

Recommendation Rename to file to better describe its content.

```
7 struct BasePositionParams {
```

```
19 struct FullRangeParams {
```

```
25 struct OneSidedParams {
```

```
31 struct TickBounds {
```

CVF-20 INFO

- **Category** Bad datatype
- **Source** PositionTypes.sol

Recommendation The type for these fields should be “IERC20”.

```
8 address currency;
  address token;
```

CVF-21 INFO

- **Category** Bad datatype
- **Source** Distribution.sol

Recommendation The type for this field should be more specific.

```
7 address strategy;
```

CVF-22 INFO

- **Category** Procedural
- **Source** UERC20.sol

Description This version requirement is inconsistent with other files in the same code base.

Recommendation Use consistent version requirements across code base. See also: USUPERC20.sol, UERC20Factory.sol, USUPERC20Factory.sol, IPermit2Forwarder.sol, ITokenLauncher.sol, LBPStrategyBasicFactory.sol, MerkleClaimFactory.sol, MerkleClaim.sol, LBPStrategyBasic.sol.

Client Comment *Not changing anything in factories folder.*

```
2 pragma solidity 0.8.26;
```

CVF-23 INFO

- **Category** Procedural
- **Source** BaseUERC20.sol

Description We didn’t review this file.

Client Comment *Not changing anything in factories folder.*

```
4 import {ERC20} from "@solady/src/tokens/ERC20.sol";
```

CVF-24 INFO

- **Category** Procedural
- **Source** UERC20MetadataLibrary.sol

Description Declaring a top-level structure in a file named after a library makes it harder navigating through code.

Recommendation Move the structure declaration into the library or move it into a separate file.

Client Comment *Not changing anything in factories folder.*

```
7 struct UERC20Metadata {
```

CVF-25 INFO

- **Category** Bad naming
- **Source** UERC20MetadataLibrary.sol

Description The function name suggests that the output format is JSON while actually it is bae64 encoded JSON which is not the same.

Recommendation Rename to function properly.

Client Comment *Not changing anything in factories folder.*

```
19 /// @notice Generates a base64 encoded JSON string of the token
      ↪ metadata
```

```
22 function toJSON(UERC20Metadata memory metadata) internal pure
      ↪ returns (string memory) {
```

CVF-26 INFO

- **Category** Bad naming
- **Source** ITokenFactory.sol

Recommendation Events are usually named via nouns, such as “Token”.

Client Comment *Not changing anything in factories folder.*

```
8 event TokenCreated(address tokenAddress);
```

CVF-27 INFO

- **Category** Bad datatype
- **Source** ITokenFactory.sol

Recommendation The parameter type should be “IERC20”.
Client Comment *Not changing anything in factories folder.*

```
8 event TokenCreated(address tokenAddress);
```

CVF-28 INFO

- **Category** Bad datatype
- **Source** ITokenFactory.sol

Recommendation The return type should be “IERC20”.
Client Comment *Not changing anything in factories folder.*

```
33 ) external returns (address tokenAddress);
```

CVF-29 INFO

- **Category** Bad datatype
- **Source** IUERC20Factory.sol

Recommendation The return type should be “IERC20”.
Client Comment *Not changing anything in factories folder.*

```
35 ) external view returns (address);
```

CVF-30 INFO

- **Category** Bad datatype
- **Source** IUSUPERC20Factory.sol

Recommendation The return type should be “IERC20”.
Client Comment *Not changing anything in factories folder.*

```
41 ) external view returns (address);
```

CVF-31 INFO

- **Category** Bad datatype
- **Source** UERC20Factory.sol

Recommendation The return type should be “IERC20”.
Client Comment *Not changing anything in factories folder.*

```
23 ) external view returns (address) {
```

```
43 ) external returns (address tokenAddress) {
```

CVF-32 INFO

- **Category** Bad datatype
- **Source** UERC20Factory.sol

Recommendation The return type should be “IERC20”.
Client Comment *Not changing anything in factories folder.*

```
43 ) external returns (address tokenAddress) {
```

CVF-33 INFO

- **Category** Suboptimal
- **Source** UERC20Factory.sol

Description This check is redundant, as it is anyway possible to pass a dead recipient address.
Recommendation Remove this check.
Client Comment *Not changing anything in factories folder.*

```
46 if (recipient == address(0)) {
       revert RecipientCannotBeZeroAddress();
```

CVF-34 INFO

- **Category** Readability
- **Source** UERC20Factory.sol

Description This like looks like a plain assignment, while actually it returns value from the function.
Recommendation Use an explicit return statement.
Client Comment *Not changing anything in factories folder.*

```
69 tokenAddress = address(new UERC20{salt: salt}());
```

CVF-35 INFO

- **Category** Bad datatype
- **Source** USUPERC20Factory.sol

Recommendation The return type should be “IERC20”.

