CANTINA

Zora Coinbase Creator Coin

Security Review

Cantina Managed review by:

0xRajeev, Lead Security Researcher
Cryptara, Security Researcher

October 21, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Base is a secure, low-cost, builder-friendly Ethereum L2 built to bring the next billion users onchain.

From Sep 14th to Sep 16th the Cantina team conducted a review of zora-protocol on commit hash ae9424ab. The team identified a total of **14** issues:

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	4	3	1
Gas Optimizations	0	0	0
Informational	10	7	3
Total	**14**	**10**	**4**

3 Findings

3.1 Low Risk

3.1.1 Stale poolKey mapping in `ZoraV4CoinHook` after hook migration may cause unexpected behavior

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: In the `ZoraV4CoinHook` contract, the `migrateLiquidity` function facilitates the migration of liquidity from an old hook to a new one. However, the function does not remove the old pool key from the `poolCoins` mapping after migration. This omission can result in stale state, where the old pool key remains associated with its coin and positions, potentially allowing swaps or other operations to be executed on liquidity that has already been migrated. This can lead to unexpected behavior, inconsistencies in reward distribution, and possible security risks if the migrated liquidity is assumed to be unavailable but is still referenced in the contract's state.

Recommendation: It is advisable to remove the old pool key from the `poolCoins` mapping after a successful migration. This ensures that the contract state accurately reflects the current liquidity configuration and prevents further interactions with migrated positions. The removal can be performed by deleting the mapping entry for the old pool key within the `migrateLiquidity` function after migration is complete. Additionally, review any related logic to ensure that stale references do not persist elsewhere in the contract.

Zora: Fixed in commit 32d683d.

Cantina Managed: Verified.

3.1.2 Missing checks allow migration to unregistered hooks

Severity: Low Risk

Context: ZoraHookRegistry.sol#L34-L36, ZoraHookRegistry.sol#L71-L92, ZoraV4CoinHook.sol#L183-L185, ZoraV4CoinHook.sol#L373-L376

Description: `ZoraV4CoinHook.migrateLiquidity(...)` allows a coin owner to migrate liquidity positions from an old hook to a new one, which may be desirable if the new hook adds new functionality or fixes any issues in the old hook.

While `migrateLiquidity(...)` checks if an upgrade path has been registered by the protocol from the old hook to the new hook via `HookUpgradeGate.isRegisteredUpgradePath(oldHook, newHook)`, it does not check if the `newHook` is registered in the hook registry via `ZoraHookRegistry.isRegisteredHook(newHook)`. This allows migration to a new hook that is unregistered but has an upgrade path available, which may happen if the protocol unregisters a hook via `removeHooks(...)` in `ZoraHookRegistry` but misses to remove its upgrade path in `HookUpgradeGate`.

Additionally, `HookUpgradeGate.registerUpgradePath(...)` allows protocol to register optional upgrade paths for hooks. However, it does not check if the upgrade path is being registered for registered hooks via `ZoraHookRegistry.isRegisteredHook(hook)`. This allows accidentally registering upgrade paths to unregistered hooks, which will allow coin owners to migrate their liquidity to unregistered hooks because migration also does not check for unregistered hooks currently.

Recommendation: Consider:

1. Checking `ZoraHookRegistry.isRegisteredHook(newHook)` during migration along with `HookUpgradeGate.isRegisteredUpgradePath(oldHook, newHook)`.
2. Ensuring that hooks are simultaneously unregistered and their upgrade paths removed.
3. Checking `isRegisteredHook(hook)` in `HookUpgradeGate.registerUpgradePath(...)`.

Zora: Acknowledged. The hook registration is more informational for 3rd parties and does not enforce any security on the actual implementation. May be a good call to validate that the hook is registered in the factory if we do have that reference. We won't fix this as the registry is more for informational purposes.

Cantina Managed: Acknowledged.

3.1.3 Creator coin vesting duration is 1.25 days less than expected

Severity: Low Risk

Context: CreatorCoin.sol#L65-L66, CreatorCoinConstants.sol#L13

Description: Creator coins are minted with a total supply of 10e9 tokens with half of that (500 * 10e6) allocated to the liquidity pool. The remaining half is claimable by the creator over a vesting period that is expected to be 5 years. However, this constant is set to `CREATOR_VESTING_DURATION = 5 * 365 days; // 5 years`, which ignores leap years. Given that there is 1 leap year every four years, this creator vesting duration is 1.25 days less than expected.

Recommendation: Consider using the more accurate calculation of `5 * 365.25` days for `CREATOR_VESTING_DURATION` to account for leap years.

Zora: Fixed in commit d3808d55.

Cantina Managed: Fix verified.

3.1.4 Swaps with native ETH will fail if one of the reward recipients cannot receive native ETH

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: Swaps with native ETH currency will fail if one of the reward recipients cannot receive native ETH because `CoinRewardsV4.distributeMarketRewards(...)` in `ZoraV4CoinHook._afterSwap()` will revert in `_transferCurrency()`.

Finding Description: `ZoraV4CoinHook._afterSwap()` calls `CoinRewardsV4.distributeMarketRewards(...)` to distribute rewards to all stakeholders. However, if the swap currency is native ETH and any of the five stakeholders of `payoutRecipient`, `platformReferrer`, `protocolRewardRecipient`, `doppler` or `tradeReferrer` is accidentally configured to not receive native ETH, then `_transferCurrency()` will revert with `ICoin.EthTransferFailed`. All coin swaps with native ETH currency will fail.

However, it is very unlikely that any of the five stakeholders will reject native ETH. Besides griefing, there is no incentive for external stakeholders of `platformReferrer`, `doppler` or `tradeReferrer` to do so.

Note: This was reported and acknowledged in the previous review.

Recommendation: Consider:

1. Emitting an event instead of reverting for failed ETH reward transfers.
2. Remitting the rewards in WETH instead of native ETH.
3. Adding a try/catch block to skip over reverts.

Zora: If the platform referrer is immutable; if it is set to an address that cannot receive ETH, it can brick swaps on the coin, as they would revert. Both the creator recipient and trade referral are changeable, the former via updating the payout recipient on the coin, and the latter via updating the trade referrer argument when doing a swap; therefore we do not need to worry about permanently bricking swaps, and don't need a backup recipient.

See commit 9c6d241.

Cantina Managed: Fixed in commit 9c6d241 by introducing a `backupRecipient` parameter for `_transferCurrency(...)`, which is set to `protocolRewardRecipient` and sent the ETH rewards if transfer fails for the initially set `platformReferrer` and `doppler` recipients.

3.2 Informational

3.2.1 Duplicated and inconsistent constants across `CoinConstants` and `MarketConstants`

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The codebase currently defines coin supply and allocation constants in multiple libraries, such as `CoinConstants`, `MarketConstants`, and `CreatorCoinConstants`. For example, `CoinConstants.POOL_LAUNCH_SUPPLY` and `MarketConstants.CONTENT_COIN_MARKET_SUPPLY` represent the same value but are declared separately. This duplication introduces a risk of inconsistency if values are updated in one location but not the other, potentially resulting in mismatches in total supply, market supply, or creator allocations. Additionally, creator and content coin-related constants are scattered across these files, making the codebase harder to maintain and increasing the likelihood of errors during upgrades or refactoring. The lack of dynamic relationships between these constants further exacerbates the risk, as dependent values are not automatically updated when base values change.

Recommendation: To mitigate these risks, it is recommended to centralize all coin-related constants into dedicated libraries, such as `CreatorCoinConstants` for creator coins and a separate library for content coins. Move all relevant values from `CoinConstants` and `MarketConstants` to ensure a single source of truth for each coin type. Where possible, define dependent values dynamically (e.g., `CREATOR_VESTING_SUPPLY = TOTAL_SUPPLY - CREATOR_COIN_MARKET_SUPPLY`) to maintain consistency and reduce manual errors. This restructuring will improve maintainability, simplify future upgrades, and ensure that supply and allocation logic remains coherent throughout the protocol.

Zora: All constants were consolidated in commit b571fe5.

Cantina Managed: Verified. Although, the dynamic constant values risk was not addressed and is still present.

3.2.2 Restrictive ETH receive function in `ZoraV4CoinHook`

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The `receive()` function in the `ZoraV4CoinHook` contract is restricted by the `onlyPoolManager` modifier, preventing direct ETH transfers from any address except the PoolManager. While this restriction reduces the risk of accidental or malicious ETH deposits, it does not fully prevent ETH from being sent to the contract through other mechanisms, such as self-destruct operations from other contracts or protocol-level rewards. As a result, ETH could become stuck in the contract with no mechanism to recover or withdraw these funds, leading to potential loss of assets.

Recommendation: Consider implementing a recovery mechanism that allows the contract owner or a designated administrator to withdraw any ETH that may become stuck in the contract. This could be achieved by adding a withdrawal function with appropriate access controls. Alternatively, evaluate whether the receive function is necessary at all, and if not, remove it to further reduce the attack surface. If the contract must accept ETH for protocol reasons, ensure there is a clear and secure process for handling and recovering these funds.

Zora: We don't want to include ownership ability on the coin hook even for ETH recovery and if someone does a self-destruct ideally they know they are locking those funds. There shouldn't be any risk in adding the receive function here since it is gated and not having a receive function on the bytecode level is essentially solidity generating an revert on any received ETH rather than a entirely different type of contract on the bytecode level.

Cantina Managed: Acknowledged.

3.2.3 Missing `_disableInitializers()` in `ZoraHookRegistry`

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The `ZoraHookRegistry` contract is upgradeable and uses an initializer function, but it does not call `_disableInitializers()` in its constructor. This omission leaves the contract vulnerable to potential re-initialization attacks, where an attacker could call the initializer on the implementation contract and manipulate critical state variables or ownership. This is a well-known risk for upgradeable contracts that do not explicitly disable initializers on the implementation. However, the client has reported that this contract is already deployed and actively used, making direct remediation challenging.

Recommendation: Ideally, `_disableInitializers()` should be called in the constructor of the implementation contract to prevent any future initialization attempts on the implementation itself. For contracts already deployed, consider deploying a patched version and migrating usage to the new contract if feasible. Alternatively, closely monitor the implementation address for any suspicious activity and ensure that only proxy contracts interact with the registry. If migration is not possible, document the risk and educate protocol users and maintainers about the potential vulnerability.

Zora: This contract is not upgradeable, thus the initialize function is meant only to be called once. Adding `_disableInitializers()` to the constructor breaks this flow and isn't possible. A comment to describe the deployment constraints / flow may be helpful here to clarify.

Comment added in commit 190a515.

Cantina Managed: Verified.

3.2.4 Misleading documentation on `ContentCoin` backing currency

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The documentation for the `ContentCoin` contract states that content coins use creator coins as their backing currency. However, this is not enforced in the contract logic, and the client has clarified that content coins can use any currency as their backing, including ETH or other tokens. This discrepancy between the documentation and actual implementation can lead to confusion for integrators, auditors, and users, potentially resulting in incorrect assumptions about protocol behavior or integration requirements.

Recommendation: Update the documentation and comments in the `ContentCoin` contract to accurately reflect the intended functionality, specifying that content coins can use any currency as their backing. Ensure that all references to creator coins as the exclusive backing currency are removed or clarified. Additionally, review related documentation and developer guides to maintain consistency across the codebase and prevent future misunderstandings.

Zora: Fixed in commit c0da39f.

Cantina Managed: Verified.

3.2.5 Using the same version across different contracts is misleading

Severity: Informational

Context: ZoraHookRegistry.sol#L64-L68, ZoraV4CoinHook.sol#L159-L161, ContractVersionBase.sol#L9-L14, ZoraFactoryImpl.sol#L116

Description: `ContractVersionBase` is inherited by `BaseCoin`, `ZoraV4CoinHook` and `ZoraFactoryImpl`, all of which will return `2.2.0` for `contractVersion()`. Even if only the coin is upgraded to a new version at some point in future, all of them will return the same new upgraded version. Unless they are treated as a unified version for tracking reasons, using the same version across upgraded and not upgraded contracts is misleading.

Recommendation: Consider different versioning for different contracts.

Zora: Acknowledged. This is a weird artifact of how we handle versioning. `ContractVersionBase` is autogenerated based off the `coins` npm package version. As the protocol is upgraded as a whole this version will change and it's quite easy to trace back the source code from the contractVersion in this way as we also have an NPM release at that version when we deploy the contracts.

```
// This file is automatically generated by code; do not manually update
```

In effect, these contracts do end up being different versions on chain. They're just versioned as a whole unit when released. As noted, the side effect here is that versions may increase with no code changes if they are indeed deployed. Deployments, however, allow this flow to work pretty well as we only deploy changed code to the chain or when the change is needed.

This system can be improved to version the contracts separately when changed, but it works for now as is. We won't fix it for now.

Cantina Managed: Acknowledged.

3.2.6 `ReentrancyGuardUpgradeable` is not required for `BaseCoin`

Severity: Informational

Context: BaseCoin.sol#L57

Description: `BaseCoin` inherits `ReentrancyGuardUpgradeable`, but this is not required given that `nonReentrant` is never used on any of its functions. This appears to be supporting legacy functionality that has since been removed.

Recommendation: Consider removing `ReentrancyGuardUpgradeable` and any other unneeded inheritance to avoid bloat, improve readability and prevent any unexpected behavior in future.

Zora: Fixed in commit daad8830.

Cantina Managed: Fix verified.

3.2.7 Using `initializer` modifier on constructor is non-standard

Severity: Informational

Context: BaseCoin.sol#L97

Description: `BaseCoin` uses the `initializer` modifier on constructor, where the intent is to prevent calls to initialize on the implementation itself when used in the proxy pattern. While this appears to effectively perform the same functionality as calling `_disableInitializers()` within the constructor, this is not a widely-used pattern.

Recommendation: Consider using `_disableInitializers()` within the constructor to lock implementation coin contracts designed to be cloned.

Zora: Acknowledged. This is a different way of doing things that accomplishes the same thing. We won't fix for now.

Cantina Managed: Acknowledged.

3.2.8 Stale/Incorrect/Missing comments reduce readability

Severity: Informational

Context: ContentCoin.sol#L17-L21, CreatorCoin.sol#L77-L80, ZoraV4CoinHook.sol#L170-L185, ZoraV4CoinHook.sol#L346-L349, CoinRewardsV4.sol#L178-L190

Description: There are some stale/incorrect and missing comments across the codebase, which reduce readability. Some examples are:

1. `* @notice Content coin implementation that uses creator coins as backing currency` notes that content coins are only backed by their creator coin, but this is not enforced because they may be backed by ETH or even other content coins.
2. `claimVesting()` has `/// @dev Optimized for frequent calls from Uniswap V4 hooks`, but this is never called from the hook.
3. `distributeMarketRewards()` is missing `@param` for `coinType`.
4. `unlockCallback()` has `/// @notice Internal fn called when the PoolManager is unlocked. Used to mint initial liquidity positions.`, but is missing that this is also used to burn all positions during migration.
5. `initializeFromMigration(...)` has `// Verify that the caller (new hook) is authorized to perform this migration`, but the caller is the old hook.

Recommendation: Consider adding/fixing the above comments.

Zora: Fixed in commit c0da39f1.

Cantina Managed: Fix verified.

3.2.9 Forced downcasting is risky

Severity: Informational

Context: CoinRewardsV4.sol#L89-L94

Description: `convertDeltaToPositiveUint128(...)` downcasts `int256 delta` to `uint128(uint256(delta))` as follows:

```
function convertDeltaToPositiveUint128(int256 delta) internal pure returns (uint128) {
    if (delta < 0) {
        revert SafeCast.SafeCastOverflow();
    }
    return uint128(uint256(delta));
}
```

While `delta` values higher than `uint128` may be unlikely, forced downcasting of `delta` is nevertheless risky for any edge-case scenarios.

Recommendation: Consider using an appropriate safe cast function or adding a bounds check `delta <= type(uint128).max` as best practice.

Coinbase: Fixed in commit 5093c5ef.

Cantina Managed: Fix verified.

3.2.10 Creator coins do not support `postDeployHook` functionality

Severity: Informational

Context: ZoraFactoryImpl.sol#L76-L120, ZoraFactoryImpl.sol#L123-L137, ZoraFactoryImpl.sol#L165-L169

Description: Content coins support `postDeployHook`, which is used to implement `afterCoinDeploy(...)` for supporting initial token purchases immediately at deployment. However, Creator coin does not support this functionality, which may be desirable.

Recommendation: Consider unifying the Creator coin functionality with that of Content coin by supporting `postDeployHook` functionality.

Coinbase: Fixed in commit 7df94915.

Cantina Managed: Fix verified.