Security Audit Report

ICN Protocol

v1.0

May 2, 2025

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by Impossible Cloud Network Foundation to perform a security audit of ICN Protocol (HyperNodes release) Smart Contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/ICN-Protocol/icn-protocol
Commit	`3c01a79f0632053979aeda2d690821b4719b3433`
Scope	The scope is restricted to the following contracts: • Proxy • AccessControl • ICNRegistry ○ `registerHyperNode()`

	◦ `getHyperNode()` ● LinkStaking ● LinkRewards ● ExternalContractManager
Fixes verified at commit	`aa970308591b8706bd7a66cf0bc4b5c7732ed74d` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The ICN protocol is a modular smart contract that enables decentralized registration, staking, and reward distribution for network participants such as Hyper Nodes.

It uses a diamond proxy architecture to manage modules like access control, node registration, staking of ICNL tokens, and reward calculation in ICNT tokens.

Users stake non-transferable ICNL NFT tokens to support nodes and earn time-based rewards defined by a fixed emission curve.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending, Acknowledged**, **Partially Resolved**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**Low-Medium**	-
Code readability and clarity	**Medium-High**	-
Level of documentation	**Medium-High**	The client provided documentation and diagrams of the protocol.
Test coverage	**Medium**	`forge coverage` reports a test coverage of `72.89%`.

Summary of Findings

No	Description	Severity	Status
1	Missing slashing impact handling in LinkRewards contract gives no disincentive to misbehave after staking NFTs	Major	Acknowledged
2	HyperNodes cannot be removed or deactivated once registered	Minor	Acknowledged
3	No upper bound validation for staking and unstaking periods	Minor	Resolved
4	Incomplete module removal leaves stale selector hash in Proxy storage	Minor	Resolved
5	Fixed reward allocation discrepancy	Minor	Resolved
6	Incorrect deadline handling in rewards calculation	Minor	Resolved
7	Partial input validation for HyperNodes registration and update	Minor	Resolved
8	Inconsistent era notification in multiple event emissions	Minor	Resolved
9	Immediate modification of `minLinkStakingPeriod` by the admin may unfairly extend lock duration for existing stakes	Minor	Resolved
10	Incorrect annual duration constant may cause reward and delegation schedule miscalculations	Minor	Acknowledged
11	Assumptions in NFT staking logic allow potential misuse under transfer-enabled scenarios	Minor	Acknowledged
12	Unsafe cast from `uint256` to `uint32` may limit specific NFT ID staking	Minor	Resolved
13	Retroactive rewards curve updates allow overcompensation for unclaimed staking periods	Minor	Resolved
14	Centralization risks	Minor	Acknowledged
15	Interface documentation inconsistencies may mislead about reward eligibility and node types	Informational	Resolved
16	Redundant contract imports across multiple files	Informational	Resolved

17	Missing public getters for external contract addresses	Informational	Resolved
18	Diamond proxy implementation lacks EIP-2535 compliance	Informational	Acknowledged
19	Contracts should implement a two-step ownership transfer	Informational	Resolved
20	Inconsistent initialization event pattern in the LinkStaking contract	Informational	Resolved
21	No maximum limit on reward curve array size	Informational	Resolved
22	Possible rewards denial-of-service in case that claimed rewards exceed total rewards	Informational	Resolved
23	The `getHyperNode` view function reverts unexpectedly	Informational	Resolved
24	Miscellaneous comments	Informational	Resolved

Detailed Findings

1. Missing slashing impact handling in LinkRewards contract gives no disincentive to misbehave after staking NFTs

Severity: Major

The LinkRewards contract defined in `src/modules/LinkRewards/LinkRewards.sol` is responsible for tracking and distributing rewards to users staking their tokens in HyperNodes.

However, the contract lacks logic to account for slashing events, which can occur due to misbehavior or protocol-defined penalties.

Specifically, the ICN protocol introduces a slashing mechanism in the form of diminishing the validity period of an NFT, whose owner performed a slashable offence. Current NFT duration is verified during staking by setting the `durationTimes` array for all staked linkIds in `src/modules/LinkStaking/LinkStaking.sol:363-372` in `_validateLink` function.

However, if slashing occurs after staking, it is not reflected in reward computations. As a result, stakers continue receiving full-duration rewards despite diminished stake validity, effectively nullifying slashing as a deterrent and allowing malicious actors to unjustly claim staking incentives.

Without integrating slashing impact, reward calculations may remain inflated or inaccurate, distributing unearned tokens to stakers despite the reduced effective stake or validator performance penalties.

Recommendation

We recommend integrating slashing event awareness into the reward calculation logic by accounting for the diminished token remaining duration.

Checking for slashing events after staking introduces risk: if the staked NFT is burned, subsequent duration checks may result in a denial of service for both reward accrual and unstaking due to reliance on `_requireOwned` in duration-related functions of ICNLink.

One mitigation approach is to verify NFT ownership before checking for an updated duration. If the user remains the owner, update the duration; otherwise, treat the NFT as expired.

This adjustment would complicate reward logic, as durations are sorted independently and their indices do not correspond 1:1 with linkId, necessitating broader changes to the reward handling mechanism.

Status: Acknowledged

The client acknowledges the issue:

"Slashing impact is primarily deferred to the end of the 4-year TTL reward curve. During the bootstrapping phase of the project we don't foresee heavy slashing events that will cause an implication to near term reward calculations. Later stage of the protocol will introduce a feature to address this."

2. HyperNodes cannot be removed or deactivated once registered

Severity: Minor

The ICNRegistry contract, defined in `src/modules/ICNRegistry/ICNRegistry.sol`, currently lacks the functionality to remove or deactivate registered HyperNodes.

This limitation complicates the lifecycle management of nodes within the network, potentially leading to an accumulation of non-functional or compromised nodes over time.

Furthermore, the contract does not provide a clear mechanism for stakers to retrieve their NFTs in a timely manner after a HyperNode ceases operation. This lack of transparency regarding hypernode status and associated NFT management can create confusion and a suboptimal experience for stakers.

Recommendation

We recommend implementing functionality to allow for the removal or deactivation of HyperNodes.

Additionally, we recommend developing a mechanism for stakers to retrieve their NFTs promptly when a HyperNode stops being active, along with a way for stakers to easily ascertain the current operational status of HyperNodes they have staked with.

Status: Acknowledged

The client acknowledges the issue:

"During this bootstrap phase of the project, HyperNodes will remain activated to offer stability during the most volatile periods. The next phase of the project will introduce deeper lifecycle management."

3. No upper bound validation for staking and unstaking periods

Severity: Minor

In `src/modules/LinkStaking/LinkStaking.sol:61-77` and `100-114`, the functions `initializeLinkStaking`, `setMinLinkStakingPeriod`, and `setLinkUnstakingPeriod` validate that input periods are greater than zero.

However, they do not enforce any upper bound checks.

Consequently, administrators could set extremely long periods for `minLinkStakingPeriod` and `linkUnstakingPeriod`, potentially locking user funds for excessive periods or making the protocol practically unusable.

Recommendation

We recommend implementing reasonable upper bounds for both parameters.

Consider adding maximum duration checks like `require(minLinkStakingPeriod <= MAX_STAKING_PERIOD)` and `require(linkUnstakingPeriod <= MAX_UNSTAKING_PERIOD)`, with constants defined based on protocol requirements.

Status: Resolved

4. Incomplete module removal leaves stale selector hash in Proxy storage

Severity: Minor

In `src/Proxy/Proxy.sol:93-105`, the `removeModule` function permits the contract admin to remove a module implementation from the proxy.

However, the function does not clear the associated `selectorsHash` for the removed implementation.

This oversight leaves stale data in storage, potentially causing inconsistencies and complicating state management due to assumptions based on outdated selector mappings.

Recommendation

We recommend updating the `removeModule` function to explicitly reset the `selectorsHash` associated with the removed module implementation.

Status: Resolved

5. Fixed reward allocation discrepancy

Severity: Minor

In `src/common/ProtocolConstants.sol:59-60`, the `INITIAL_REWARD_DISTRIBUTION` constant is calculated as `0.15 * 140,000,000 / 55,000`, implementing a `15%` allocation of the total rewards pool for fixed rewards.

This contradicts the architecture documentation, which states:

"`20%` of the total `140,000,000` [NFT Rewards Pool] can be paid through a special function at any time without staking and without waiting period."

Recommendation

We recommend updating either the implementation or the documentation to ensure consistency.

If `20%` is the intended allocation, the constant should be set as `0.20 * 140,000,000 / 55,000`.

Status: Resolved

6. Incorrect deadline handling in rewards calculation

Severity: Minor

In `src/modules/LinkRewards/LinkRewards.sol:179`, the function uses <= to check if the current timestamp has reached the expiration time.

This implementation prevents users from claiming rewards at the exact moment of expiration. Best practices (as established in standards like EIP-2612) dictate that operations at the exact deadline timestamp should be valid.

Recommendation

We recommend replacing the comparison operator from <= to < to allow reward calculations at the exact expiration time.

Status: Resolved

7. Partial input validation for HyperNodes registration and update

Severity: Minor

The `registerHyperNode` function in `src/modules/ICNRegistry/ICNRegistry.sol:415-431` allows entities with the `ICN_OPERATOR_ROLE` to register new hyper nodes by submitting details such as operator address, public key and location code.

However, the function lacks sufficient input validation. Specifically, the function does not enforce uniqueness checks on the `publicKey`, creating a risk of registering multiple nodes with identical or conflicting identifiers.

Recommendation

We recommend enhancing input validation in the `registerHyperNode` function.

Additionally, we recommend enforcing constraints on the uniqueness of `publicKey` values across all registered hyper nodes to prevent duplication.

Status: Resolved

8. Inconsistent era notification in multiple event emissions

Severity: Minor

In `src/modules/ICNRegistry/ICNRegistry.sol`, five functions emit events with era information that is inconsistent with the client's stated behavior:

- `updateReleaseSchedule`
- `updateClusterMaxPrice`
- `verifyScalerNode`
- `removeScalerNode`

Each function emits an event that includes `_getEraManagerCurrentEra() + 1`, indicating changes take effect in the next era (`E+1`).

However, according to the client:

"Era related changes happening during the current Era '`E`' will begin to be applied in the Protocol in the Era '`E+2`'."

This discrepancy could cause integration issues for systems or users monitoring these events, as they would incorrectly assume changes become active in era `E+1` rather than `E+2`.

Recommendation

We recommend updating all affected event emissions to include the correct era when changes take effect, ensuring consistency between event data and actual system behavior.

Status: Resolved

9. Immediate modification of `minLinkStakingPeriod` by the admin may unfairly extend lock duration for existing stakes

Severity: Minor

In `src/modules/LinkStaking/LinkStaking.sol:101-107`, the `DEFAULT_ADMIN_ROLE` possesses the authority to update the `minLinkStakingPeriod`.

However, any changes to this parameter are applied instantly and retroactively affect all existing stakes. This behavior introduces a fairness concern, as stakers may find their tokens locked for longer durations than originally agreed upon.

Such unexpected extensions can undermine user trust and violate the principle of immutability typically expected in staking mechanisms.

Recommendation

We recommend implementing a mechanism that ensures changes to `minLinkStakingPeriod` only apply to new stakes created after the update.

Status: Resolved

10. Incorrect annual duration constant may cause reward and delegation schedule miscalculations

Severity: Minor

In `src/common/ProtocolConstants.sol:10`, the `ONE_YEAR` constant is defined as `12 * ONE_MONTH`.

However, `ONE_MONTH` is statically set to `30` days, resulting in a `ONE_YEAR` value equivalent to `360` days instead of the standard `365`.

This discrepancy leads to inaccuracies in time-dependent protocol logic, specifically impacting hardware provider delegation durations and the release schedules used for capacity-based reward distribution.

Recommendation

We recommend changing `ONE_YEAR` to `365.25 days` to accurately measure years, accounting for leap years.

Status: Acknowledged

The client acknowledges the issue:

"The protocol uses a standard measure of 1 month = 30 days, and the largest timescale is denominated in months. The usage of 1 year is as an alias for simplification of `12 months` where months remain the centrally defined constant of 30 days."

11. Assumptions in NFT staking logic allow potential misuse under transfer-enabled scenarios

Severity: Minor

In `src/modules/LinkStaking/LinkStaking.sol:363-364`, LINK NFTs are verified for ownership at the time of staking. The protocol assumes NFTs remain under the user's control throughout the staking period, as LINK NFT transfers are disabled by default.

However, this design implicitly relies on the immutability of NFT ownership. Since the ICN LINK token contract allows the admin to enable transfers, if it is enabled in the future, it would be possible to flashloan a LINK NFT and stake it multiple times, bypassing intended staking limitations.

Additionally, even with transfers disabled, it is technically feasible to transfer a LINK NFT to the zero address due to the burn logic using similar transfer semantics. This edge case affects the correctness of the `LinkStaking.isLinkStaked` function, potentially leading to inconsistencies in off-chain clients or external smart contracts that depend on this status indicator.

Recommendation

We recommend, for the current stage, that the staking contract verify whether LINK NFT transfers are enabled and explicitly fail if they are.

If transfer functionality is activated in the future, the staking mechanism must be restructured to handle transferable NFTs safely, potentially by transferring them to the INCP contract for escrow to maintain control and prevent misuse.

Status: Acknowledged

The client acknowledges the issue:

"Transfers will remain disabled at the current version of the protocol. Transfers enabled at a later date will occur with a simultaneous feature change to alleviate any unintended staking behaviour as a result."

12. Unsafe cast from `uint256` to `uint32` may limit specific NFT ID staking

Severity: Minor

The protocol employs storage optimization by packing up to eight NFT IDs into a single storage slot using low-level byte manipulation via `ArraysLib.store32`, invoked in `src/modules/LinkStaking/LinkStaking.sol:319`.

This method assumes that each `linkId` fits within 32 bits, but performs an unsafe cast from `uint256` to `uint32` without enforcing bounds. If a `linkId` exceeds the `uint32` limit, it causes overflow and spills into adjacent 32-bit segments, corrupting the packed data structure.

This could result in an unstaked `linkId` being permanently marked as staked, undermining the integrity of the staking system.

While the codebase references `55,000` NFTs, there are no enforced upper bounds on token IDs. The `ICNLink.batchSafeMintWithIds` function, accessible via the `MINTER` role, permits arbitrary high-value linkId minting without restriction, increasing the risk of such overflow conditions.

Recommendation

We recommend enforcing a constraint that each `linkId` is strictly less than `type(uint32).max` during the staking process.

Status: Resolved

13. Retroactive rewards curve updates allow overcompensation for unclaimed staking periods

Severity: Minor

The ICN protocol employs a progressive rewards curve to define the monthly distribution percentages of total rewards to NFT stakers. This curve is implemented as an array in `src/modules/LinkRewards/LinkRewards.sol:51-59`, with logic permitting updates to the curve while enforcing that new monthly percentages are not lower than the existing values.

However, the contract lacks safeguards against retroactive changes affecting unclaimed rewards. If a user delays claiming rewards, and the curve is updated to allocate higher percentages for past months, they can retroactively receive more than they were originally eligible for.

This undermines the fairness of the reward mechanism and may lead to excessive reward distribution inconsistent with the protocol's intended schedule.

Recommendation

We recommend introducing logic to freeze reward calculations for past periods at the time they elapse. This could be achieved by snapshotting the rewards curve monthly or maintaining a historical mapping of curve states per reward period.

Status: Resolved

14. Centralization risks

Severity: Minor

The smart contracts in scope are designed to rely on a trusted party to perform privileged operations.

Consequently, the overall security of the system depends on the trusted parties, particularly in relation to key management and operations.

Specifically:

- Admin role has complete control over all module upgrades via `addModule` and `removeModule`, while `renounceAdminRole` could permanently disable the contract if called.
- The `ICN_OPERATOR_ROLE` controls critical economic parameters, infrastructure registration, node verification, and reward mechanisms without safeguards like timelocks or multi-signature requirements.

Recommendation

We recommend enforcing strict key management, the usage of multi-signature accounts and evaluating the removal of the aforementioned privileged operations.

Status: Acknowledged

The client acknowledges the issue:

"This will be operating only during the near term bootstrapping phase of the protocol to ensure agility to respond to potential bugs at this critical time. At a later stage, the protocol will be transitioned to a governance-driven approach where all admin functions must pass governance votes."

15. Interface documentation inconsistencies may mislead about reward eligibility and node types

Severity: Informational

Two separate interface documentation inconsistencies may lead to developer confusion and misinterpretation of protocol behavior:

- In `src/modules/LinkRewards/interfaces/ILinkRewards.sol:45-46`, the `claimFixedRewards` function is documented as involving a 90-day waiting period before rewards can be claimed.

  However, the actual implementation in the LinkRewards contract allows for immediate reward transfer via `reserve.withdraw`. This aligns with external documentation stating that *"`20%` of the total `140,000,000` can be paid through a special function at any time without staking and without waiting period"* but contradicts the function comment.

- In `src/modules/LinkStaking/interfaces/ILinkStaking.sol:128-127`, the `NodeType` enum defines values `SN` and `HN`.

  However, comments throughout the interface mistakenly refer to node types as *"HN or HP"* introducing ambiguity about supported node classifications. These discrepancies undermine clarity and could mislead developers or auditors integrating or analyzing the protocol.

- In `src/modules/LinkRewards/interfaces/ILinkRewards.sol:52`, the comment states that reward curve values are expressed with four decimal places (e.g., `2500` represents `0.25` or `25%`). This contradicts the implementation, where values such as `1e18` are used directly in test scenarios, and no scaling logic is applied during reward calculation. The raw value is multiplied by the NFT count, and the resulting ICNT token amount is directly transferred, implying full precision rather than fixed decimal representation.

Recommendation

We recommend thoroughly reviewing and correcting the interface documentation to reflect the actual implementation logic and valid enum values.

Status: Resolved

16. Redundant contract imports across multiple files

Severity: Informational

In `src/modules/ExternalContractManager/ExternalContractManager.sol` and `src/modules/LinkStaking/LinkStaking.sol`, there are redundant import statements which decrease code readability and maintainability.

Specifically, in `ExternalContractManager`, the interfaces `IICNLink`, `IERC20`, and the contract types `ReservePool` and `Treasury` are imported directly, despite already being imported in `ExternalContractManagerStorage`.

Similarly, in `LinkStaking`, several interfaces and contracts are imported multiple times through different dependency paths.

Recommendation

We recommend removing redundant imports to improve code readability and potentially reduce deployment gas costs.

Status: Resolved

17. Missing public getters for external contract addresses

Severity: Informational

In `src/modules/ExternalContractManager/ExternalContractManager.sol`, the contract stores critical addresses like `icnLink`, `icnToken`, `reserve`, and `treasury`, but does not provide public getter functions to retrieve these values.

While the contract provides setters for reserve and treasury addresses and a getter for version, the lack of getters for external contract addresses reduces transparency.

Recommendation

We recommend adding public getter functions for all stored contract addresses to improve observability. This would allow users, auditors, and integrating systems to verify which external contracts are being used easily.

Status: Resolved

18. Diamond proxy implementation lacks EIP-2535 compliance

Severity: Informational

The Proxy contract in `src/Proxy/Proxy.sol` implements a customized and partial version of the Diamond proxy pattern without leveraging any established, battle-tested libraries.

Specifically, it omits all public view functions defined in EIP-2535, resulting in non-compliance with the standard.

This deviation impairs compatibility with EIP-2535-dependent infrastructure, including off-chain components and blockchain indexers, potentially disrupting automated interactions and introducing blind spots in system observability.

Recommendation

We recommend evaluating the usage of a battle-tested library and fully supporting the EIP-2535 standard to ensure compliance and enable proper interaction by off-chain systems.

Status: Acknowledged

The client acknowledges the issue:

“Strict compliance with EIP-2535 is not a key design requirement. We have pursued a minimal implementation that enables a modular approach while reducing complexity as much as possible to minimize potential attack surface.”

19. Contracts should implement a two-step ownership transfer

Severity: Informational

The contracts within the scope of this audit allow the current owner to execute a one-step ownership transfer. While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to an incorrect address.

A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated and lowercased.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Resolved

20. Inconsistent initialization event pattern in the LinkStaking contract

Severity: Informational

In `src/modules/LinkStaking/LinkStaking.sol:75-76`, the `initializeLinkStaking` function emits discrete events for individual parameters (`MinLinkStakingPeriodSet` and `LinkUnstakingPeriodSet`).

However, it does not emit a consolidated initialization event that encapsulates the full initialization context. This diverges from the approach used in other modules, such as LinkRewards, which emit a single, comprehensive initialization event for greater clarity and traceability.

Recommendation

We recommend introducing a dedicated event that aggregates all relevant initialization parameters.

Status: Resolved

21. No maximum limit on reward curve array size

Severity: Informational

In src/modules/LinkRewards/LinkRewards.sol:307-320, the _setRewardCurve function accepts and processes a dynamic array representing monthly reward percentages.

While the intended use case suggests approximately 48 elements to represent a four-year curve, there is no explicit upper bound enforced on the array length.

This omission theoretically permits submission of arbitrarily large arrays, which could result in excessive gas consumption and potentially contribute to block gas limit exhaustion.

Although this is currently mitigated by the Base network's 120 million gas limit per block, the lack of an enforced constraint introduces unnecessary risk.

Recommendation

We recommend adding an explicit maximum length check for the _rewardsCurve array to formalize the implicit 48-month limitation.

Status: Resolved

22. Possible rewards denial-of-service in case that claimed rewards exceed total rewards

Severity: Informational

In src/modules/LinkRewards/LinkRewards.sol:343-344, the protocol calculates accrued rewards for NFT staking by subtracting claimedRewards from totalCumRewards, returning the difference as the per-NFT reward.

This logic functions correctly under the current implementation, where claimedRewards is always less than or equal to totalCumRewards. However, the design does not guard against future changes that could disrupt this assumption. If updated logic results in claimedRewards exceeding totalCumRewards for a particular stake, the subtraction will underflow, causing a transaction revert.

Since this calculation is also performed during the unstaking process, such a condition could create a denial of service, preventing users from withdrawing their staked NFTs.

Recommendation

We recommend adding safety measures to return 0 in case that claimedRewards is major than totalCumRewards.

Status: Resolved

23. The `getHyperNode` view function reverts unexpectedly

Severity: Informational

In `src/modules/ICNRegistry/ICNRegistry.sol:445-449`, the `getHyperNode` function is declared as external and is intended for use by other contracts and off-chain clients.

However, it reverts if the specified HyperNode does not exist. This behavior complicates integration for external protocols, which must perform additional error handling to differentiate between a non-existent node.

Recommendation

We recommend modifying the function to return a tuple containing the `HyperNode` struct and a bool `isRegistered` flag.

Status: Resolved

24. Miscellaneous comments

Severity: Informational

Miscellaneous recommendations can be found below.

Recommendation

The following are some recommendations to improve the overall code quality and readability:

- Use `!= 0` instead of `> 0` for non-zero checks to save gas across the multiple instances where this pattern appears.

- In `src/modules/LinkStaking/interfaces/ILinkStaking.sol:63-65`, the comment on `initializeLinkStaking` incorrectly states it *"Can only be called by admin"* while the implementation restricts access with `onlySelf`. We recommend updating the interface comment to accurately reflect that the function can only be called internally by the contract itself.

- In `src/modules/LinkStaking/interfaces/ILinkStaking.sol:124`, add a `require` statement to validate that the `sortedLinkIds` array is not empty in the `claimFixedRewards` function to prevent confusing out-of-bounds errors.

- In `src/modules/LinkStaking/LinkStaking.sol:176-180`, there are two storage reads of `unstakingRequestEra`, which can be simplified to one storage read by caching the value early.

- In `src/modules/LinkStaking/LinkStaking.sol:163-165`, `currentEra + $.linkUnstakingPeriod` is stored as `linkStake.unstakingRequestEra` and then recalculated again in the event. Using `linkStake.unstakingRequestEra` when emitting the `LinkUnstakingRequested` event would save `SLOAD` operation.

Status: Resolved