Client Comment *Not changing anything in factories folder.*

```
24 ) external view returns (address) {
```

```
44 ) external returns (address tokenAddress) {
```

CVF-36 INFO

- **Category** Suboptimal
- **Source** USUPERC20Factory.sol

Description This check is redundant, as it is anyway possible to pass a dead recipient address.

Recommendation Remove this check.

Client Comment *Not changing anything in factories folder.*

```
54 if (recipient == address(0)) {
       revert RecipientCannotBeZeroAddress();
```

CVF-37 INFO

- **Category** Suboptimal
- **Source** TickCalculations.sol

Recommendation This assembly block seems redundant, as the same logic could be implemented in pure Solidity.

Client Comment *Taken right from v4 so won't fix*

```
17 assembly ("memory-safe") {
       tickSpacing := signextend(2, tickSpacing)
       let minTick := sub(sdiv(MIN_TICK, tickSpacing), slt(smod(
           ↪ MIN_TICK, tickSpacing), 0))
20     let maxTick := sdiv(MAX_TICK, tickSpacing)
       let numTicks := add(sub(maxTick, minTick), 1)
       result := div(sub(shl(128, 1), 1), numTicks)
   }
```

CVF-38 FIXED

- **Category** Documentation
- **Source** TokenPricing.sol

Description The number format of the parameter is unclear.
Recommendation Explain in a documentation comment.

```
14 error InvalidPrice(uint256 price);
```

CVF-39 FIXED

- **Category** Suboptimal
- **Source** TokenPricing.sol

Recommendation This error could be made more useful by adding certain parameters into it.

```
17 error AmountOverflow();
```

CVF-40 FIXED

- **Category** Procedural
- **Source** ParamsBuilder.sol

Recommendation These imports should be merged together.

```
6 import {FullRangeParams, OneSidedParams} from "../types/
    ↪ PositionTypes.sol";
  import {TickBounds} from "../types/PositionTypes.sol";
```

CVF-41 INFO

- **Category** Procedural
- **Source** ActionsBuilder.sol

Recommendation This could be merged into a single bytes5 value.

```
15 uint8(Actions.SETTLE),
   uint8(Actions.SETTLE),
   uint8(Actions.MINT_POSITION_FROM_DELTAS),
   uint8(Actions.CLEAR_OR_TAKE),
   uint8(Actions.CLEAR_OR_TAKE)
```

CVF-42 INFO

- **Category** Procedural
- **Source** ActionsBuilder.sol

Recommendation This could be merged into a single bytes3 values.

```
31 uint8(Actions.SETTLE),
   uint8(Actions.MINT_POSITION_FROM_DELTAS),
   uint8(Actions.CLEAR_OR_TAKE)
```

CVF-43 INFO

- **Category** Suboptimal
- **Source** StrategyPlanner.sol

Description The same condition is checked twice.

Recommendation Refactor the code to check only once.

```
87 currencyIsCurrency0 == oneSidedParams.inToken ? 0 : oneSidedParams.
      ↪ amount,
   currencyIsCurrency0 == oneSidedParams.inToken ? oneSidedParams.
      ↪ amount : 0
```

CVF-44 INFO

- **Category** Suboptimal
- **Source** StrategyPlanner.sol

Description Here the position liquidity is compared to the maximum liquidity per tick, even though the position can be wider than one tick.

