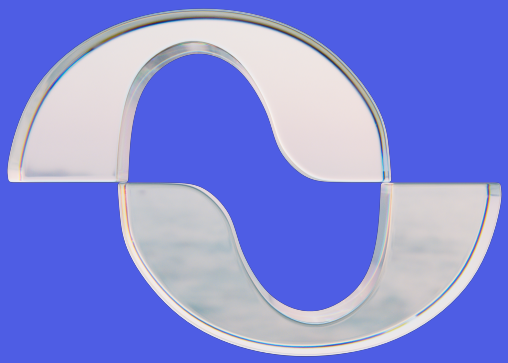
OpenZeppelin | security

ZKC Audit



September 9, 2025

Table of Contents

Summary

Type	DeFi
Timeline	From 2025-08-19 To 2025-08-28
Languages	Solidity

Total Issues	27 (20 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	4 (2 resolved)
Notes & Additional Information	23 (18 resolved)

Scope

OpenZeppelin audited the boundless-xyz/zkc repository at commit f6f8f89.

In scope were the following files:

```
zkc
├── script
│   └── PrecomputeSupply.s.sol
└── src
    ├── ZKC.sol
    ├── veZKC.sol
    ├── components
    │   ├── Clock.sol
    │   ├── Rewards.sol
    │   ├── Staking.sol
    │   ├── Storage.sol
    │   └── Votes.sol
    ├── interfaces
    │   ├── IRewards.sol
    │   ├── IStaking.sol
    │   ├── IVotes.sol
    │   └── IZKC.sol
    ├── libraries
    │   ├── Checkpoints.sol
    │   ├── Constants.sol
    │   ├── RewardPower.sol
    │   ├── StakeManager.sol
    │   ├── Supply.sol
    │   └── VotingPower.sol
    └── rewards
        └── StakingRewards.sol
```

System Overview

The system is composed of three primary components: the ZKC token, the `veZKC` contract, and the `StakingRewards` contract. Together, these elements implement a vote-escrow model that governs token emissions, staking mechanics, and reward distribution, with delegation support.

The **ZKC token** is an upgradeable ERC-20 contract that serves as the foundational asset. Its supply expands indefinitely according to a predefined emission schedule, ensuring a continuous flow of tokens into the system. These emissions fund both staking rewards and rewards for provers submitting Proofs of Verifiable Work (PoVW).

The **`veZKC` contract** is the core governance mechanism. Users lock their ZKC tokens into this contract and, in return, receive a non-transferable NFT that represents their vote-escrowed position. The amount of voting power a user holds is determined by the quantity of ZKC locked. The contract supports vote delegation, reward delegation, and enforces a 30-day waiting period before locked tokens can be withdrawn. During the waiting period, no voting/reward power is assigned to the stake, and all delegations must be cancelled prior to initiating a withdrawal.

The **`StakingRewards` contract** manages the distribution of staking rewards generated by the token emission schedule. A user's rewards are proportional to their reward power, which is calculated based on the user's active stake during each epoch. This design ensures that rewards are fairly allocated to participants according to their level of contribution and time committed to the protocol.

Together, these components establish a governance and incentive framework that ties voting influence and economic benefits to long-term participation in the protocol.

Security Model and Trust Assumptions

During the audit, the following trust assumptions were made:

- The `totalSupply` function of the ZKC token returns the theoretical epoch-based supply and is not ERC-20-compliant. The exact total supply is returned by the `claimedTotalSupply` function.
- The `ADMIN_ROLE` (synonymous with the `DEFAULT_ADMIN_ROLE`) of the `ZKC`, `veZKC`, and `StakingRewards` contracts is authorized to perform contract upgrades, which can change the currently deployed logic of the system.
- The `POVW_MINTER_ROLE` and `STAKING_MINTER_ROLE` can mint predefined amounts of ZKC tokens, and these roles must not be assigned to entities capable of minting arbitrarily.
- The initial supply of ZKC tokens is minted by two designated `initialMinter` addresses, which are assumed to distribute tokens in accordance with the stated tokenomics.

Supply Script Review

The `scripts/PrecomputeSupply.s.sol` script was reviewed for correctness as part of this engagement. The script was confirmed to not contain any vulnerabilities and to compute values that maintain all required invariants. Specifically:

- The supply is an increasing function of the epoch.
- The supply is continuous at the boundaries.
- The forward error in the growth factors is within an acceptable error tolerance.

The method used to determine the per-epoch growth factors is impacted by the errors in the algorithms used by the PRBMath library within the `ln` and `exp` functions. It was found that by using an optimization-based approach that minimizes errors, such as a root-finding algorithm like the bisection method or Newton's method, the forward error in the resulting values could be decreased by two orders of magnitude. This reduction in error does not have a

material impact on the growth of the supply of the ZKC token as the errors in the current script are on the order of `1e-14`, which are too small to impact the supply.

Low Severity

L-01 Missing Access Control on `initialize` Functions

The `initializeV2` function is intended to be called after upgrading the ZKC token to set `epoch0StartTime`. However, it currently lacks an access-control modifier, allowing anyone to call it.

Consider adding the appropriate access-control modifiers to ensure that only authorized accounts can call `initialize` functions.

Update: *Resolved in pull request #26 and pull request #30.*

L-02 Lack of Proportional Rewards Based on Staking Duration Within Epochs

The staking mechanism allows users to stake without a lock and withdraw at any time, with withdrawals requiring a 30-day processing period. A user staking from the start of an epoch should ideally earn more voting and reward power than one who stakes at the very end. However, the current implementation does not differentiate between users staking at the beginning of an epoch and those staking just before it ends. This results in equal rewards for unequal participation, incentivizing users to stake only at the end of the epoch.

Consider adjusting the staking mechanism to account for the duration of participation within an epoch so that rewards and voting power are proportional to the actual time staked.

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

> This is a known issue and addressing it would add extra complexity. We believe that this issue is mitigated by withdrawal period.

L-03 Checkpoint Underflow Can Inflate Voting and Reward Power

In `Checkpoints.sol`, functions update voting and reward amounts using the `uint256(int256(lastValue) + Delta)` pattern. Ideally, if `Delta` is positive, the value should increase, and if it is negative, the value should decrease without ever dropping below zero. However, when `Delta` is negative and its magnitude exceeds `lastValue`, the intermediate result becomes negative. In Solidity, casting a negative `int256` to `uint256` does not revert but instead wraps the value into a very large number. This can inflate voting or reward amounts to unrealistic levels, breaking protocol invariants.

Consider adding an underflow check before updating the value.

Update: *Resolved in pull request #20.*

L-04 Inaccurate `EPOCHS_PER_YEAR`

Within the `Supply` library, the value of `EPOCHS_PER_YEAR` is set to 182, with a duration of 2 days per epoch. This implies a calendar year of 364 days, which deviates from the true value of 365.25 by 0.35%. As a result, the growth rates which had been calculated using the number of epochs as well as the starting epoch for the year will deviate from their expected values. Over the course of 8 years, the supply schedule will have been accelerated by 5 epochs in total, increasing the supply by an additional 0.1% from its expected value at the 3% annual growth rate.

Consider defining the `EPOCHS_PER_YEAR` variable as a `UD60x18` type to represent a finer level of precision, with minimal changes to supply calculations.

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

> We do not plan to fix this issue. There is no requirement to match the true value of a year.

Notes & Additional Information

N-01 Ineffective Use of Named Return Parameters in `checkpointWithDelegation`

Within the `Checkpoints library`, the `checkpointWithDelegation` function defines named return parameters `userVotingDelta` and `userRewardDelta` and assigns them values within the function body. In addition, within the `StakingRewards` contract, the `_claim` function includes the `amount` named return parameter. However, in both functions, the final return statement explicitly overrides these values, making the earlier assignments redundant.

Consider removing the named return parameters and instead defining `userVotingDelta`, `userRewardDelta`, and `amount` as local variables to simplify the code and avoid confusion.

Update: *Resolved in pull requests #19, #30, and #5.*

N-02 Redundant Check in `validateUnstakeCompletion`

The `validateUnstakeCompletion` function validates the conditions required to complete an unstake by checking whether a withdrawal has been initiated and whether the withdrawal period is complete. However, it explicitly calls `isWithdrawing(stake)` before calling `canCompleteWithdrawal(stake)`, even though `canCompleteWithdrawal` already calls `isWithdrawing` internally. This makes the standalone `isWithdrawing` check redundant.

Consider removing the explicit `isWithdrawing` check and relying on `canCompleteWithdrawal` for this validation to simplify the logic.

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

> We are leaving this as a defensive check, and also in case of future modifications.

N-03 Duplicate Event Emission

Within the `Stake` contract, the `StakeAdded` event is intended to signal when a new stake is created, while the `UnstakeInitiated` event is intended to signal when an unstake process has begun. However, `StakeAdded` is emitted by both `_addToStake` and `_addStakeAndCheckpoint`, and `UnstakeInitiated` is emitted by both `initiateUnstake` and `_initiateUnstakeAndCheckpoint`, leading to duplicate event emissions.

Consider ensuring that each event is emitted only once per logical action to avoid redundancy and potential confusion for off-chain consumers.

Update: *Resolved in pull request #14.*

N-04 Unused Constant

The `Y8_R_PER_EPOCH` constant in `Supply.sol` was intended to represent the per-epoch growth factor for year 8. However, it is never used in the contract, since `FINAL_R_PER_EPOCH` is referenced instead.

Consider removing the unused `Y8_R_PER_EPOCH` constant to improve code clarity and maintainability.

Update: *Resolved in pull request #16.*

N-05 Redundant Scaling Logic in `Supply.sol`

The `Supply.sol` library converts numbers to the `UD60x18` type by using the `ud(x * SCALE)` expression, where `SCALE = 1e18`. This was intended to scale integers into fixed-point format. However, this approach is redundant since the `UD60x18` library already provides a `convert` function that performs the same operation. Moreover, the `SCALE` constant in `Supply.sol` duplicates the functionality of the `uUNIT` constant defined in `UD60x18`.

Consider replacing `ud(x * SCALE)` with `convert(x)` and removing the redundant `SCALE` constant to simplify the code and improve maintainability.

Update: *Resolved in pull request #22.*

N-06 Redundant Conditional Logic in Delegation Checkpoints

Throughout `Checkpoints.sol`, the ternary operation is used to handle the case when `userEpoch` is 0, creating a new `Point` struct with zero values and setting `updatedAt` to `block.timestamp`. However, this conditional expression is unnecessary because at index 0, all values of the `Point` struct are already 0, and `updatedAt` is never used.

Consider simplifying the code by directly assigning the `Point` from `userStorage.userPointHistory[account][userEpoch]`.

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

> We are leaving this as it is to maintain a record of when the first update took place.

N-07 `supportsInterface` Does Not Acknowledge `IStaking`

Within `Staking.sol`, in line 42:

The contract inherits `IStaking`, yet its ERC-165 implementation only reports support for `IERC721` and whatever the parent contracts provide:

```
        return interfaceId == type(IERC721).interfaceId ||
super.supportsInterface(interfaceId);
```

Since the interface identifier of `IStaking` differs from `IERC721`, a caller querying `supportsInterface(type(IStaking).interfaceId)` will incorrectly receive `false`. This violates ERC-165 expectations and can break integrations that rely on interface detection to interact with the staking contract.

Consider also returning `true` when `interfaceId == type(IStaking).interfaceId`.

Update: *Resolved in pull request #21.*

N-08 Unused State Variables

Throughout the codebase, multiple instances of unused state variables were identified:

- In `Storage.sol`, the `_ownedTokens` state variable

- In `Storage.sol`, the `_ownedTokensIndex` state variable

To improve the overall clarity and intent of the codebase, consider removing any unused state variables.

Update: *Resolved in pull request #16.*

N-09 Multiple Storage Variables Could Be Cached on the Stack

In the EVM, reading from **storage** is much more expensive than reading from the **stack**. If a storage variable is accessed multiple times (for example, inside a loop), it is more gas-efficient to read it once, store it in a temporary stack variable, and then reuse that cached value.

In `StakingRewards.sol`, multiple instances where this optimization can be applied were identified:

- `veZKC`
- `veZKC`
- `zkc`

By caching these storage variables into local (stack) variables before reusing them in loop, the contract will avoid repeated storage lookups, reducing gas costs and improving efficiency.

Update: *Resolved in pull request #21.*

N-10 Redundant Division by One in `Supply.sol`

The `Supply.sol` library performs divisions by `ud(SCALE)`, where `SCALE = 1e18`. This was intended to normalize values when working with the `UD60x18` fixed-point type. However, since `ud(SCALE)` evaluates to 1, these divisions have no effect and only add unnecessary complexity to the code.

Consider removing the divisions by `ud(SCALE)` to simplify calculations and improve code readability.

Update: *Resolved in pull request #22.*

N-11 Redundant `ADMIN_ROLE`

Within the `ZKC`, `veZKC`, and `StakingRewards` contracts, the `ADMIN_ROLE` is simply set to the `DEFAULT_ADMIN_ROLE` value, with the same access-control permissions as `DEFAULT_ADMIN_ROLE`.

Consider removing this redundant role and directly using the `DEFAULT_ADMIN_ROLE` instead.

Update: *Resolved in pull request #16.*

N-12 Variables Initialized With Their Default Values

Throughout the codebase, multiple instances of variables being initialized with their default values were identified:

- In `ZKC.sol`, the `minted` variable
- In `ZKC.sol`, the `i` variable
- In `Staking.sol`, the `withdrawableAt` variable
- In `Checkpoints.sol`, the `min` variable
- In `Checkpoints.sol`, the `min` variable
- In `StakingRewards.sol`, the `i` variable
- In `StakingRewards.sol`, the `i` variable

To avoid wasting gas, consider not initializing variables with their default values.

Update: *Resolved in pull request #23.*

N-13 Unaddressed TODO Comments

During development, having well described TODO/Fixme comments will make the process of tracking and solving them easier. However, left unaddressed, these comments might age and important information for the security of the system might be forgotten by the time it is released to production. As such, they should be tracked in the project's issue backlog and resolved before the system is deployed.

Throughout the codebase, multiple instances of TODO/Fixme comments were identified:

- The `TODO` comment in line 44 of `Storage.sol`
- The `TODO` comment in line 12 of `Checkpoints.sol`

- The `TODO` comment in line 23 of `Checkpoints.sol`

Consider removing all instances of TODO/Fixme comments and instead tracking them in the issues backlog. Alternatively, consider linking each inline TODO/Fixme to the corresponding issues backlog entry.

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

> We are leaving these comments as they are in case of future releases/upgrades.

N-14 Prefix Increment Operator (`++i`) Can Save Gas in Loops

Throughout the codebase, multiple instances where the subject optimization can be applied were identified:

- The `i++` in `ZKC.sol`
- The `i++` in `StakingRewards.sol`
- The `i++` in `StakingRewards.sol`

Consider using the prefix increment operator (`++i`) instead of the post-increment operator (`i++`) in order to save gas. This optimization skips storing the value before the incremental operation, as the return value of the expression is ignored.

Update: *Resolved in pull request #23.*

N-15 Missing Named Parameters in Mapping

Since Solidity 0.8.18, mappings can include named parameters to provide more clarity about their purpose. Named parameters allow mappings to be declared in the form `mapping(KeyType KeyName? => ValueType ValueName?)`. This feature enhances code readability and maintainability.

In the `_userClaimed` state variable of the `StakingRewards` contract, the mapping does not have any named parameters.

Consider adding named parameters to mappings in order to improve the readability and maintainability of the codebase.

Update: *Resolved in pull request #24.*

N-16 Variables Could Be `constant`

If a variable is only ever assigned a value when it is declared, then it could be declared as `constant`.

Within `ZKC.sol`, multiple variables that could be `constant` were identified:

- The `POVW_MINTER_ROLE` state variable
- The `STAKING_MINTER_ROLE` state variable

To better convey the intended use of variables and to potentially save gas, consider adding the `constant` keyword to variables that are only set when they are declared.

Update: *Resolved in pull request #24.*

N-17 Unused Functions

Throughout the various libraries in the codebase, multiple instances of unused functions were identified:

- The `emptyStake` function of the `StakeManager` library
- The `getUserPoint` function of the `Checkpoints` library
- The `getGlobalPoint` function of the `Checkpoints` library
- The `getUserEpoch` function of the `Checkpoints` library
- The `getGlobalEpoch` function of the `Checkpoints` library
- The `checkpoint` function of the `Checkpoints` library
- The `getGrowthFactor` function of the `Supply` library
- The `getYearForEpoch` function of the `Supply` library
- The `getTotalSupply` function of the `VotingPower` library

Consider removing any unused functions or incorporating them into the codebase to improve code clarity and maintainability.

Update: *Resolved in pull request #16 and pull request #30.*

N-18 Missing Security Contact

Providing a specific security contact (such as an email address or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to

dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, multiple instances of contracts not having a security contact were identified:

- The `ZKC` contract
- The `StakingRewards` contract
- The `veZKC` contract

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

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

> We intend to use other methods for sharing a security contact.

N-19 Floating Pragma

Throughout the codebase, pragma directives are specified using floating versions (e.g., `^0.8.20`).
This can lead to compilation inconsistencies if newer compiler versions introduce changes or unexpected behavior.

Consider using fixed pragma directives to ensure deterministic and reproducible builds across all environments.

Update: *Resolved in pull request #24.*

N-20 Unnecessary Withdrawal Check in `updateGlobalCheckpoint`

The `updateGlobalCheckpoint` function calculates effective amounts for old and new stakes, setting them to zero if `withdrawalRequestedAt > 0`. This was intended to exclude stakes that are in the process of being withdrawn. However, the function is only ever called when staking or adding to an existing stake, meaning that `withdrawalRequestedAt` will

not be relevant in this context. As a result, the conditional check is unnecessary and adds redundant logic.

Consider removing the `withdrawalRequestedAt > 0` check when calculating `oldEffectiveAmount` and `newEffectiveAmount` to simplify the function.

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

> We are leaving this as a defensive check, and also in case of future modifications.

N-21 Missing Docstrings

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

- In `ZKC.sol`, the `initialize` function
- In `Clock.sol`, the `CLOCK_MODE` function
- In `Staking.sol`, the `supportsInterface` function
- In `Votes.sol`, the `getVotes` function
- In `Votes.sol`, the `getPastVotes` function
- In `Votes.sol`, the `getPastTotalSupply` function
- In `Votes.sol`, the `delegates` function
- In `Votes.sol`, the `delegate` function
- In `Votes.sol`, the `delegateBySig` function
- In `IRewards.sol`, the `RewardDelegateChanged` event
- In `IStaking.sol`, the `StakeCreated` event
- In `IStaking.sol`, the `StakeAdded` event
- In `IStaking.sol`, the `StakeBurned` event
- In `IStaking.sol`, the `UnstakeInitiated` event
- In `IStaking.sol`, the `UnstakeCompleted` event
- In `StakingRewards.sol`, the `initialize` function
- In `veZKC.sol`, the `initialize` function

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

Update: *Resolved in pull request #25.*

N-22 Incomplete Docstrings

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

- In `ZKC.sol`, within the `totalSupply` function, not all return values are documented.
- In `Storage.sol`, within the `nonces` function, the `owner` parameter and multiple return values are not documented.
- In `IZKC.sol`, within the `PoVWRewardsClaimed` event, the `recipient` and `amount` parameters are not documented.
- In `IZKC.sol`, within the `StakingRewardsClaimed` event, the `recipient` and `amount` parameters are not documented.
- In `veZKC.sol`, within the `supportsInterface` function, the `interfaceId` parameter and multiple return values are not documented.

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of a contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #25 and pull request #30.*

N-23 Missing Error Messages in Require Statements

It is recommended to provide informative error messages in `require` statements. This practice improves code clarity and assists with troubleshooting when a requirement is not satisfied.

Within `ZKC.sol`, multiple `require` statements with missing error messages were identified:

- The `require` statement in line 88
- The `require` statement in line 104
- The `require` statement in line 105

To improve the clarity and maintainability of the code, consider adding informative error messages to all `require` statements.

Update: *Resolved in pull request #25.*

Conclusion

The Boundless protocol features an architecture that incentivizes staking through rewards and governance participation, along with the distribution of rewards to provers who generate Proofs of Verifiable Work. Multiple fixes and improvements were suggested to enhance the clarity of the codebase. Overall, the codebase was found to be well-structured with robust mechanisms.

The RISC Zero team is commended for their professionalism throughout the engagement, including their detailed protocol walkthrough and responsiveness to questions, which enabled the audit team to conduct a thorough assessment.

Appendix

Issue Classification

OpenZeppelin classifies smart contract vulnerabilities on a 5-level scale:

- Critical
- High
- Medium
- Low
- Note/Information

Critical Severity

This classification is applied when the issue's impact is catastrophic, threatening extensive damage to the client's reputation and/or causing severe financial loss to the client or users. The likelihood of exploitation can be high, warranting a swift response. Critical issues typically involve significant risks such as the permanent loss or locking of a large volume of users' sensitive assets or the failure of core system functionalities without viable mitigations. These issues demand immediate attention due to their potential to compromise system integrity or user trust significantly.

High Severity

These issues are characterized by the potential to substantially impact the client's reputation and/or result in considerable financial losses. The likelihood of exploitation is significant, warranting a swift response. Such issues might include temporary loss or locking of a significant number of users' sensitive assets or disruptions to critical system functionalities, albeit with potential, yet limited, mitigations available. The emphasis is on the significant but not always catastrophic effects on system operation or asset security, necessitating prompt and effective remediation.

Medium Severity

Issues classified as being of medium severity can lead to a noticeable negative impact on the client's reputation and/or moderate financial losses. Such issues, if left unattended, have a moderate likelihood of being exploited or may cause unwanted side effects in the system.

These issues are typically confined to a smaller subset of users' sensitive assets or might involve deviations from the specified system design that, while not directly financial in nature, compromise system integrity or user experience. The focus here is on issues that pose a real but contained risk, warranting timely attention to prevent escalation.

Low Severity

Low-severity issues are those that have a low impact on the client's operations and/or reputation. These issues may represent minor risks or inefficiencies to the client's specific business model. They are identified as areas for improvement that, while not urgent, could enhance the security and quality of the codebase if addressed.

Notes & Additional Information Severity

This category is reserved for issues that, despite having a minimal impact, are still important to resolve. Addressing these issues contributes to the overall security posture and code quality improvement but does not require immediate action. It reflects a commitment to maintaining high standards and continuous improvement, even in areas that do not pose immediate risks.