Impossible Cloud Network Protocol

Security Review

Cantina Managed review by:

Noah Marconi, Lead Security Researcher
Hash, Security Researcher

June 12, 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

ICN Protocol is a decentralized infrastructure network, providing composable hardware for permissionless operation of internet and web3 services.

From May 2nd to May 17th the Cantina team conducted a review of icn-protocol on commit hash 90a54a01. The team identified a total of **30** issues:

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	5	5	0
Medium Risk	6	6	0
Low Risk	9	6	3
Gas Optimizations	5	4	1
Informational	5	4	1
Total	**30**	**25**	**5**

The Cantina Managed team reviewed icn-protocol holistically on commit hash 6f3fbdbf and concluded that all findings were addressed and no new issues were identified.

3 Findings

3.1 High Risk

3.1.1 Not updating `nodeIndex` when removing scalar nodes can cause future removals to revert

Severity: High Risk

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

Description: After moving the last element of the array to its new index the `scalerNodeIndexes` mapping is not updated to reflect the new position.

```
function removeScalerNode(uint256 scalerNodeId) external override whenNotPaused {

// ...

uint256 nodeIndex = ds.clusters[clusterId].scalerNodeIndexes[scalerNodeId];
ds.clusters[clusterId].scalerNodeIds[nodeIndex] =
    ds.clusters[clusterId].scalerNodeIds[ds.clusters[clusterId].scalerNodeIds.length - 1];
ds.clusters[clusterId].scalerNodeIds.pop();
```

This can cause removals to revert (causing stuck collateral) or clear another another node from the `scalerNodeIds` array.

Recommendation: Update the index of the moved node to `newIndex`.

ICN Protocol: Fixed in PR 188.

Cantina Managed: Fix verified.

3.1.2 Users can unstake a single link token several times making other link tokens to be non unstakeable

Severity: High Risk

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

Description: In case there are pending reward claims, a link stake is marked as `unstaked` rather than being removed. Such stake is already unstaked and should not be allowed to re-unstake.

```
function completeUnstaking(NodeType nodeType, uint256 stakeIndex) external override whenNotPaused {

// ...

if (getLinkRewardsStorage().rewards[linkStake.stakeId].claims.length == 0) {
    if (stakeIndex != stakesCount - 1) {
        $.linksStakes[msgSender][nodeType][stakeIndex] = $.linksStakes[msgSender][nodeType][stakesCount - 1];
    }
    $.linksStakes[msgSender][nodeType].pop();
} else {
    linkStake.unstaked = true;
}
```

But the kept validations in `completeUnstaking` doesn't enforce this and allows an unstaked link to be re-unstaked unlimited number of times. This will excessively reduce the `stakers` count and cause future untakings to revert due to underflow.

```
if (nodeType == NodeType.HN) {
    $.hyperNodes[nodeId].stakers -= linksCount;
} else {
    $.scalerNodes[nodeId].stakers -= linksCount;
}
```

Recommendation: in case `linkStake.unstaked == true`, revert in the `completeUnstaking` function.

ICN Protocol: Fixed in commit 74ac99b7.

Cantina Managed: Fix verified.

3.1.3 Node removal will cause delegators to loose their assets and rewards

Severity: High Risk

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

Description: The `settleHpRewardsDelegatorShare` function which is internally called by both `undelegateCollateral` and `initiateDelegationRewardsClaim` will revert incase the status of the node is not `ScalerNodeStatus.Validated`.

```
function settleHpRewardsDelegatorShare(uint256 _scalerNodeId) external override onlySelf {
    HPRewardsStorageData storage hs = getHPRewardsStorage();
    ICNRegistryStorageData storage rs = getICNRegistryStorage();
    ScalerNode storage scalerNode = rs.scalerNodes[_scalerNodeId];
    require(scalerNode.status == ScalerNodeStatus.Validated, IICNRegistryErrors.InvalidScalerNode()); // <<<
```

This is flawed because nodes can be removed as soon as their commitment ends (even when there are delegated assets and pending delegator rewards) and its status will change to `ScalerNodeStatus.None`. Hence the delegators will be unable to claim their assets and rewards.

Recommendation: Create and allow a new status similar to `ScalerNodeStatus.Removed` rather than reverting if `status != ScalerNodeStatus.Validated`.

ICN Protocol: Fixed in PR 189.

Cantina Managed: Fix verified.

3.1.4 Commitment end is not handled in `_processAddCollateralFromNodeRewards` causing underflow and stuck assets

Severity: High Risk

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

Description: Inside `_processAddCollateralFromNodeRewards`, commitmentRemaining is always calculated as `scalerNode.commitmentStart + scalerNode.commitmentDuration - block.timestamp`.

```
function _processAddCollateralFromNodeRewards(
    uint256 _scalerNodeId,
    uint256 _nodeCollateralToBeAdded,
    uint256 _networkCollateralToBeAdded
) internal {
    // ...

    uint256 commitmentRemaining = scalerNode.commitmentStart + scalerNode.commitmentDuration - block.timestamp;
```

This will cause the execution to revert in case block.timestamp is > `scalerNode.commitmentStart + scalerNode.commitmentDuration`. Since `_processAddCollateralFromNodeRewards` is invoked internally when removing delegator assets and node/delegator rewards, these actions can't be performed causing assets to be lost.

Recommendation: Explicitly set `commitmentRemaining` to 0 incase block.timestamp is > `scalerNode.commitmentStart + scalerNode.commitmentDuration`.

ICN Protocol: Fixed in PR 187.

Cantina Managed: Fix verified.