Recommendation Compare to the maximum liquidity of a position with the same width and the current position.

```
91 if (baseParams.liquidity + newLiquidity > baseParams.poolTickSpacing
      ↪ .tickSpacingToMaxLiquidityPerTick()) {
```

CVF-45 INFO

- **Category** Suboptimal
- **Source** StrategyPlanner.sol

Description The same expreion is check twice.
Recommendation Refactor the code to check only once.

```
96 currency0: Currency.wrap(currencyIsCurrency0 ? baseParams.currency :
      ↪  baseParams.token),
   currency1: Currency.wrap(currencyIsCurrency0 ? baseParams.token :
      ↪ baseParams.currency),
```

CVF-46 INFO

- **Category** Bad naming
- **Source** ILBPStrategyBasic.sol

Recommendation Event are usually named via nouns, such as “Migration” or “Auction”.

```
13 event Migrated(PoolKey indexed key, uint160 initialSqrtPriceX96);
```

```
17 event AuctionCreated(address indexed auction);
```

```
25 event TokensSwept(address indexed operator, uint256 amount);
```

```
28 event CurrencySwept(address indexed operator, uint256 amount);
```

CVF-47 INFO

- **Category** Bad datatype
- **Source** ILBPStrategyBasic.sol

Recommendation The parameter type should be more specific.

```
17 event AuctionCreated(address indexed auction);
```

CVF-48 INFO

- **Category** Bad datatype
- **Source** ITokenLauncher.sol

Recommendation The type for the “tokenAddress” parameters shold be “IERC20”.

```
15 event TokenCreated(address indexed tokenAddress);
```

```
21 event TokenDistributed(address indexed tokenAddress, address indexed
      ↪  distributionContract, uint256 amount);
```

CVF-49 INFO

- **Category** Bad naming
- **Source** ITokenLauncher.sol

Recommendation Events are usually named via nouns, such as “Token” or “Distribution”.

```
15 event TokenCreated(address indexed tokenAddress);
```

```
21 event TokenDistributed(address indexed tokenAddress, address indexed
   ↪  distributionContract, uint256 amount);
```

CVF-50 INFO

- **Category** Bad datatype
- **Source** ITokenLauncher.sol

Recommendation The type for this argument should be more specific.

```
37 address factory,
```

CVF-51 INFO

- **Category** Bad datatype
- **Source** ITokenLauncher.sol

Recommendation The return type should be “IERC20”.

```
44 ) external returns (address tokenAddress);
```

CVF-52 INFO

- **Category** Bad datatype
- **Source** ITokenLauncher.sol

Recommendation The type for the “tokenAddress” argument should be “IERC20”.

```
52 function distributeToken(address tokenAddress, Distribution memory
   ↪ distribution, bool payerIsUser, bytes32 salt)
```

CVF-53 INFO

- **Category** Bad datatype
- **Source** LBPStrategyBasicFactory.sol

Recommendation The type for the “token” argument should be “IERC20”.

```
27 function initializeDistribution(address token, uint256 totalSupply,
      ↪ bytes calldata configData, bytes32 salt)
```

```
53 function getLBPAddress(address token, uint256 totalSupply, bytes
      ↪ calldata configData, bytes32 salt, address sender)
```

CVF-54 INFO

- **Category** Bad datatype
- **Source** LBPStrategyBasicFactory.sol

Recommendation The return type should be more specific.

```
56 returns (address)
```

CVF-55 INFO

- **Category** Bad datatype
- **Source** MerkleClaimFactory.sol

Recommendation The type for the “token” argument should be “IERC20”.

```
19 function initializeDistribution(address token, uint256 totalSupply,
      ↪ bytes calldata configData, bytes32 salt)
```

```
40 function getMerkleClaimAddress(address token, bytes calldata
      ↪ configData, bytes32 salt, address sender)
```

CVF-56 INFO

- **Category** Procedural
- **Source** MerkleClaim.sol

Description We didn’t review this file.

```
4 import {MerkleDistributorWithDeadline} from "merkle-distributor/
     ↪ contracts/MerkleDistributorWithDeadline.sol";
```

CVF-57 INFO

- **Category** Bad datatype
- **Source** MerkleClaim.sol

Recommendation The type for the "_token" argument should be "IERC20".

```
11 constructor(address _token, bytes32 _merkleRoot, address _owner,
   ↪ uint256 _endTime)
```

CVF-58 INFO

- **Category** Suboptimal
- **Source** MerkleClaim.sol

Description This check is redundant, as it is anyway possible to pass a dead token.
Recommendation Remove this check.

```
14 if (_token == address(0)) {
```

CVF-59 INFO

- **Category** Procedural
- **Source** MerkleClaim.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
22 function onTokensReceived() external {}
```

CVF-60 INFO

- **Category** Suboptimal
- **Source** LBPStrategyBasic.sol

Description This denominator is unconventional.
Recommendation Consider using more conventional denominator, such as 1e8 or 1e18.

```
41 /// @notice The maximum percentage of the supply for distribution
   ↪ that can be sent to the auction, expressed in mps (1e7 = 100%)
```

CVF-61 INFO

- **Category** Bad datatype
- **Source** LBPStrategyBasic.sol

Recommendation The type for these variables should be “IERC20”.

```
45 address public immutable token;
```

```
47 address public immutable currency;
```

CVF-62 FIXED

- **Category** Documentation
- **Source** LBPStrategyBasic.sol

Description The number format for this variable is unclear.
Recommendation Explain in the documentation comment.

```
49 /// @notice The LP fee that the v4 pool will use
50 uint24 public immutable poolLPFee;
```

CVF-63 INFO

- **Category** Bad datatype
- **Source** LBPStrategyBasic.sol

Recommendation The type for this argument should be “IERC20”.

```
80 address _token,
```

CVF-64 FIXED

- **Category** Suboptimal
- **Source** LBPStrategyBasic.sol

Description This “mulDiv” call is performed twice: once here, and another time inside the “_validateMigratorParams” call above.
Recommendation Refactor the code to avoid calculating the same number twice.

```
98 _totalSupply - FullMath.mulDiv(_totalSupply, _migratorParams.
   ↪ tokenSplitToAuction, MAX_TOKEN_SPLIT);
```

CVF-65 INFO

- **Category** Suboptimal
- **Source** LBPStrategyBasic.sol

Description The comparison to zero address is redundant, as it is anyway possible to pass a dead token address.

Recommendation Remove redundant check.

```
189 // token validation (cannot be zero address or the same as the
    ↪ currency)
190 else if (_token == address(0) || _token == migratorParams.currency)
    ↪ {
```

CVF-66 INFO

- **Category** Procedural
- **Source** LBPStrategyBasic.sol

Description The current currency balance is queried twice.

Recommendation Refactor the code to query it once and reuse.

```
251 if (Currency.wrap(currency).balanceOf(address(this)) <
    ↪ currencyAmount) {
     revert InsufficientCurrency(currencyAmount, Currency.wrap(
        ↪ currency).balanceOf(address(this)));
```

CVF-67 INFO

- **Category** Procedural
- **Source** LBPStrategyBasic.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
452 receive() external payable {}
```

CVF-68 FIXED

- **Category** Procedural
- **Source** Multicall.sol

Description Consider specifying as “^0.8.0” unless there is something special regarding this particular version.

```
2 pragma solidity ^0.8.23;
```

CVF-69 INFO

- **Category** Procedural
- **Source** TokenLauncher.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
20 constructor(IAllowanceTransfer _permit2) Permit2Forwarder(_permit2)
      ↪ {}
```

CVF-70 INFO

- **Category** Bad datatype
- **Source** TokenLauncher.sol

Recommendation The type for this argument should be “ITokenFactory”.

```
24 address factory,
```

CVF-71 INFO

- **Category** Bad datatype
- **Source** TokenLauncher.sol

Recommendation The return type should be “IERC20”.

```
31 ) external override returns (address tokenAddress) {
```

CVF-72 INFO

- **Category** Suboptimal
- **Source** TokenLauncher.sol

Description This check is redundant, as it is anyway possible to pass a dead recipient address.

Recommendation Remove this check.

```
32 if (recipient == address(0)) {
       revert RecipientCannotBeZeroAddress();
```

CVF-73 INFO

- **Category** Bad datatype
- **Source** TokenLauncher.sol

Recommendation The type for the “token” argument should be “IERC20”.

```
43 function distributeToken(address token, Distribution calldata
   ↪ distribution, bool payerIsUser, bytes32 salt)
```

```
74 function _transferToken(address token, address from, address to,
   ↪ uint256 amount) internal {
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 300 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting