BLOCKSEC

Security Audit

Report for launchpad-arc

Date: October 10, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction **1**

1.1 About the Audit Target . . . 1
1.2 Disclaimer . . . 2
1.3 Procedure of Auditing . . . 2
1.3.1 Security Issues . . . 2
1.3.2 Additional Recommendation . . . 3
1.4 Security Model . . . 3

Chapter 2 Findings **5**

2.1 Security Issue . . . 6
2.1.1 Unexpected graduation behaviors under extreme configurations . . . 6
2.1.2 Potential overwrite of a valid recipient . . . 7
2.1.3 Inconsistent fee calculation between exact-input and exact-output swaps 8
2.2 Recommendation . . . 10
2.2.1 Add validations in function `validateCombination()` . . . 10
2.2.2 Align comments with implementation . . . 11
2.2.3 Validate liquidity in function `_validateSeed()` . . . 12
2.2.4 Remove redundant computation in the function `buyExactIn()` . . . 13
2.2.5 Revise misleading variable names . . . 14
2.2.6 Allow disabled partners to claim previously accrued fees . . . 14
2.3 Note . . . 15
2.3.1 Potential centralization risk . . . 15
2.3.2 Ensure a correct slippage bound . . . 15
2.3.3 Different fee rates across launch phases . . . 16
2.3.4 Careful configuration of the partner share cap . . . 17
2.3.5 Fee sweep requires settlement of all partners . . . 17
2.3.6 Ensure off-chain validation of launch configurations . . . 18
2.3.7 Refactoring and new features are outside the audit scope . . . 19

Report Manifest

Item	Description
Client	Peach
Target	launchpad-arc

Version History

Version	Date	Description
1.0	October 10, 2026	First release

Signature



Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-10-14 04:39:45+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About the Audit Target

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of launchpad-arc of Peach.

The launchpad-arc is a non-upgradeable token launch protocol deployed on Arc, an EVM chain. Each launch deploys one fixed-supply launch token and one dedicated bonding curve via CREATE2, with the entire supply minted to that curve. Trading runs on a constant product curve with a phantom quote reserve. Once the sellable supply is exhausted, the launch graduates permissionlessly into a Uniswap V4 pool, with the full-range position locked permanently. Fees are split among the platform, creators, and partners. Launches may optionally enable buyback, routing a portion of the creator share into launch tokens that vest in contract BuybackVault over five years. One factory serves multiple approved quote tokens and captures each launch's economics at creation. Therefore, later governance changes affect only future launches.

Note this audit only focuses on the codes located in the following directories/files.

- contracts/emergency-distributor/src
- contracts/orphaned-assets-vault/src
- contracts/production/src

Other files are not within the scope of the audit. Additionally, all dependencies of the Target are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
launchpad-arc	`Version 1`	`93ad8042fb317bec54bd75718093a08a98a2fb3b`
	`Version 2`	`443ca4fb29f2285b578d5251aed63a6fd317a7f2`

[1] https://github.com/peachbuild/launchpad-arc

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the Target, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of the Target.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan the Target with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of the Target and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow
* Exception handling
* Data handling and flow
* Events operation
* Error-prone randomness

- Oracle security
- Business logic correctness
- Semantic and functional consistency
- Emergency mechanism
- Economic and incentive impact

1.3.2 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we found **three** potential security issues. Besides, we have **six** recommendations and **seven** notes.

- Medium Risk: 1
- Low Risk: 2
- Recommendation: 6
- Note: 7

ID	Severity	Description	Category	Status
1	Medium	Unexpected graduation behaviors under extreme configurations	Security Issue	Fixed
2	Low	Potential overwrite of a valid recipient	Security Issue	Fixed
3	Low	Inconsistent fee calculation between exact-input and exact-output swaps	Security Issue	Fixed
4	-	Add validations in function `validateCombination()`	Recommendation	Confirmed
5	-	Align comments with implementation	Recommendation	Fixed
6	-	Validate liquidity in function `_validateSeed()`	Recommendation	Fixed
7	-	Remove redundant computation in the function `buyExactIn()`	Recommendation	Confirmed
8	-	Revise misleading variable names	Recommendation	Fixed
9	-	Allow disabled partners to claim previously accrued fees	Recommendation	Fixed
10	-	Potential centralization risk	Note	-
11	-	Ensure a correct slippage bound	Note	-
12	-	Different fee rates across launch phases	Note	-
13	-	Careful configuration of the partner share cap	Note	-
14	-	Fee sweep requires settlement of all partners	Note	-
15	-	Ensure off-chain validation of launch configurations	Note	-
16	-	Refactoring and new features are outside the audit scope	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Unexpected graduation behaviors under extreme configurations

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description When creating a launch, the contract `LaunchFactory` does not reject extreme configurations. Such configurations can result in extreme pool states at graduation, leading to unexpected behaviors.

First, extreme price can prevent the pool from graduating or entering the emergency-exit flow. In the contract `GraduationExecutor`, the function `_validateSeed()` validates whether a graduation seed can initialize a Uniswap V4 pool. It invokes the function `_sqrtPriceX96()`, which computes `FullMath.mulDiv(amount1, Q192, amount0)`. When `amount1/amount0 >= 2**64`, the result exceeds the maximum value of `uint256`, causing the invocation to revert. Both functions `graduate()` and `forceSweptGraduation()` depend on the boolean result of function `_validateSeed()` before moving funds from the contract `BondingCurve`. Consequently, the revert blocks both functions, leaving funds stuck.

Second, extreme configurations can amplify the impact of precision loss, leading to price inconsistency between the bonding curve and the Uniswap V4 pool. Function `_validateSeed()` computes `poolTokenAmount` via `FullMath.mulDiv(tokenAmount, quoteAmount, quoteAmount + phantomQuote)`, which rounds down. Function `_sqrtPriceX96()` uses this rounded amount and `quoteAmount` to determine the initial Uniswap V4 pool price. When `tokenAmount` is small, this rounding error can be significant, creating an arbitrage opportunity after the pool opens.

```
223   function _validateSeed(GraduationSeed calldata seed) private view returns (bool) {
224       if (
225           seed.token == address(0) || seed.token == seed.quoteToken || seed.tokenAmount == 0
226               || seed.quoteAmount == 0 || seed.phantomQuote == 0
227               || seed.protocolFeeRecipient == address(0) || seed.tickSpacing <= 0
228               || seed.tickSpacing > TickMath.MAX_TICK_SPACING || seed.token.code.length == 0
229               || (seed.quoteToken != address(0) && seed.quoteToken.code.length == 0)
230       ) return false;
231       uint256 poolTokenAmount = FullMath.mulDiv(
232           seed.tokenAmount, seed.quoteAmount, seed.quoteAmount + seed.phantomQuote
233       );
234       if (poolTokenAmount == 0 || poolTokenAmount >= seed.tokenAmount) return false;
235       uint160 price =
236           _sqrtPriceX96(seed.token, seed.quoteToken, poolTokenAmount, seed.quoteAmount);
237       return price >= TickMath.MIN_SQRT_PRICE && price <= TickMath.MAX_SQRT_PRICE
238           && Hooks.isValidHookAddress(IHooks(address(hook)), 0);
239   }
240
241   function _sqrtPriceX96(
242       address token,
243       address quoteToken,
244       uint256 tokenAmount,
245       uint256 quoteAmount
```

```
246   ) private pure returns (uint160 result) {
247       (uint256 amount0, uint256 amount1) = token < quoteToken
248           ? (tokenAmount, quoteAmount)
249           : (quoteAmount, tokenAmount);
250       uint256 ratioX192 = FullMath.mulDiv(amount1, Q192, amount0);
251       uint256 root = _sqrt(ratioX192);
252       if (root == 0 || root > type(uint160).max) revert InvalidConfiguration();
253       result = uint160(root);
254   }
```

Listing 2.1: contracts/production/src/v4/GraduationExecutor.sol

```
71    function graduate(address token) external nonReentrant {
72        LaunchedTokenRecord memory launch = _requireLaunch(token);
73        _requireState(token, GraduationStatus.NotGraduated);
74        GraduationSeed memory seed = _previewSeed(launch);
75        if (!IGraduationExecutor(_executor()).validateSeed(seed)) {
76            revert GraduationSeedNotFeasible(token);
77        }
78        _sweep(launch);
79    }
```

Listing 2.2: contracts/production/src/graduation/GraduationCoordinator.sol

Impact Extreme configurations may block graduation and recovery, or inflate the initial token price in the Uniswap V4 pool.

Suggestion In function `createToken()`, validate the graduation price and enforce a minimum `poolTokenAmount` to reject extreme configurations. Revise function `_validateSeed()` to safely support a price range slightly narrower than Uniswap V4's.

Feedback from the project The project states that it will validate launch configurations off-chain before registration to reject extreme values that could lead to the behaviors described above.