3.1.5 Incorrect `nodeId` is used to update existing delegation inside `delegateUnclaimedRewards` function

Severity: High Risk

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

Description: Inside `delegateUnclaimedRewards`, the to-be-delegated nodeId is used to update the existing delegation as well (instead of using nodeDelegation.nodeId). This messes up the existing delegation since there is no real connection between the existing delegation and the to-be-delegated nodeId and can cause excess rewards to be gained by the user.

```
function delegateUnclaimedRewards(
    uint256 _lockedDelegationIndex,
    uint256 _nodeDelegationIndex,
    uint256 _nodeId,
    uint256 _lockupDurationInSeconds
) external override onlyVerifiedNode(_nodeId) validLockupDuration(_lockupDurationInSeconds) whenNotPaused {
    uint256 baseIncentiveAccumulation = _commitDelegatorIncentiveRewards();
    IHPRewards(address(this)).settleHpRewardsDelegatorShare(_nodeId);

    // ...

    (UserDelegation storage userDelegation, NodeDelegation storage nodeDelegation) =
        _getUserNodeDelegation(msg.sender, _lockedDelegationIndex, _nodeDelegationIndex);
```

Recommendation: Use `nodeDelegation.nodeId` in-order to update the existing delegation.

ICN Protocol: Fixed in PR 186.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 Late delegation check causes collateral redirection to be skipped

Severity: Medium Risk

Context: HPRewards.sol#L74-L86

Description: The `HPRewards.initiateHpRewardsClaim` function subtracts the `delegatorShare` from `unclaimedRewards` before determining how many rewards should be redirected to collateral:

```
uint256 delegatorShare = (unclaimedRewards * scalerNode.nodeRewardShare) / M;
unclaimedRewards -= delegatorShare;

// ...

(unclaimedRewards, nodeCollateralToBeAdded, networkCollateralToBeAdded) =
    _calculateNodeRewardsLeftAfterRedirection(_scalerNodeId, unclaimedRewards);
unclaimedRewards += scalerNodeData.rewardDebt;

// Credit the delegator rewards
{
    bool hasDelegation = _addNodeRewardShareForDelegators(_scalerNodeId, delegatorShare);
    if (!hasDelegation) {
        // If there is no delegation, the delegator share is 0
        unclaimedRewards += delegatorShare;
        delegatorShare = 0;
    }
}
```

In the case where there is no delegation the `delegatorShare` is added back to `unclaimedRewards` to be claimed after the `claimUnlockTimestamp` and are not used for collateral as required by the `Collateral Requirements and Rewards Redirection`.

For any period where there are no delegators, a node may redirect all of its rewards to delegators, allowing them to circumvent any redirection of rewards for collateral purposes.

Recommendation: Perform the delegation check prior to calculating reward redirection amounts.

ICN Protocol: Fixed in PR 189.

Cantina Managed: Fix verified.

3.2.2 Slippage can cause user's to pay more for booking capacity than they are willing

Severity: Medium Risk

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

Description: The price to book capacity can change in between a user signing the transaction and its execution due to changes in `reservationPrice`, `maxPrice` or `marketAdjustmentFactor`. This can cause users to spend more than what they had expected when signing the transaction.

```
function bookCapacity(uint256 capacity, uint256 period, string calldata clusterId) external override
↪   whenNotPaused {

// ...

uint256 bookingPrice;
for (uint256 i; i < rs.clusters[clusterId].scalerNodeIds.length; ++i) {
    ICNRegistryStorage.ScalerNode storage scalerNode = rs.scalerNodes[rs.clusters[clusterId].scalerNodeIds[i]];
    if (
        capacity == scalerNode.totalCapacity && scalerNode.utilizedCapacity == 0
            && block.timestamp + period <= scalerNode.commitmentStart + scalerNode.commitmentDuration
    ) {
        bookedNodeId = rs.clusters[clusterId].scalerNodeIds[i];

        bookingPrice = _transferBookingPrice(capacity, period, clusterId, scalerNode.reservationPrice);
```

Recommendation: Add a parameter to limit the maximum amount that can be spend.

ICN Protocol: Fixed in PR 204.

Cantina Managed: Fix verified.

3.2.3 Lack of validation for `nodeRewardShare` allows a malicious node to DoS delegator withdrawals

Severity: Medium Risk

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

Description: `nodeRewardShare` can be set by a node and is not validated to be <= 100% (ie. the conceptual maximum). This allows a node to set `nodeRewardShare` to arbitrarily high values like uint.max which will cause overflow in multiple functions including `settleHpRewardsDelegatorShare` which will disable delegators from withdrawing their collateral.

```
function updateScalerNodeNodeRewardShare(uint256 scalerNodeId, uint256 nodeRewardShare)
    external
    override
    onlyHPAndValidatedScalerNode(scalerNodeId)
    whenNotPaused
{
    ICNRegistryStorageData storage $ = getICNRegistryStorage();
    ScalerNode storage scalerNode = $.scalerNodes[scalerNodeId];

    // Settle the HP rewards delegator share for the scaler node, so that post-update
    // the new rewards are distributed according to the new node reward share
    IHPRewards(address(this)).settleHpRewardsDelegatorShare(scalerNodeId);

    scalerNode.nodeRewardShare = nodeRewardShare;
    emit ScalerNodeNodeRewardShareUpdated(scalerNodeId, nodeRewardShare);
```

Recommendation: Limit `nodeRewardShare` to 100%.

ICN Protocol: Fixed in commit ed611117.

Cantina Managed: Fix verified.

3.2.4 Link associated collateral is computed incorrectly inside `getScalerNodeTotalCollateral`

Severity: Medium Risk

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

Description: The link associated collateral should actually be the pending unemitted rewards. But in the implementation, this is calculated as `getNftCumCurvePoint(endPoint, startPoint)` with (usually) `startPoint == block.timestamp` and `endPoint == es.icnLink.activationTime() + es.icnLink.durationTime()`.

```
function getScalerNodeTotalCollateral(uint256 scalerNodeId)
    external
    view
    override
    returns (uint256 totalCollateral, uint256 nodeCollateral, uint256 networkCollateral)
{

    // ...

    uint256 endPoint = es.icnLink.activationTime() + es.icnLink.durationTime();
    LinkRewardsStorageData storage ds = getLinkRewardsStorage();
    uint256 startPoint = Math.max(ds.rewardsActivationTs, block.timestamp);

    if (endPoint > startPoint) {
        uint256 totalCumReward = ILinkRewards(address(this)).getNftCumCurvePoint(endPoint, startPoint);
        uint256 potentialRewards =
            stakers * (totalCumReward * ProtocolConstants.TOTAL_NFT_REWARDS_POOL) /
            ↪  ProtocolConstants.TOTAL_NFT_TOKENS;
        networkCollateral += potentialRewards;
```

This is incorrect as `getNftCumCurvePoint` will always begin from index 0 of the curve rather than the remaining unemitted portion.

eg:

```
rewardCurve = [10,90,95,100]
time delta b/w each index = 10s
```

Now after 20s, 90% of the entire rewards have been claimed but the calculation above will still attribute 90% to node's collateral instead of considering the actual remaining unclaimed amount ie. 10%.

Recommendation: Subtract the already emitted fraction from the total fraction.

```
uint256 endPoint = es.icnLink.activationTime() + es.icnLink.durationTime();
LinkRewardsStorageData storage ds = getLinkRewardsStorage();
uint256 startPoint = Math.max(ds.rewardsActivationTs, block.timestamp);
    uint basePoint = ds.rewardsActivationTs;

if (endPoint > startPoint) {
        uint refRewards = ILinkRewards(address(this)).getNftCumCurvePoint(startPoint,basePoint);
        uint totalRewards = ILinkRewards(address(this)).getNftCumCurvePoint(endPoint,basePoint);
    uint256 totalCumReward = totalRewards - refRewards;
// ...
```

ICN Protocol: Fixed in PR 197.

Cantina Managed: Fix verified.

3.2.5 Performing reward curve updation from (`currentMonth + 1`) can cause some portion of rewards to be unclaimable

Severity: Medium Risk

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

Description: Currently reward curve updations are allowed to be performed starting from `(block.timestamp - $.rewardsActivationTs) / ProtocolConstants.ONE_MONTH + 1`. Although this avoid changing the value of the currently active month, it still allows changes to activeMonth + 1.

```
function updateRewardsCurve(uint256[] calldata _rewardsCurve) external override onlyRole(ICN_OPERATOR_ROLE) {
    LinkRewardsStorageData storage $ = getLinkRewardsStorage();
    require($.rewardsCurve.length != 0, RewardCurveNotSet());
    uint256 currentIndex = (block.timestamp - $.rewardsActivationTs) / ProtocolConstants.ONE_MONTH + 1;
    require($.rewardsCurve.length == currentIndex + _rewardsCurve.length, LinkRewardsInvalidParams());

    // ...

    for (uint256 i = 0; i < _rewardsCurve.length - 2; i++) {
        require(
            (i == 0 ? _rewardsCurve[i] != 0 : _rewardsCurve[i] >= _rewardsCurve[i - 1])
                && ($.rewardsCurve[currentIndex + i] <= _rewardsCurve[i]),
            InvalidRewardCurveValue()
        );
        $.rewardsCurve[currentIndex + i] = _rewardsCurve[i];
    }

    emit RewardCurveUpdated($.rewardsCurve, block.timestamp);
}
```

This can cause some portion of rewards to be unclaimable by the users as the extrapolation for reference reward performed inside `getNftCumCurvePoint` will assume that the new increased amount has been claimed.

```
function getPendingRewards(address claimer, uint256 stakeIndex, ILinkStakingStorage.NodeType nodeType)
    public
    view
    override
    returns (uint256 pendingRewards)
{

    // ...

    uint256 referenceTs = latestClaimTs != 0 ? latestClaimTs : Math.max(stakeStartTs, rewardsActivationTs);
    uint256 refRewards = getNftCumCurvePoint(referenceTs, rewardsActivationTs);
```

```
function getNftCumCurvePoint(uint256 endPoint, uint256 startPoint) public view override returns (uint256) {
    // ...

    uint256 rt0 = ds.rewardsCurve[index];
    uint256 rt1 = ds.rewardsCurve[index + 1];
    return rt0 + ((rt1 - rt0) * remainder) / ProtocolConstants.ONE_MONTH;
}
```

eg:

```
reward curve = [5,10,20,30,90,100]
time delta b/w each index == 10s
lastClaimedTs == currentTimestamp = 29 seconds
hence currentIndex = 2
claimed rewards == 29% of total
now updation of reward curve occurs,
new reward curve = [5,10,20,90,95,100]
now users will only be able to claim (100 - (20 + 0.9 * 70)) == 17% more, making their total claimable to 46%
```

Recommendation: Only perform updations from `currentIndex + 2`.

ICN Protocol: Fixed in PR 199.

Cantina Managed: Fix verified.

