Zora Protocol

Security Review

Cantina Managed review by:

0xRajeev, Lead Security Researcher
Cryptara, Security Researcher

August 27, 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	**Description**
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

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 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

Zora is a social network where every post is a coin.

From Jun 2nd to Jun 11th the Cantina team conducted a review of zora-protocol on commit hash eb2c8b99. The team identified a total of **16** issues:

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	0	0	0
Low Risk	5	2	3
Gas Optimizations	0	0	0
Informational	10	5	5
Total	**16**	**8**	**8**

3 Findings

3.1 High Risk

3.1.1 Coin pools are configured for half the fees than intended causing loss of rewards for all stakeholders

Severity: High Risk

Context: CoinSetup.sol#L35-L36, MarketConstants.sol#L12-L18, ZoraFactoryImpl.sol#L217-L226

Summary: Coin UniV4 pools are configured for half the fees than intended, thus causing loss of rewards for all stakeholders.

Finding Description: Coins created for Uniswap V4 pools are intended to configured with `V4_LP_FEE = 20000` (i.e. 2%) but instead are configured with the legacy pool fees of `LP_FEE = 10000` (i.e. 1%).

While `CoinDopplerMultiCurve.setupPool()` returns `poolConfiguration` with `fee: MarketConstants.V4_LP_FEE` as shown below, the configured pool will actually reward half of that LP FEE. This effectively results in loss of expected rewards to all stakeholders.

```
poolConfiguration = PoolConfiguration({
    version: CoinConfigurationVersions.DOPPLER_MULTICURVE_UNI_V4_POOL_VERSION,
    numPositions: uint16(totalDiscoveryPositions + 1), // Add one for the final tail position
    fee: MarketConstants.V4_LP_FEE,
    tickSpacing: MarketConstants.TICK_SPACING,
    numDiscoveryPositions: numDiscoveryPositions_,
    tickLower: tickLower_,
    tickUpper: tickUpper_,
    maxDiscoverySupplyShare: maxDiscoverySupplyShare_
});
```

Impact Explanation: Medium, because while pool configuration shows a fee of 2% it only rewards 1%, which that leads to loss of expected rewards for all stakeholders (creator, platform/trade referrers, Doppler and protocol). Creator token trading rewards are a key part of the Zora protocol.

Likelihood Explanation: High, because this happens for all coins deployed with the current integration of Uniswap V4 pools.

Recommendation: Consider using the correct `MarketConstants.V4_LP_FEE` in both `CoinV4.poolConfiguration` and `CoinV4.poolKey`. For example, `CoinSetup.buildPoolKey()` could take `poolConfiguration` as a parameter in `ZoraFactoryImpl._setupV4Coin()` and build based on `poolConfiguration.fee` instead of relying again on the fee constant.

Zora: Fixed in commit c266b79.

Cantina Managed: Reviewed that commit c266b79 uses `MarketConstants.LP_FEE_V4` (updated constant name from `V4_LP_Fee`) in both `poolConfiguration` and `poolKey`. It also updates `TRADE_REFERRAL_REWARD_BPS` value to 1500 (15%) from 1000 (10%).

3.2 Low Risk

3.2.1 Using a single-step ownership transfer pattern for `ZoraFactoryImpl` is risky

Severity: Low Risk

Context: ZoraFactoryImpl.sol#L35, ZoraFactoryImpl.sol#L363

Description: `ZoraFactoryImpl` uses single-step ownership transfer inherited from OpenZeppelin's `OwnableUpgradeable`, which is used by `_authorizeUpgrade` with the `onlyOwner` modifier. This is risky because it is susceptible to common mistakes such as transfers of ownership to incorrect accounts, or to contracts that are unable to interact with the permission system as documented in `Ownable2StepUpgradeable.sol`.

Recommendation: Consider using OpenZeppelin's `Ownable2StepUpgradeable.sol`, which enforces a two-step ownership transfer using `transferOwnership()` and `acceptOwnership()`.

Zora: Fixed in commit acb9ff94.

Cantina Managed: Fix verified.

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

Severity: Low Risk

Context: CoinRewardsV4.sol#L120-L155

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

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

Impact Explanation: High, because all coin swaps with native ETH currency will fail.

Likelihood Explanation: Very low, because it is very unlikely that any of the five stakeholders will reject native ETH. Besides griefing, there is no incentive for external stakeholders of `coin.platformReferrer()`, `coin.doppler()` or `tradeReferrer` to do so.

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. Redesigning to a pull-based withdrawal as in `ProtocolRewards` instead of pushing the rewards.

Zora: Acknowledged. We're okay with reverting if any of the recipient addresses can't receive ETH.

Cantina Managed: Acknowledged.

3.2.3 Unauthorized coin registration in `ZoraV4CoinHook`

Severity: Low Risk

Context: ZoraV4CoinHook.sol#L97-L99

Description: The `ZoraV4CoinHook` contract's `_afterInitialize` function lacks proper validation of the pool key and coin relationship, allowing malicious actors to initialize unauthorized pools with custom tokens that pass the basic coin verification. The current implementation only checks if the sender is a coin through `V4Liquidity.isCoin(coin)`, which merely verifies if the contract implements the `IHasRewardsRecipients` interface. This weak verification allows any contract that implements this interface to register as a valid coin and claim a pool hash within the hook contract. A malicious actor could create a spoof contract that implements `IHasRewardsRecipients` but isn't a legitimate `CoinV4` instance, potentially creating unauthorized pools and manipulating the hook's state.

Impact Explanation: Medium, it could create confusion in the protocol's pool management and potentially affect reward distributions if the hook's logic is influenced by the total number of registered pools.

Likelihood Explanation: Low, because the economic incentive is limited unless there's a specific reward mechanism that can be exploited. Additionally, the attack would be easily detectable as the spoofed coins would be registered in the hook's state.

Recommendation: The hook contract should implement a strict pool key validation mechanism. This can be achieved by:

1. Maintaining a mapping of authorized pool keys to their corresponding coins.
2. Adding a validation step in the `_afterInitialize` function that checks if the pool key matches the expected configuration for the coin.
3. Implementing a whitelist mechanism for authorized coins that can use the hook.
4. Adding additional validation in the `_afterSwap` function to ensure the pool is authorized.

The hook should also consider implementing a more robust coin verification system that goes beyond the basic ERC165 interface check, such as verifying against the factory's authorized coin list.

Zora: Fixed.

Cantina Managed: Fixed. The code is now using a `DeployedCoinVersionLookup` contract to store all deployed coins from the factory with its deploy version. During validation instead of relaying on the interface the contract values are checked.

3.2.4 Invalid tick alignment in `DopplerMath` library

Severity: Low Risk

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

Description: The `alignTickToTickSpacing` function in `DopplerMath` library fails to properly handle boundary cases when aligning ticks to tick spacing, particularly for negative values near `MIN_TICK` and positive values near `MAX_TICK`. The current implementation can produce ticks that are outside the valid Uniswap V4 tick range (-887272 to 887272), leading to potential failures in pool operations and incorrect liquidity calculations.

Input Tick	isToken0	Aligned Tick	Valid?	Issue
887272	True	887200	Yes	Rounded down, still in range
887272	False	887400	No	Exceeds MAX_TICK
-887272	True	-887600	No	Below MIN_TICK
-887272	False	-887200	Yes	Rounded up, still in range

This issue propagates through the `calculateLogNormalDistribution` function, which uses these aligned ticks to create LP positions. When these positions are used to mint liquidity via `modifyPosition`, the operation will revert due to Uniswap's internal tick bounds validation.

Impact Explanation: Medium, because while the invalid tick alignment will cause pool operations to revert at the Uniswap V4 pool level due to its internal tick bounds validation, there's a potential for discrepancies in mid-range values. If the alignment function produces invalid ticks but the actual pool operations use different (valid) tick values, this could lead to inconsistencies between the expected and actual liquidity positions. This discrepancy could affect future calculations and operations that rely on the tick alignment logic.

Likelihood Explanation: Low, because the issue only manifests when operating near the extreme tick boundaries (-887272 and 887272) with specific tick spacing values. The current implementation in `CoinDopplerMultiCurve` uses a tick spacing of 200, which means the issue would only occur when trying to create positions very close to the minimum or maximum possible ticks.

Recommendation: Implement a `checkTicks` validation function in the `DopplerMath` library that verifies tick values are within the valid range. This function should be called after any tick alignment operations to ensure the resulting ticks are valid. The validation should be added to both `alignTickToTickSpacing` and any functions that use its results, particularly `calculateLogNormalDistribution`. Example implementation of `checkTicks`:

```
function checkTicks(int24 tickLower, int24 tickUpper) internal pure {
    if (tickLower >= tickUpper) {
        revert InvalidTickRangeMisordered(tickLower, tickUpper);
    }
    if (tickLower < TickMath.MIN_TICK) {
        revert TickLowerOutOfBounds(tickLower);
    }
    if (tickUpper > TickMath.MAX_TICK) {
        revert TickUpperOutOfBounds(tickUpper);
    }
}
```

This validation should be called after any tick alignment operations to ensure the resulting ticks are valid. The validation should be added to both `alignTickToTickSpacing` and any functions that use its results, particularly `calculateLogNormalDistribution`.

Zora: Acknowledged. Pending remediation.

Cantina Managed: Acknowledged.

3.2.5 Expected address prefix of `0x4444` for `ZoraV4CoinHook` is ignored

Severity: Low Risk

Context: HooksDeployment.sol#L14-L38, HooksDeployment.sol#L46-L48

Summary: The expected address prefix of `0x4444` for `ZoraV4CoinHook` is ignored in `HookMinerWithCreationCodeArgs.find()`.

Finding Description: `HooksDeployment.deployZoraV4CoinHook()` sets `flags = uint160(Hooks.AFTER_SWAP_FLAG | Hooks.AFTER_INITIALIZE_FLAG) ^ (0x4444 << 144);` while mining the hook address so that the mined address not only has the appropriate lower-order bits corresponding to the hooks enabled in `ZoraV4CoinHook` but also an expected address prefix of `0x4444`. This is apparently to help give `ZoraV4CoinHook` a unique vanity address with lower probability of collision with other deployed hooks.

However, when this `flags` is used as a parameter to `HookMinerWithCreationCodeArgs.find()`, that function overwrites the `flags` parameter to `flags = flags & FLAG_MASK` (where `FLAG_MASK = Hooks.ALL_HOOK_MASK; // 0000 ... 0000 0011 1111 1111 1111`) which zeroes out the desired address prefix of `0x4444` in the top-order bits while only retaining the bottom 14 bits. This effectively ignores the desired address prefix.

Impact Explanation: Very low, because the distinct address prefix is expected to only help give `ZoraV4CoinHook` a unique vanity address with lower probability of collision with other deployed hooks.

Likelihood Explanation: High, because the address prefix is always ignored during deployment.

Recommendation: Consider:

1. Changing the flag mask values and logic to get the expected `0x4444` prefixed hook address.
2. Evaluate if `MAX_LOOP = 160_444` is sufficient number of iterations to get an address that satisfies both the expected prefix and bottom 14 bits of hook flag values.

Zora: Acknowledged. Will work on this as part of a deployment ops improvement in future.

Cantina Managed: Acknowledged.

3.3 Informational

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

Severity: Informational

Context: ZoraFactoryImpl.sol#L42

Description: `ZoraFactoryImpl` uses the `initializer` modifier on constructor to prevent calls to `initialize` on the implementation itself. 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 contracts designed to be called through proxies.

Zora: Fixed in commit 25da59aa.

Cantina Managed: Fix verified.

3.3.2 Compromised coin owner can arbitrarily set `setPayoutRecipient` to claim all future rewards

Severity: Informational

Context: BaseCoin.sol#L42, BaseCoin.sol#L130-L140, MultiOwnable.sol#L102-L118

Summary: Malicious/compromised coin owner can arbitrarily set `setPayoutRecipient` to claim all future rewards.

Finding Description: Coins inherit from `MultiOwnable`, which allows multiple addresses to have owner privileges over the coin. Coin owners can set `setPayoutRecipient` and `setContractURI`. If one of the coin owners is malicious/compromised then they can arbitrarily set `setPayoutRecipient` to claim all future rewards. They can also claim sole ownership by removing all other owners and can resist removal, if detected before that, by front-running (where possible) any attempts to removing their owner privileges.

Recommendation: Document this trust model and warn users if/when they add new owners.

Zora: Acknowledged. This is an intentional design choice on the ownership system - owners have full access and should only be first party trusted wallets for one reviewer.

Cantina Managed: Acknowledged.

3.3.3 Enforcing `hookData.length > 0` in `getTradeReferral()` may cause reverts

Severity: Informational

Context: CoinRewardsV4.sol#L42, CoinRewardsV4.sol#L45-L47

Description: `getTradeReferral()` enforces a `hookData.length > 0` to determine if the Uniswap V4 pool manager sent any trade referrer address in `hookData` to be used in `CoinRewardsV4.distributeMarketRewards()` for sending it `tradeReferrerAmount` rewards.

However, if `hookData.length < 20` for some reason then this will cause `abi.decode(hookData, (address))` to revert the swap instead of ignoring any corrupt `hookData`.

Recommendation: Consider replacing `hookData.length > 0` with `hookData.length >= 20` so that any corrupt `hookData` is effectively ignored.

Zora: Fixed in commit 2215c14b.

Cantina Managed: Fix verified.

3.3.4 Any rewards sent to zero address in `depositRewards()` will remain stuck in `ProtocolRewards`

Severity: Informational

Context: README.md#L13, ProtocolRewards.sol#L99-L146

Description: `ProtocolRewards.depositRewards()` allows Zora ERC-721 & ERC-1155 contracts to deposit protocol rewards for `creator`, `createReferral`, `mintReferral`, `firstMinter` and `zora`. It increases their balances only if they are non-zero addresses. Any rewards sent to zero addresses for these roles will remain in `ProtocolRewards` without any function to withdraw/rescue them.

This is acknowledged in the README as follows:

> *The `ProtocolRewards` contract has an implementation caveat. If you send any value to a zero (`address(0)`) address in `depositRewards`, that value is implicitly burned by being locked in the contract at the zero address. The function will not revert or redirect those funds as currently designed. We may re-visit this design in the future but for the release of v1.1 this is the current and expected behavior.*

Recommendation: Consider revisiting this design to prevent stuck rewards.

Zora: Acknowledged. This is a known issue with this contract and is documented. In the future we may want to fix this and move to a new contract but many application and third party integrations rely on this specific contract.

Cantina Managed: Acknowledged.

3.3.5 Duplicated `POOL_LAUNCH_SUPPLY` constant may lead to inconsistent usage

Severity: Informational

Context: CoinConstants.sol#L7-L11, CoinDopplerMultiCurve.sol#L106, CoinDopplerMultiCurve.sol#L123, MarketConstants.sol#L7

Description: Both `CoinConstants` and `MarketConstants` define `constant POOL_LAUNCH_SUPPLY = 990_000_000e18`, which is the launch supply of coin pools. Only `MarketConstants.POOL_LAUNCH_SUPPLY` is used in `CoinDopplerMultiCurve.sol`. However, `CoinConstants.MAX_TOTAL_SUPPLY` is used elsewhere. If either of these `POOL_LAUNCH_SUPPLY` constants is changed in future and get used in different places then that inconsistency would lead to issues.

Recommendation: Consider retaining only `MarketConstants.POOL_LAUNCH_SUPPLY` and removing the unused `CoinConstants.POOL_LAUNCH_SUPPLY`.

Zora: Fixed in commit 498e5c9d.

Cantina Managed: Fix verified.

3.3.6 Missing ability to update `trustedMessageSender` may force upgrade

Severity: Informational

Context: ZoraV4CoinHook.sol#L31-L46, ZoraV4CoinHook.sol#L133-L137, ZoraV4CoinHook.sol#L153-L155

Description: `ZoraV4CoinHook` has a notion of `trustedMessageSender` to track if routers triggering coin swaps are "trusted" by the protocol or not. This is used to attribute swaps at the indexing layer via the `Swapped` event emission.

However, `trustedMessageSender` is initialized during `ZoraV4CoinHook` deployment and there is no ability to update this list via an authorized function/registry. If one of the routers is deemed untrustworthy later or another trusted router needs to be added after deployment then a new hook that supports this functionality has to be deployed.

Recommendation: Consider adding this missing functionality to update `trustedMessageSender` to avoid future upgrades.

Zora: Acknowledged.

Cantina Managed: Acknowledged.

3.3.7 Unused code reduces readability

Severity: Informational

Context: CoinConfigurationVersions.sol#L50-L56, MarketConstants.sol#L24-L28, V4Liquidity.sol#L37-L39, V4Liquidity.sol#L45, V4Liquidity.sol#L64, V4Liquidity.sol#L132-L139

Description: There are instances of unused logic, return values and declared errors, which are never used and therefore reduce readability.

Recommendation: Consider removing all such instances of unused code constructs.

Zora: Fixed in commit 8b85ab94.

Cantina Managed: Fix verified.

3.3.8 Insufficient owner count validation in ownership management

Severity: Informational

Context: MultiOwnable.sol#L120-L125

Description: The `MultiOwnable` contract's `revokeOwnership` function lacks validation to ensure that at least one owner remains in the system after the operation. This could lead to a situation where all owners are removed, effectively locking the contract's owner-only functions and potentially causing permanent loss of administrative control.

Recommendation: Add a validation check in the `revokeOwnership` function to ensure that at least one owner remains after the operation. This can be implemented by checking the size of the `_owners` set before removing an owner. If the removal would result in zero owners, the operation should revert.

Zora: Acknowledged. This is intentional because we wish for users to be able to fully revoke ownership.

Cantina Managed: Acknowledged.

3.3.9 Inconsistent pool key hashing implementation

Severity: Informational

Context: CoinCommon.sol#L12-L14

Description: The `CoinCommon` library's `hashPoolKey` function implements a custom hashing mechanism for pool keys instead of using the standardized `PoolId.toId` function from Uniswap V4's `PoolIdLibrary`.

This inconsistency could lead to different pool key hashes being generated across the protocol, potentially causing integration issues with Uniswap V4's core functionality.

Recommendation: Replace the custom `hashPoolKey` implementation with Uniswap V4's `PoolId.toId` function to ensure consistency with the core protocol. This will maintain compatibility with Uniswap V4's standard pool identification system and prevent potential issues with future protocol updates.

Zora: Acknowledged.

Cantina Managed: Acknowledged.

3.3.10 Inconsistent version validation in pool setup

Severity: Informational

Context: CoinSetup.sol#L43-L44

Description: The `setupPoolWithVersion` function in the codebase lacks proper version validation by not checking for `LEGACY_POOL_VERSION`. The current implementation uses direct version comparisons instead of leveraging the existing `CoinConfigurationVersions` library functions like `isV3` and `isV4`. This inconsistency could lead to maintenance issues and potential bugs if version checks are updated in one place but not others.

Recommendation: Update the `setupPoolWithVersion` function to use the `CoinConfigurationVersions` library functions for version validation. This will ensure consistency across the codebase and make future updates easier. The function should check for all supported versions, including `LEGACY_POOL_VERSION`, using the appropriate library functions.

Zora: Fixed in commit dac72691.

Cantina Managed: Only the `DOPPLER_MULTICURVE_UNI_V4_POOL_VERSION` version is being checked, other versions are considered as invalid/deprecated.