2.1.2 Potential overwrite of a valid recipient

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In contract `LaunchFactory`, function `proposeCreatorFeeRecipientOverride()` is intended to handle an unresponsive creator fee recipient. While a replacement proposal is pending, the original recipient may regain control and invoke function `transferCreatorFeeRecipient()` to update the recipient to a new address. This update makes the pending proposal obsolete. However, function `transferCreatorFeeRecipient()` does not clear the pending proposal. Once the timelock expires, the stale proposal can still be executed and overwrite the newly configured recipient.

```
254   function executeCreatorFeeRecipientOverride(address token) external {
255       PendingCreatorRecipientChange memory pending = pendingCreatorRecipientChanges[token];
```

```
256         if (pending.proposedRecipient == address(0)) revert NoPendingCreatorRecipientChange(token);
257         if (block.timestamp < pending.effectiveAt) {
258             revert CreatorRecipientTimelockNotElapsed(pending.effectiveAt);
259         }
260         if (block.timestamp > pending.expiresAt) {
261             revert CreatorRecipientChangeExpired(pending.expiresAt);
262         }
263         delete pendingCreatorRecipientChanges[token];
264         LaunchedTokenRecord storage record = _requireLaunch(token);
265         _setCreatorFeeRecipient(record, pending.proposedRecipient);
266     }
```

Listing 2.3: contracts/production/src/factory/LaunchFactory.sol

```
221     /// @notice Lets the current creator-fee recipient immediately transfer its role.
222     function transferCreatorFeeRecipient(address token, address newRecipient) external {
223         LaunchedTokenRecord storage record = _requireLaunch(token);
224         if (msg.sender != record.creatorFeeRecipient) revert NotCreatorFeeRecipient(msg.sender);
225         _setCreatorFeeRecipient(record, newRecipient);
226     }
227
228     /// @notice Starts a timelocked owner override of an unresponsive creator-fee recipient.
229     function proposeCreatorFeeRecipientOverride(address token, address newRecipient)
230         external
231         onlyOwner
232     {
233         LaunchedTokenRecord storage record = _requireLaunch(token);
234         if (newRecipient == address(0)) revert ZeroAddress();
235         if (newRecipient == record.creatorFeeRecipient) revert AlreadySet();
236         uint256 effectiveAt = block.timestamp + CREATOR_RECIPIENT_OVERRIDE_DELAY;
237         uint256 expiresAt = effectiveAt + CREATOR_RECIPIENT_OVERRIDE_WINDOW;
238         pendingCreatorRecipientChanges[token] = PendingCreatorRecipientChange({
239             proposedRecipient: newRecipient,
240             effectiveAt: uint64(effectiveAt),
241             expiresAt: uint64(expiresAt)
242         });
243         emit CreatorFeeRecipientOverrideProposed(token, newRecipient, effectiveAt, expiresAt);
244     }
```

Listing 2.4: contracts/production/src/factory/LaunchFactory.sol

Impact A stale proposal may replace the new legitimate recipient after the original recipient recovers and transfers the recipient role.

Suggestion Delete the entry `pendingCreatorRecipientChanges[token]` when function `transferCreatorFeeRecipient()` succeeds.

2.1.3 Inconsistent fee calculation between exact-input and exact-output swaps

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In contract `MemeHook`, function `afterSwap()` charges swap fees on the unspecified currency. Specifically, for exact-input swaps, the fee is charged on the output token, while for exact-output swaps, the fee is charged on the input token. This results in different costs between exact-input and exact-output swaps, as illustrated by the following example.

This example ignores price impact and rounding errors. For the purchase of launch tokens, let `X` be the amount of quote token that could buy `Y` launch tokens before fees, and let `f` be the total fee rate. An exact-input swap costs `X` and returns `Y(1 - f)` launch tokens while an exact-output swap returning the same `Y(1 - f)` costs `X(1 + f)(1 - f)`. Consequently, the exact-input swap costs more for the same token amount.

```
752   function afterSwap(
753       address sender,
754       PoolKey calldata key,
755       SwapParams calldata params,
756       BalanceDelta delta,
757       bytes calldata hookData
758   ) external returns (bytes4, int128) {
759       _requirePoolManager();
760       bytes32 poolId = PoolId.unwrap(key.toId());
761       PoolConfig storage config = poolConfig[poolId];
762       if (config.status != PoolStatus.Active) revert InvalidConfiguration();
763       if (sender == address(this)) {
764           return (IHooks.afterSwap.selector, 0);
765       }
766
767       bytes32 partnerId = _decodePartner(hookData);
768       bool specifiedTokenIs0 = (params.amountSpecified < 0 == params.zeroForOne);
769       (Currency feeCurrency, int128 rawBasis) =
770           specifiedTokenIs0 ? (key.currency1, delta.amount1()) : (key.currency0, delta.amount0())
                  ;
771       uint256 basis = uint256(rawBasis < 0 ? -int256(rawBasis) : int256(rawBasis));
772       uint256 hookFee = basis * config.hookFeeBps / BPS;
773       uint256 creatorTax = basis * config.creatorTaxBps / BPS;
774       uint256 totalFee = hookFee + creatorTax;
775       if (totalFee == 0) return (IHooks.afterSwap.selector, 0);
776
777       address currency = Currency.unwrap(feeCurrency);
778       (uint256 creatorImmediate, uint256 buybackBudget, uint256 partnerFee, uint256 protocolNet)
              =
779           _allocate(config, partnerId, hookFee);
780       pendingCreator[poolId][currency] += creatorImmediate + creatorTax;
781       pendingCreatorTax[poolId][currency] += creatorTax;
782       pendingBuyback[poolId][currency] += buybackBudget;
783       pendingProtocol[poolId][currency] += protocolNet;
784       if (partnerFee != 0) {
785           uint256 epochId = currentEpoch[poolId];
786           PartnerEpoch storage epoch = partnerEpoch[poolId][epochId];
787           if (!partnerActive[poolId][epochId][partnerId]) {
788               partnerActive[poolId][epochId][partnerId] = true;
789               ++epoch.activePartnerCount;
790           }
791           if (currency == config.token) {
```

```
792                 pendingPartnerLaunch[poolId][epochId][partnerId] += partnerFee;
793                 epoch.partnerLaunchInputTotal += partnerFee;
794             } else {
795                 pendingPartnerQuote[poolId][epochId][partnerId] += partnerFee;
796                 epoch.partnerDirectQuoteTotal += partnerFee;
797             }
798             emit PartnerFeeAccrued(poolId, epochId, partnerId, currency, partnerFee);
799         }
800         poolManager.take(feeCurrency, address(this), totalFee);
801         emit PoolFeeAllocated(
802             config.token,
803             poolId,
804             currency,
805             partnerId,
806             hookFee,
807             creatorTax,
808             creatorImmediate,
809             buybackBudget,
810             partnerFee,
811             protocolNet
812         );
813         return (IHooks.afterSwap.selector, totalFee.toInt128());
814     }
```

Listing 2.5: contracts/production/src/v4/MemeHook.sol

Impact Exact-input and exact-output swaps incur different costs for the same token amount, breaking the intended design.

Suggestion Revise the code logic accordingly.

2.2 Recommendation

2.2.1 Add validations in function `validateCombination()`

Status Confirmed

Introduced by `Version 1`

Description In library `LaunchConfigurationValidator`, function `validateCombination()` only validates that the quote economics are non-zero and that a minimal purchase of `phantomQuote` is quotable on the bonding curve. However, the function `_validateSeed()` in the contract `GraduationExecutor` additionally requires the derived pool price to fit within the `TickMath` price bounds and the square root of the intermediate price ratio to fit in `uint160`. As a result, a launch configuration accepted by function `validateCombination()` may still fail at graduation. It is recommended to apply the same validation in function `validateCombination()`, so that unsupported quote economics are rejected at configuration time.

```
43     function validateCombination(LaunchConfig memory config, QuoteTokenEconomics memory economics)
44         internal
45         pure
46     {
```

```
47        validateQuoteEconomics(economics);
48        _validateQuotable(config.supply, economics);
49    }
```

Listing 2.6: contracts/production/src/libraries/LaunchConfigurationValidator.sol

```
231       uint256 poolTokenAmount = FullMath.mulDiv(
232           seed.tokenAmount, seed.quoteAmount, seed.quoteAmount + seed.phantomQuote
233       );
234       if (poolTokenAmount == 0 || poolTokenAmount >= seed.tokenAmount) return false;
235       uint160 price =
236           _sqrtPriceX96(seed.token, seed.quoteToken, poolTokenAmount, seed.quoteAmount);
237       return price >= TickMath.MIN_SQRT_PRICE && price <= TickMath.MAX_SQRT_PRICE
238           && Hooks.isValidHookAddress(IHooks(address(hook)), 0);
```

Listing 2.7: contracts/production/src/v4/GraduationExecutor.sol

```
247       (uint256 amount0, uint256 amount1) = token < quoteToken
248           ? (tokenAmount, quoteAmount)
249           : (quoteAmount, tokenAmount);
250       uint256 ratioX192 = FullMath.mulDiv(amount1, Q192, amount0);
251       uint256 root = _sqrt(ratioX192);
252       if (root == 0 || root > type(uint160).max) revert InvalidConfiguration();
253       result = uint160(root);
```

Listing 2.8: contracts/production/src/v4/GraduationExecutor.sol

Suggestion Validate the price and intermediate ratio in the function `validateCombination()`.

Feedback from the project The project states that the necessary checks will be performed off-chain before launch creation to reject unsupported configurations upfront.

2.2.2 Align comments with implementation

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `BondingCurve`, the comment of the function `rescueCurveFees()` states that it is restricted to the configured sweep operator. However, the implementation requires `msg.sender == factory`, and the entry point in the contract `LaunchFactory` is gated by `onlyOwner`.

```
288       bool isOperator = msg.sender == IFeeSweepPolicyView(factory).feeSweepOperator();
289       if (!isOperator && msg.sender != creatorFeeRecipient) {
290           revert NotFeeSweepOperator(msg.sender);
```

Listing 2.9: contracts/production/src/curve/BondingCurve.sol

```
411   /// @notice Recovers fee liabilities after normal conversion cannot make progress.
412   /// @dev Restricted to the configured sweep operator and preserves recipient accounting.
413   function rescueCurveFees() external nonReentrant {
414       if (msg.sender != factory) revert Unauthorized(msg.sender);
```

Listing 2.10: contracts/production/src/curve/BondingCurve.sol

```
202   function rescueCurveFees(address token) external onlyOwner {
203       IBondingCurve(_requireLaunch(token).curve).rescueCurveFees();
204   }
```

Listing 2.11: contracts/production/src/factory/LaunchFactory.sol

```
156   /// @notice Indicates whether tracked quote reserves reached the snapshotted threshold.
157   function graduationReady() external view returns (bool) {
158       return token != address(0) && !graduated && sellableTokens() == 0;
159   }
```

Listing 2.12: contracts/production/src/curve/BondingCurveState.sol

Suggestion Rewrite the comment of function `rescueCurveFees()` to match the actual access control.

2.2.3 Validate liquidity in function `_validateSeed()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `GraduationExecutor`, the function `validateSeed()` returning true indicates that the current configuration is considered valid for creating the graduation pool. However, the function does not validate the liquidity of the full-range position derived from the relevant parameters. As a result, a configuration that passes `validateSeed()` may still revert in the function `createPool()` because the liquidity is zero or exceeds the per-tick cap for the configured tick spacing, and the affected assets can only be handled through the function `emergencyExit()`. It is recommended to add a liquidity check in the function `_validateSeed()`, keeping the return value of function `validateSeed()` consistent with the actual pool creation conditions.

```
223   function _validateSeed(GraduationSeed calldata seed) private view returns (bool) {
224       if (
225           seed.token == address(0) || seed.token == seed.quoteToken || seed.tokenAmount == 0
226               || seed.quoteAmount == 0 || seed.phantomQuote == 0
227               || seed.protocolFeeRecipient == address(0) || seed.tickSpacing <= 0
228               || seed.tickSpacing > TickMath.MAX_TICK_SPACING || seed.token.code.length == 0
229               || (seed.quoteToken != address(0) && seed.quoteToken.code.length == 0)
230       ) return false;
231       uint256 poolTokenAmount = FullMath.mulDiv(
232           seed.tokenAmount, seed.quoteAmount, seed.quoteAmount + seed.phantomQuote
233       );
234       if (poolTokenAmount == 0 || poolTokenAmount >= seed.tokenAmount) return false;
235       uint160 price =
236           _sqrtPriceX96(seed.token, seed.quoteToken, poolTokenAmount, seed.quoteAmount);
237       return price >= TickMath.MIN_SQRT_PRICE && price <= TickMath.MAX_SQRT_PRICE
238           && Hooks.isValidHookAddress(IHooks(address(hook)), 0);
239   }
```

Listing 2.13: contracts/production/src/v4/GraduationExecutor.sol

```
143         int24 tickLower = TickMath.minUsableTick(seed.tickSpacing);
144         int24 tickUpper = TickMath.maxUsableTick(seed.tickSpacing);
145         uint160 sqrtLowerX96 = TickMath.getSqrtPriceAtTick(tickLower);
146         uint160 sqrtUpperX96 = TickMath.getSqrtPriceAtTick(tickUpper);
147         (uint256 amount0Available, uint256 amount1Available) = seed.token < seed.quoteToken
148             ? (poolTokenAmount, seed.quoteAmount)
149             : (seed.quoteAmount, poolTokenAmount);
150         uint128 liquidity = LiquidityAmounts.getLiquidityForAmounts(
151             sqrtPriceX96, sqrtLowerX96, sqrtUpperX96, amount0Available, amount1Available
152         );
153         if (liquidity == 0) revert InvalidConfiguration();
```

Listing 2.14: contracts/production/src/v4/GraduationExecutor.sol

Suggestion Compute the full-range liquidity in function `_validateSeed()` and return `false` when it is zero or exceeds `tickSpacingToMaxLiquidityPerTick(tickSpacing)`.

2.2.4 Remove redundant computation in the function `buyExactIn()`

Status Confirmed

Introduced by `Version 1`

Description In the contract `BondingCurve`, the function `_buyExactIn()` recalculates the quote amount and fees when the calculated `tokenOut` exceeds the pool's sellable amount. However, the condition `tokenOut >= available` triggers this recalculation when `tokenOut == available`. In this case, the sellable amount is sufficient to fulfill the swap, so the subsequent computation is redundant. It is recommended to change the condition from `tokenOut >= available` to `tokenOut > available` to avoid the redundant computation.

```
141     function _buyExactIn(
142         uint256 quoteInRaw,
143         address tokenRecipient,
144         uint256 minTokenOut,
145         uint256 deadline,
146         bytes32 partnerId
147     ) private returns (uint256 tokenOut, uint256 quoteSpentRaw) {
148         _requireTradingOpen(deadline);
149         if (quoteInRaw == 0) revert InvalidAmount(quoteInRaw);
150         if (tokenRecipient == address(0)) revert InvalidTokenRecipient(tokenRecipient);
151
152         _receiveQuoteExact(msg.sender, quoteInRaw);
153
154         uint256 snipeTaxBps = currentSnipeTaxBps(tokenRecipient);
155         (uint256 baseFee, uint256 snipeTaxRaw, uint256 creatorTax, uint256 netQuoteIn) =
156             _feesFromGross(quoteInRaw, snipeTaxBps);
157         (uint256 quoteReserve, uint256 tokenReserve) = pricingReserves();
158         uint256 available = sellableTokens();
159         tokenOut = BondingCurveMath.getAmountOut(netQuoteIn, quoteReserve, tokenReserve);
160         quoteSpentRaw = quoteInRaw;
161
162         if (tokenOut >= available) {
```

```
163         tokenOut = available;
164         uint256 requiredNet = BondingCurveMath.getAmountIn(tokenOut, quoteReserve, tokenReserve
                );
165         uint256 totalFeeBps = curveFeeBps + snipeTaxBps + creatorTaxBps;
166         if (totalFeeBps >= BPS) {
167             revert TradeFeeConsumesInput(quoteInRaw, quoteInRaw);
168         }
```

Listing 2.15: contracts/production/src/curve/BondingCurve.sol

Suggestion Change the condition from `tokenOut >= available` to `tokenOut > available`.

Feedback from the project This behavior is by design.

2.2.5 Revise misleading variable names

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In contract `BondingCurve`, the `sweepFees()` uses the variable `quotedTokens` for the expected amount of `LauncherToken` from a buyback using quote assets. This name is misleading because it does not represent a quote amount. Meanwhile, the variable `buybackBurnBps` represents the portion of creator fees allocated to buybacks. However, the purchased tokens are locked rather than burned. It is recommended to rename both variables to reflect their actual purposes.

```
306         if (movementBps <= _feePolicy.maxInternalPriceImpactBps) {
307             uint256 quotedTokens =
308                 BondingCurveMath.getAmountOut(buybackAmount, quoteReserve, tokenReserve);
309             if (quotedTokens <= sellableTokens()) tokensLocked = quotedTokens;
310         }
```

Listing 2.16: contracts/production/src/curve/BondingCurve.sol

Suggestion Rename `quotedTokens` and `buybackBurnBps` to reflect their actual semantics.

2.2.6 Allow disabled partners to claim previously accrued fees

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In contract `PartnerRegistry`, the owner can disable a partner via the function `setPartnerEnabled()`. However, in contract `FeeEscrow`, the function `claimPartner()` rejects disabled partners, and rescue functions (i.e., `BondingCurve.rescuePartnerFees()` and `MemeHook.rescuePartnerFees()`) apply the same restriction. As a result, fees accrued before a partner was disabled cannot be claimed or rescued unless the owner re-enables the partner.

```
137   function claimPartner(bytes32 partnerId, address asset)
138       external
139       nonReentrant
140       returns (uint256 amount)
141   {
```

```
142         PartnerConfig memory config = IPartnerRegistry(partnerRegistry).partner(partnerId);
143         if (!config.exists) revert PartnerNotFound(partnerId);
144         if (!config.enabled) revert PartnerDisabled(partnerId);
```

Listing 2.17: contracts/production/src/escrow/FeeEscrow.sol

Suggestion Allow disabled partners to claim or rescue fees accrued before they were disabled.

2.3 Note

2.3.1 Potential centralization risk

Introduced by Version 1

Description In this project, several privileged roles (e.g., the owner of contract `LaunchFactory` and the fee sweep operator) can conduct sensitive operations, which introduces potential centralization risks. For example, the owner can set fee and tax parameters, manage quote-token economics and launch configurations, replace core dependencies, and trigger rescue flows. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