3.2.6 Not capping `t2` to `basis + ProtocolConstants.RELEASE_SCHEDULE_DURATION` will cause lost rewards due to negative value addition

Severity: Medium Risk

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

Description: The reward in the linear decreasing region will drop to 0 after `RELEASE_SCHEDULE_DURATION`. Any timestamp after that will result in negative values and hence the rewards calculation should limit the timestamp to `basis/startingPoint + RELEASE_SCHEDULE_DURATION`. But this is not enforced in the implementation causing negative value addition to take places thereby decreasing the rewards.

```
function _calculateAggregateBootstrapReleaseInLinearDecreaseRegion(
    uint256 _t1,
    uint256 _t2,
    uint256 _releaseSchedule,
    uint256 _marketAdjustmentFactor
) internal pure returns (uint256) {
    int256 value = int256(_t2 - _t1) * IM;
    value -= ((int256(_t2) * int256(_t2 + 1) - int256(_t1) * int256(_t1 + 1)) * IM)
        / (2 * int256(ProtocolConstants.RELEASE_SCHEDULE_DURATION));
    value = value < 0 ? int256(0) : value;
    return (uint256(value) * _marketAdjustmentFactor * _releaseSchedule) / M;
}
```

Apply the following diff and run `forge test --mt testPOC_rewardsNegativeCancelOutPositive -vv`. It can be seen that the rewardAmount peaks at `t == 10`, decreases afterwards and drops to 0 at `t == 20`.

```
diff --git a/src/modules/HPRewards/HPRewards.sol b/src/modules/HPRewards/HPRewards.sol
index f851765..72f5348 100644
--- a/src/modules/HPRewards/HPRewards.sol
+++ b/src/modules/HPRewards/HPRewards.sol
@@ -580,4 +580,15 @@ contract HPRewards is
         value = value < 0 ? int256(0) : value;
         return (uint256(value) * _marketAdjustmentFactor * _releaseSchedule) / M;
     }
+
+    function forTesting_calculateAggregateBootstrapReleaseInLinearDecreaseRegion(uint256 _t1,
+        uint256 _t2,
+        uint256 _releaseSchedule,
+        uint256 _marketAdjustmentFactor,uint totalDuration) public returns (uint256){
+        int256 value = int256(_t2 - _t1) * IM;
+        value -= ((int256(_t2) * int256(_t2 + 1) - int256(_t1) * int256(_t1 + 1)) * IM)
+            / (2 * int256(totalDuration));
+        value = value < 0 ? int256(0) : value;
+        return (uint256(value) * _marketAdjustmentFactor * _releaseSchedule) / M;
+    }
 }
diff --git a/src/modules/HPRewards/interfaces/IHPRewards.sol b/src/modules/HPRewards/interfaces/IHPRewards.sol
index 6317080..264c644 100644
--- a/src/modules/HPRewards/interfaces/IHPRewards.sol
+++ b/src/modules/HPRewards/interfaces/IHPRewards.sol
@@ -172,4 +172,12 @@ interface IHPRewards is IHPRewardsErrors, IHPRewardsStorage {
     /// @param _hp The address of the HP.
     /// @return The reward claims.
     function getHpRewardClaims(address _hp) external view returns (RewardClaim[] memory);
+
+    function forTesting_calculateAggregateBootstrapReleaseInLinearDecreaseRegion(
+        uint256 _t1,
+        uint256 _t2,
+        uint256 _releaseSchedule,
+        uint256 _marketAdjustmentFactor,
+        uint256 totalDuration
+    ) external returns (uint256);
 }
diff --git a/test/HPRewards.t.sol b/test/HPRewards.t.sol
index ca78590..2aacf61 100644
--- a/test/HPRewards.t.sol
+++ b/test/HPRewards.t.sol
@@ -10,6 +10,7 @@ import {IHPRewardsErrors} from "../src/modules/HPRewards/interfaces/IHPRewardsEr
 import {Math} from "@openzeppelin/contracts/utils/math/Math.sol";
 import {Strings} from "@openzeppelin/contracts/utils/Strings.sol";
 import {console2} from "forge-std/console2.sol";
+import "forge-std/Test.sol";

 contract HPRewardsTest is Init {
     uint256 private constant M = ProtocolConstants.DEFAULT_PRECISION;
@@ -139,6 +140,35 @@ contract HPRewardsTest is Init {
         mockICNToken.approve(address(icnp), 1000000000e18);
     }

+    function testPOC_rewardsNegativeCancelOutPositive() public {
+        /**
+        curve:
+         s = 10
+         te = 10
+         t from 0 to 10 increases
+         */
+        uint maxRewardsObtainable =
↪   icnp.forTesting_calculateAggregateBootstrapReleaseInLinearDecreaseRegion(0,10,10e18,1e18,10);
+
+        uint decreasedRewardAmountAfterTime =
↪   icnp.forTesting_calculateAggregateBootstrapReleaseInLinearDecreaseRegion(0,20,10e18,1e18,10);
+        console.log("maxRewardsObtainable",maxRewardsObtainable);
+
+        console.log("decreasedRewardAmountAfterTime",decreasedRewardAmountAfterTime);
+
+    }
+
+    function testPOC_incorrectFormulaCauseT1Omit() public {
+        /**
+        curve:
+         s = 10
+         te = 10
+         t from 0 to 2. should be [t1,t2) else t1 reward would be wasted. so should be 10 + 9 == 19. but here
↪   it will be 9 + 8. also could just take the sum of total duration and verify that it omits the first
↪   timestamp
+         */
+        uint rewardsCalculated =
↪   icnp.forTesting_calculateAggregateBootstrapReleaseInLinearDecreaseRegion(0,2,10e18,1e18,10);
+
+        console.log("rewards calculated",rewardsCalculated);
+
+    }
+
```

Recommendation: Limit t2 to `basis + RELEASE_SCHEDULE_DURATION`.

ICN Protocol: Fixed in PR 191.

Cantina Managed: Fix verified.

3.3 Low Risk

3.3.1 `regionId` based deposits overwrite previous amount

Severity: Low Risk

Context: ReservePool.sol#L62

Description: The `ReservePool.deposit` has to overloaded variations, one taking a single argument `deposit(uint256 depositAmount)`, and the other taking two arguments `deposit(string calldata regionId, uint256 baseReward)`. The later variation overwrites the `baseReward` on each subsequent call, rather than adding to the existing value:

```
$.regionReward[regionId].baseReward = baseReward;
```

If this function were used in the protocol, it would not be possible to withdraw all funds due to the restriction in the withdraw function:

```
require($.regionReward[regionId].withdrawnReward + amount <= $.regionReward[regionId].baseReward,
↪   AmountExceedBaseReward());
```

Recommendation: Notably, neither of these two functions are used within the protocol. As such, the severity is noted as low. However, it is still recommended to remediate by removing them both from the code base. This prevents the issue from occurring and reduces the overall surface area.

ICN Protocol: Fixed in PR 202.

Cantina Managed: Fix verified.

3.3.2 `initiateHpRewardsClaim` should always be manually invoked before node removal in-order to not loose rewards

Severity: Low Risk

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

Description: The `removeScalerNode` only commits capacity rewards of the hwClass and doesn't handle pending rewards of the node. This can cause nodes to loose their pending rewards in case they directly invoke `removeScalerNode` without invoking `initiateHpRewardsClaim` first.

```
function removeScalerNode(uint256 scalerNodeId) external override whenNotPaused {
    ICNRegistryStorageData storage ds = getICNRegistryStorage();

    // Commit the rewards for the node's region and hwClass since the total capacity will be changed
    string memory clusterId = ds.scalerNodes[scalerNodeId].clusterId;
    string memory regionId = ds.clusters[clusterId].regionId;
    string memory hwClass = ds.scalerNodes[scalerNodeId].hwClass;
    IHPRewards(address(this)).commitHpRewards(regionId, hwClass);
```

Recommendation: Either document this behaviour or process the pending rewards in `removeScalarNode` itself.

ICN Protocol: Fixed in PR 189.

Cantina Managed: Fix verified.

3.3.3 Slashing is un-enforceable during final moments of commitment due to instant collateral withdrawal

Severity: Low Risk

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

Description: Users derive their security from the expectation that the scalar nodes are slashable throughout their commitment period. But the current implementation allows nodes to remove themselves instantly as soon as their commitment period ends (without enforcing any queue/delay mechanism). This means that a node can behave maliciously in the final moments of their commitment and escape the slashing by removing themselves.

```
function slashScalerNode(uint256 scalerNodeId, uint256 slashedAmount) external override
↪   onlyRole(ICN_OPERATOR_ROLE) {
    require(slashedAmount != 0, SlashingInvalidAmount(0, 1));
    ICNRegistryStorageData storage rs = ICNRegistryStorage.getICNRegistryStorage();
    require(rs.scalerNodes[scalerNodeId].status == ScalerNodeStatus.Validated,
    ↪   IICNRegistryErrors.InvalidScalerNode());
    require(
        rs.scalerNodes[scalerNodeId].collateralAmount >= slashedAmount,
        SlashingInsufficientCollateral(rs.scalerNodes[scalerNodeId].collateralAmount, slashedAmount)
    );

    rs.scalerNodes[scalerNodeId].collateralAmount -= slashedAmount;
```

Recommendation: Document this behaviour or introduce a queue/delay mechanism for collateral withdrawal.

ICN Protocol: Acknowledged because the issue will be fixed separately after upgrading the slashing logic. The issue, while valid, will not occur until the node's commitment period has ended and by that time we would have updated the slashing logic.

Cantina Managed: Acknowledged.

3.3.4 Booking can get overwritten in case the reservation price was 0

Severity: Low Risk

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

Description: In order to check the existence of a booking, currently `bookingPrice` is checked against 0 value. This is not accurate as an existing booking will have bookingPrice == 0 if the reservation price was 0 (although unlikely). This will cause this booking to be freely overwritten.

```
function extendBooking(uint256 scalerNodeId, uint256 period) external override whenNotPaused {

    // ....

    // Write the new booking to storage
    Booking storage b0 = scalerNode.bookings[0];
    Booking storage b1 = scalerNode.bookings[1];
    if (b1.bookingPrice != 0) {
        // Check that the first booking is expired before overwriting it
        require(
            b0.startBookingPeriod + b0.bookingPeriod < block.timestamp,
            FirstBookingNotExpired(b0.startBookingPeriod + b0.bookingPeriod, block.timestamp)
        );

        scalerNode.bookings[0] = b1;
    }

    scalerNode.bookings[1] = newBooking;

    emit BookingExtended(scalerNodeId, period);
}
```

Recommendation: Check for `booking.startBookingPeriod == 0` instead.

ICN Protocol: Fixed in commit 868f534c.

Cantina Managed: Fix verified.

3.3.5 Used summation formula omits the first timestamp and hence its reward

Severity: Low Risk

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

Description: The used formula for the summation of linear region currently covers (t1,t2] instead of [t1,t2). This will cause the rewards of the first timestamp to be omitted.

```
function _calculateAggregateBootstrapReleaseInLinearDecreaseRegion(
    uint256 _t1,
    uint256 _t2,
    uint256 _releaseSchedule,
    uint256 _marketAdjustmentFactor
) internal pure returns (uint256) {
    int256 value = int256(_t2 - _t1) * IM;
    value -= ((int256(_t2) * int256(_t2 + 1) - int256(_t1) * int256(_t1 + 1)) * IM)
        / (2 * int256(ProtocolConstants.RELEASE_SCHEDULE_DURATION));
    value = value < 0 ? int256(0) : value;
    return (uint256(value) * _marketAdjustmentFactor * _releaseSchedule) / M;
}
```

Apply the diff in the finding "Not capping `t2` to `basis + ProtocolConstants.RELEASE_SCHEDULE_DURATION` will cause lost rewards due to negative value addition" and run `forge test --mt testPOC_incorrectFormulaCauseT1Omit -vv`. It can be seen that the reward calculation omits the reward of the first timestamp.

Recommendation: Change the range to cover $[t_1, t_2)$ instead.

ICN Protocol: Acknowledged. We ran a test to measure the amount of base rewards generated by a node in an overall network configuration similar to what we expect in production. The difference in the base rewards generated by this node in the first day was 0.005%, which is within an acceptable error margin. The same applies despite using $(t1, t2]$.

Cantina Managed: Acknowledged.

3.3.6 `unclaimedHpRewards` doesn't handle the case of 0 delegations causing incorrect reward reporting

Severity: Low Risk

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

Description: In case there are no delegations, the entire reward accrued should go to the node. But the `unclaimedHpRewards` function doesn't consider this scenario and always assumes that `nodeRewardShare` percentage of rewards will go to the delegators (and hence be subtracted from the nodeClaimableRewards).

```
function unclaimedHpRewards(uint256 _scalerNodeId)
    public
    view
    override
    returns (uint256 nodeClaimableRewards, uint256 delegatorRewards)
{

    // ...

    delegatorRewards = (unclaimedRewards * scalerNode.nodeRewardShare) / M;
    (nodeClaimableRewards,,) = _calculateNodeRewardsLeftAfterRedirection(_scalerNodeId, unclaimedRewards -
    ↪  delegatorRewards);
    nodeClaimableRewards += hs.scalerNodeData[_scalerNodeId].rewardDebt;
}
```

Recommendation: Include the scenario where `nodeTotalDelegatedICNT == 0`.

ICN Protocol: Fixed in PR 189.

Cantina Managed: Fix verified.

3.3.7 Users can DoS future node bookings by keeping < `minBookingPeriod` leftover

Severity: Low Risk

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

Description: A booking should at least be `minBookingPeriod` (initially set to 3 months) long. A user can abuse this to make a node unbookable by booking a period of time such that the remaining commitmentPeriod is less than `minBookingPeriod`.

```
function bookCapacity(uint256 capacity, uint256 period, string calldata clusterId) external override
↪  whenNotPaused {

    // ...

    require($.minBookingPeriod <= period && period <= $.maxBookingPeriod, InvalidBookingPeriod()); // <<<
```

Eg:

```
commitment end = 100
minbookingPeriod = 10
at t == 50, a user books for 41
```

Now the capacity cannot be booked and the node will only receive capacity rewards for this timeframe.

Recommendation: For a booking always ensure that the remaining amount of commitment time is alteast minBookingPeriod or is 0.

ICN Protocol: Acknowledged. This is part of the tokenomics design.

Cantina Managed: Acknowledged.

3.3.8 Excessively high capacity permitted when `marketAdjustmentFactor` or `minCollateralPercent` are at or near 0

Severity: Low Risk

Context: ICNRegistry.sol#L80-L88, ICNRegistry.sol#L91-L98

Description: The docs suggest a range of $p \in (0, 1]$ for `minCollateralPercent` meaning a minimum of 1e18 according to the implementation. There is, however, no validation to prevent a value of 0 from being set. Similarly, `marketAdjustmentFactor` may be set to 0 which will then DoS the validation in the `_calculateCapacityRewardsCheckPointIncreaseSinceLastUpdate` function.

When permitted to be 0, a high capacity may be registered with no collateral needed. What can make the issue more damaging is that temporarily setting 0 and later setting higher, can make a previously safe calculation revert due to overflow.

Recommendation: Consider a reasonable range for `marketAdjustmentFactor`. Enforce at the time of setting that `minCollateralPercent` is a number between 1 and 100.

ICN Protocol: Fixed in PR 190.

Cantina Managed: Fix verified.

3.3.9 Separately calculating `xSlope` causes lower precision and possible revert due to rounding error

Severity: Low Risk

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

Description: The `xSlope` is calculated separately where a rounded down division is performed. This lowers the precision that is attainable in the calculations and can cause reverts due to `lowIndex + 1` being greater than array length:

```
function calculateMaxApy(uint256 _collateralizationRate) public view override returns (uint256) {
    // ...

    uint256 xSlope = (maxX - minX) / (curveLength - 1); // <<<
    uint256 lowIndex = (_collateralizationRate - minX) / xSlope;
    if (((curveLength - 1) * (_collateralizationRate - minX)) % (maxX - minX) == 0) {
        return ds.maxApyCurve[lowIndex];
    }
    int256 lowX = int256(lowIndex * xSlope + minX);
    int256 lowY = int256(maxApyCurve[lowIndex]);
    int256 highY = int256(maxApyCurve[lowIndex + 1]);
    // ...
```

Eg:

```
curveLength == 7
collateralizationRate = 9999999999999999
xSlope = (maxX - minX) / (curveLength - 1) = 1666666666666666
lowIndex = (_collateralizationRate - minX) / xSlope = 6
```

Hence `(lowIndex + 1) == 7` which gives out of bounds array access.

Recommendation: Inline the `xSlope` calculation wherever it is used.

ICN Protocol: Fixed in PR 200.

Cantina Managed: Fix verified.

3.4 Gas Optimization

3.4.1 Avoid duplicate `sload`s by using the expression's evaluated value

Severity: Gas Optimization

Context: HPDelegationICNT.sol#L562, LinkStaking.sol#L140

Description:

- **First Instance:** A revision as follows eliminates an `SLOAD`.

  ```
  - linkStake.stakeId = $.stakeIdCounter;
  - $.stakeIdCounter++;
  + linkStake.stakeId = $.stakeIdCounter++;
  ```

  Original generated `yul`:

```
// sload to write to linkStake.stakeId
sstore(_4, sload(/** @src 43:6275:6291  "$.stakeIdCounter" */
↪  0x1c11073d71c45bef9120006caf8fea8a61a41b4d1b4b911bb106459d47d1440a))

// sload again to increment
update_storage_value_offset_uint256_to_uint256(increment_uint256(/** @src 43:1040:17800  "contract
↪  LinkStaking is..." */ sload(/** @src 43:6275:6291  "$.stakeIdCounter" */
↪  0x1c11073d71c45bef9120006caf8fea8a61a41b4d1b4b911bb106459d47d1440a)))
```

`yul` after revision:

```
// resulting yul
let _6 := sload(/** @src 43:6275:6291  "$.stakeIdCounter" */
↪  0x1c11073d71c45bef9120006caf8fea8a61a41b4d1b4b911bb106459d47d1440a)
update_storage_value_offset_uint256_to_uint256(increment_uint256(_6))
sstore(_4, _6)
```

- **Second Instance:** A second opportunity for the same savings is in the extra sload to read the new array length after pushing to increase its length.

```
- ds.delegations[_delegator].push();
- UserDelegation storage userDelegation = ds.delegations[_delegator][ds.delegations[_delegator].length -
↪  1];
+ UserDelegation storage userDelegation = ds.delegations[_delegator].push();
```

Original generated `yul`:

```
// Array reference
let _1 := mapping_index_access_mapping_address_bytes32_of_address_30734(var_delegator)

// Increases array length
sstore(_1, add(oldLen, 1))

// Check length and references newly created array element in storage
let slot, offset := storage_array_index_access_struct_UserDelegation__dyn(_1, oldLen)

// Array reference
let _2 := mapping_index_access_mapping_address_bytes32_of_address_30734(var_delegator)

// Load length again
let length := sload(/** @src 29:24573:24599  "ds.delegations[_delegator]" */
↪  mapping_index_access_mapping_address_bytes32_of_address_30734(var_delegator))

// Length - =
let diff := add(length, not(0))
if gt(diff, length)
{
    /// @src 20:252:259  "30 days"
    mstore(/** @src -1:-1:-1 */ 0, /** @src 20:252:259  "30 days" */ shl(224, 0x4e487b71))
    mstore(4, 0x11)
    revert(/** @src -1:-1:-1 */ 0, /** @src 20:252:259  "30 days" */ 0x24)
}

// Same as `slot, offset` above
let _3, _4 := storage_array_index_access_struct_UserDelegation__dyn(_2, /** @src 29:24573:24610
↪  "ds.delegations[_delegator].length - 1" */ diff)
```

`yul` after revision, uses `slot` and skips extra length `SLOAD`s:

```
let _1 := mapping_index_access_mapping_address_bytes32_of_address(var_delegator)
let oldLen := sload(_1)
if iszero(lt(oldLen, 18446744073709551616))
{
    mstore(/** @src -1:-1:-1 */ 0, /** @src 20:252:259  "30 days" */ shl(224, 0x4e487b71))
    mstore(4, 0x41)
    revert(/** @src -1:-1:-1 */ 0, /** @src 29:976:31979  "contract HPDelegationICNT is..." */ 0x24)
}
sstore(_1, add(oldLen, 1))
let slot, offset := storage_array_index_access_struct_UserDelegation__dyn(_1, oldLen)
```

ICN Protocol: Fixed in PR 196.

Cantina Managed: Fix verified.

3.4.2 The overloaded `updateModule` (without `initdata` argument) applies the `onlyAdmin` modifier twice

Severity: Gas Optimization

Context: Proxy.sol#L84

Description: This function applies the `onlyAdmin` modifier then calls the other `updateModule` function which again applies the same modifier.

Recommendation: Save gas by applying the modifier only once.

ICN Protocol: Acknowledged. Proxy contract is not upgradeable and has already been deployed. This finding will not be fixed.

Cantina Managed: Acknowledged.

3.4.3 May use unchecked math to save small amounts of gas

Severity: Gas Optimization

Context: HPRewards.sol#L399-L400, HPRewards.sol#L411-L413, Slashing.sol#L34-L39, ReservePool.sol#L105-L107

Description: There are instances where the arithmetic is certain to not under/overflow due to an explicit check in the code preceding the operation.

Recommendation: Consider using unchecked math to save small amounts of gas.

ICN Protocol: Fixed in PR 201 and PR 206.

Cantina Managed: Fix verified.

3.4.4 Array elements deleted twice

Severity: Gas Optimization

Context: HPDelegationICNT.sol#L588-L591

Description: `HPDelegationICNT._setMaxApyCurve` iterates and deletes each element in the array before deleting the array itself. This results in the array elements being iterated over twice, once by the code and once by the language level `delete`.

```
// First iteration

let var_i := /** @src -1:-1:-1 */ 0
for { }
/** @src 29:976:32053  "contract HPDelegationICNT is..." */ 1
{
    var_i := /** @src 29:976:32053  "contract HPDelegationICNT is..." */ add(/** @src 29:25871:25874  "i++" */
    ↪  var_i, /** @src 29:976:32053  "contract HPDelegationICNT is..." */ 1)
}
{
    if iszero(lt(var_i, /** @src 29:976:32053  "contract HPDelegationICNT is..." */ sload(/** @src
    ↪  29:25848:25862  "ds.maxApyCurve" */
    ↪  0x86284dd90e18a3083f5174fbac7645faf9a1f193a5535c362180782092a3ff07)))
    { break }

    let _1, _2 := storage_array_index_access_uint256_dyn(var_i)

    let _3 := sload(_1)

    // Sets to 0
    sstore(_1, and(_3, not(shl(shl(3, _2), not(0)))))
}


// Second iteration


let oldLen := sload(/** @src 29:25848:25862  "ds.maxApyCurve" */
↪  0x86284dd90e18a3083f5174fbac7645faf9a1f193a5535c362180782092a3ff07)
```

```
sstore(/** @src 29:25848:25862  "ds.maxApyCurve" */
↪  0x86284dd90e18a3083f5174fbac7645faf9a1f193a5535c362180782092a3ff07, /** @src -1:-1:-1 */ 0)

if iszero(iszero(oldLen))
{
    mstore(/** @src -1:-1:-1 */ 0, /** @src 29:25848:25862  "ds.maxApyCurve" */
    ↪  0x86284dd90e18a3083f5174fbac7645faf9a1f193a5535c362180782092a3ff07)
    let data := keccak256(/** @src -1:-1:-1 */ 0, /** @src 29:976:32053  "contract HPDelegationICNT is..." */
    ↪  0x20)
    let _4 := add(data, oldLen)
    let start := data

    for { } lt(start, _4) { start := add(start, 1) }
    {
        sstore(start, /** @src -1:-1:-1 */ 0)
    }
}
```

Recommendation: Save gas by using the language level delete only `delete ds.maxApyCurve;`.

ICN Protocol: Fixed in PR 201.

Cantina Managed: Fix verified.

3.4.5 Revert early to save gas on storage write in reverting case

Severity: Gas Optimization

Context: LinkStaking.sol#L106-L110

Description/Recommendation: Moving `minLinkStakingPeriod <= ProtocolConstants.MAX_MIN_-LINK_STAKING_PERIOD_IN_SECONDS` to appear before `$.minLinkStakingPeriod = minLinkStakingPeriod;` would skip the storage write in cases where the transactions revert. Saving a small amount of gas for the reverting scenario.

ICN Protocol: Fixed in commit c415dbac.

Cantina Managed: Fix verified.

3.5 Informational

3.5.1 Enforce a reasonable maximum on `minWaitPeriodForClaimsWithdrawal` to prevent admin error

Severity: Informational

Context: LinkRewards.sol#L43

Description: `minWaitPeriodForClaimsWithdrawal` dictates when withdraws are permitted. This value is used non-retroactively meaning updates that affect claim are locked for the period, even if a lower period is set by an admin later.

Recommendation: Adding validation to the `setMinWaitPeriodForClaimsWithdrawal` function to enforce a reasonable max would prevent admin errors.

ICN Protocol: Fixed in commit 41166f8f.

Cantina Managed: Fix verified.

3.5.2 Node registration `reservationPrice` is a maximum price not a minimum

Severity: Informational

Context: ICNRegistry.sol#L316

Description: Booking prices are determined by selecting the smaller of two possible prices:

```
bookingPrice = Math.min(getMaxBookingPrice(capacity, period, clusterId), reservationPrice * capacity * period);
```

Recommendation: Ensure in documentation, and the UI, this style of pricing is well communicated. No changes to the code recommended.

ICN Protocol: Acknowledged. This will be included in documentation for booking pricing.

Cantina Managed: Acknowledged.

3.5.3 `getCluster` function doesn't return `hwClass`

Severity: Informational

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

Description: `hwClass` is one of the most important fields of a cluster but this is not returned.

```
function getCluster(string calldata clusterId)
    external
    view
    override
    returns (
        bool status,
        uint256 creationDate,
        uint256 totalCapacity,
        uint256 utilizedCapacity,
        string memory _regionId,
        uint256 maxPrice
    )
```

Recommendation: Return `hwClass` as well.

ICN Protocol: Fixed in PR 192.

Cantina Managed: Fix verified.

3.5.4 `registerScalerNode` doesn't validate `hwClass`

Severity: Informational

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

Description: The `registerScalerNode` function allows an user to pass in non-existing `hwClass`. Depending on the behaviour of the off-chain part, this can cause either the collateral to be lost or allow a user to withdraw their `grantedCollateral` without actually providing any capacity to the network.

```
function registerScalerNode(
    string memory regionId,
    string memory name,
    uint256 hpId,
    uint256 capacity,
    LocationCode location,
    uint256 reservationPrice,
    string memory hwClass,
    uint256 nodeRewardShare,
    uint256 collateralAmount,
    uint256 commitmentDuration
) external override whenNotPaused {
```

Recommendation: Validate `hwClass` for existance.

ICN Protocol: Fixed in PR 198.

Cantina Managed: Fix verified.

3.5.5 `removeScalerNode` sets the timestamp of `nil regionId`

Severity: Informational

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

Description: `removeScalerNode` has to be invoked for rejected scalar nodes in-order to reclaim the collateral. When doing so, the clusterId and regionId of the node will be `nil`. Since `commitHpRewards` is always invoked, this will set the `hs.lastUpdatedTimestamp` of `nil regionId` to `block.timestamp`.

```
function removeScalerNode(uint256 scalerNodeId) external override whenNotPaused {
    ICNRegistryStorageData storage ds = getICNRegistryStorage();

    // Commit the rewards for the node's region and hwClass since the total capacity will be changed
    string memory clusterId = ds.scalerNodes[scalerNodeId].clusterId;
    string memory regionId = ds.clusters[clusterId].regionId;
    string memory hwClass = ds.scalerNodes[scalerNodeId].hwClass;
    IHPRewards(address(this)).commitHpRewards(regionId, hwClass);
```

Recommendation: Invoke `commtHpRewards` only if `ds.scalerNodes[scalerNodeId].status == ScalerNodeStatus.Validated`.

ICN Protocol: Fixed in PR 189.

Cantina Managed: Fix verified.