2.3.2 Ensure a correct slippage bound

Introduced by Version 1

Description In the contract `MemeHook`, the function `executeBuyback()` swaps via function `_swapExactInput()`, which derives the price limit from the pool's current spot price and a configured impact bound, and enforces a caller-supplied minimum output. The caller must specify the minimum output based on a reliable estimate of the expected output and an acceptable slippage tolerance. If the minimum is too low, the swap may execute at an unfavorable price within the impact limit, spending the buyback budget for fewer tokens than intended.

```
272   function executeBuyback(address token, uint256 quoteAmount, uint256 minTokensOut)
273       external
274       returns (uint256 quoteSpent, uint256 tokensOut)
275   {
276       if (msg.sender != address(this)) revert Unauthorized(msg.sender);
277       PoolConfig storage config = poolConfig[poolForToken[token]];
278       if (config.status != PoolStatus.Active) revert InvalidConfiguration();
279       (quoteSpent, tokensOut) = _swapExactInput(config, config.quoteToken, quoteAmount);
280       if (tokensOut != 0) {
281           if (tokensOut < minTokensOut) revert SlippageExceeded(tokensOut, minTokensOut);
282           address vault = ILaunchFactoryDependencies(factory).buybackVault();
283           IERC20(token).forceApprove(vault, tokensOut);
284           IBuybackVault(vault).lock(token, tokensOut);
285       } else if (quoteSpent != 0) {
286           revert SlippageExceeded(0, minTokensOut);
287       }
288   }
```

Listing 2.18: contracts/production/src/v4/MemeHook.sol

```
669   function _swapExactInput(PoolConfig storage config, address currencyIn, uint256 amountIn)
670       private
671       returns (uint256 amountInConsumed, uint256 amountOut)
672   {
673       PoolKey memory key = _poolKey(config.token, config.quoteToken, config.tickSpacing);
674       bool zeroForOne = currencyIn == Currency.unwrap(key.currency0);
675       if (!zeroForOne && currencyIn != Currency.unwrap(key.currency1)) {
676           revert InvalidConfiguration();
677       }
678       (uint160 currentPrice,,,) = poolManager.getSlot0(key.toId());
679       uint160 limit = _priceLimit(currentPrice, zeroForOne, config.maxInternalPriceImpactBps);
680       (amountInConsumed, amountOut) = abi.decode(
681           poolManager.unlock(abi.encode(key, zeroForOne, amountIn, limit)), (uint256, uint256)
682       );
683   }
684
685   function _priceLimit(uint160 sqrtPriceX96, bool zeroForOne, uint256 maxImpactBps)
686       private
687       pure
688       returns (uint160)
689   {
690       uint256 factor = BPS - maxImpactBps;
691       if (zeroForOne) {
692           uint256 limit = FullMath.mulDiv(sqrtPriceX96, factor, BPS);
693           return limit <= TickMath.MIN_SQRT_PRICE ? TickMath.MIN_SQRT_PRICE + 1 : uint160(limit);
694       }
695       uint256 upperLimit = FullMath.mulDiv(sqrtPriceX96, BPS, factor);
696       return
697           upperLimit >= TickMath.MAX_SQRT_PRICE
698               ? TickMath.MAX_SQRT_PRICE - 1
699               : uint160(upperLimit);
```

Listing 2.19: contracts/production/src/v4/MemeHook.sol

2.3.3 Different fee rates across launch phases

Introduced by Version 1

Description Bonding curve trades use `curveFeeBps` as the base trading fee rate, while Uniswap V4 trades after graduation use `hookFeeBps`. These parameters are configured independently and are not required to be equal, so the applicable base fee rate may change after graduation.

```
162       if (tokenOut >= available) {
163           tokenOut = available;
164           uint256 requiredNet = BondingCurveMath.getAmountIn(tokenOut, quoteReserve, tokenReserve
                  );
165           uint256 totalFeeBps = curveFeeBps + snipeTaxBps + creatorTaxBps;
166           if (totalFeeBps >= BPS) {
167               revert TradeFeeConsumesInput(quoteInRaw, quoteInRaw);
```

```
168         }
```

Listing 2.20: contracts/production/src/curve/BondingCurve.sol

```
772     uint256 hookFee = basis * config.hookFeeBps / BPS;
773     uint256 creatorTax = basis * config.creatorTaxBps / BPS;
774     uint256 totalFee = hookFee + creatorTax;
775     if (totalFee == 0) return (IHooks.afterSwap.selector, 0);
```

Listing 2.21: contracts/production/src/v4/MemeHook.sol

2.3.4 Careful configuration of the partner share cap

Introduced by Version 1

Description In contract `PartnerRegistry`, variable `maxPartnerShareOfPlatformBps` defines the maximum share of the platform fee that each partner can receive. Function `setPartnerPolicy()` updates this limit but only allows increases, meaning the limit cannot be lowered once set. Therefore, the project should configure this limit carefully.

```
162   function _setPartnerPolicy(PartnerPolicy memory newPolicy) private {
163       if (
164           newPolicy.defaultPartnerShareOfPlatformBps > newPolicy.maxPartnerShareOfPlatformBps
165               || newPolicy.maxPartnerShareOfPlatformBps > LaunchConstants.BPS
166       ) revert InvalidConfiguration();
167       if (
168           partnerNonce != 0
169               && newPolicy.maxPartnerShareOfPlatformBps
170                   < _partnerPolicy.maxPartnerShareOfPlatformBps
171       ) revert InvalidConfiguration();
172
173       _partnerPolicy = newPolicy;
174       emit PartnerPolicyUpdated(
175           newPolicy.defaultPartnerShareOfPlatformBps, newPolicy.maxPartnerShareOfPlatformBps
176       );
177   }
```

Listing 2.22: contracts/production/src/partner/PartnerRegistry.sol

2.3.5 Fee sweep requires settlement of all partners

Introduced by Version 1

Description In the contract `MemeHook`, the function `finalizePoolFeeSweep()` requires that every active partner of the current epoch has been settled via the function `settlePoolFeeSweepPartners()` (i.e., `settledPartnerCount == activePartnerCount`). Until this requirement is met, the epoch cannot be finalized, and subsequent invocations to function `beginPoolFeeSweep()` are also blocked. Therefore, the project must ensure that all active partners settle in time for each epoch.

```
516         ++epoch.settledPartnerCount;
517     }
```

```
518
519         if (
520             epoch.settledPartnerCount == epoch.activePartnerCount
521                 && epoch.partnerConvertedQuoteRemaining != 0
522         ) {
523             bytes32 finalPartnerId = partnerIds[length - 1];
524             uint256 rounding = epoch.partnerConvertedQuoteRemaining;
525             epoch.partnerConvertedQuoteRemaining = 0;
526             bool merged;
527             for (uint256 i; i < creditCount; ++i) {
528                 if (creditIds[i] == finalPartnerId) {
529                     amounts[i] += rounding;
530                     merged = true;
531                     break;
532                 }
533             }
534             if (!merged) {
535                 creditIds[creditCount] = finalPartnerId;
536                 amounts[creditCount] = rounding;
537                 ++creditCount;
538             }
539             totalCredit += rounding;
540         }
541
542         if (totalCredit != 0) {
543             assembly ("memory-safe") {
544                 mstore(creditIds, creditCount)
545                 mstore(amounts, creditCount)
546             }
547             address escrow = ILaunchFactoryDependencies(factory).feeEscrow();
548             IERC20(config.quoteToken).forceApprove(escrow, totalCredit);
549             IFeeEscrow(escrow)
550                 .creditPartnerTokenBatch(config.quoteToken, creditIds, amounts, totalCredit);
551         }
552         emit PoolFeeSweepPartnersSettled(poolId, epochId, length, totalCredit);
553     }
554
555     /// @notice Finalizes an epoch after all snapshotted partner launch-token input is settled.
556     /// @dev Any integer rounding remainder is assigned during final partner settlement.
557     function finalizePoolFeeSweep(bytes32 poolId, uint256 epochId) external {
558         if (settlingEpoch[poolId] != epochId + 1) revert InvalidConfiguration();
559         PartnerEpoch storage epoch = partnerEpoch[poolId][epochId];
560         if (
561             epoch.settledPartnerCount != epoch.activePartnerCount
```

Listing 2.23: contracts/production/src/v4/MemeHook.sol

2.3.6 Ensure off-chain validation of launch configurations

Introduced by Version 1

Description In the project, the owner can revoke function `LaunchConfiguration.addLaunchConfig()` to register configurations that creators can select when launching tokens. These con-

figurations define the bonding curve economics and affect the subsequent initialization of the Uniswap V4 pool at graduation. Therefore, the project should verify configurations off-chain before registration to prevent unexpected behaviors during graduation. Two example scenarios are described below.

1. In the configuration and launch flow, the project does not verify whether the resulting full-range liquidity exceeds Uniswap V4's per-tick liquidity cap. This limit is only enforced when liquidity is added during graduation, potentially causing the transaction to revert and preventing graduation.

2. In contract `GraduationExecutor`, the function `_validateSeed()` computes `poolTokenAmount` via `FullMath.mulDiv()`, which rounds down. The function `_sqrtPriceX96()` derives the initial Uniswap V4 pool price from the rounded amount. When `tokenAmount` is small, the initial pool price deviates from the bonding curve and creates an arbitrage opportunity after the pool opens.

2.3.7 Refactoring and new features are outside the audit scope

Introduced by `Version 2`

Description Version 2 introduces a new `contracts/v2` directory. The directory contains additional refactoring and new features alongside the code changes to address the findings identified in Version 1. This audit covers only the changes in the `contracts/v2` directory, relative to Version 1, that address those findings. All other changes fall outside the audit scope.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS