Quantstamp

Beam ERC721 Staking

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Staking, NFT, Validators
Timeline	2025-01-27 through 2025-02-02
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• neobase-one/icm-contracts #dc0918d
Auditors	• Gereon Mendler Auditing Engineer • Ibrahim Abouzied Auditing Engineer • Paul Clemson Auditing Engineer • Rabib Islam Senior Auditing Engineer • Tim Sigl Auditing Engineer • Yamen Merhi Auditing Engineer

Documentation quality	Medium
Test quality	Medium
Total Findings	14 Fixed: 12 Acknowledged: 2
High severity findings	6 Fixed: 6
Medium severity findings	3 Fixed: 2 Acknowledged: 1
Low severity findings	3 Fixed: 3
Undetermined severity findings	2 Fixed: 1 Acknowledged: 1
Informational findings	0

Summary of Findings

The Beam team adapted the validator contracts designed by Avalanche to support a multi-asset staking system with a unique new reward mechanism based on Euler reward streams. This allows users to submit stakes for new validators and delegate assets to existing validators, who are rewarded according to their share of the total staked amount. Freshly introduced features also include an unlock period before assets can be withdrawn and a direct redelegation option that avoids this delay. This combination of features unfortunately presents some novel logical issues that have not been well addressed yet. Particularly the new reward mechanism requires further consideration to be a viable alternative.

Fix-Review Update 2025-03-06:
The protocol has redesigned its reward mechanism. Rather than using the Euler reward streams integration, all rewards are now tracked natively in the contract. Token rewards are allocated to different epochs, and a permissioned submission of validator uptimes assigns rewards to validators and their delegates. Two additional issues were identified (BEAM-6 & BEAM-12), both of which were properly addressed along with all of the original issues.

ID	DESCRIPTION	SEVERITY	STATUS
BEAM-1	Delegations Can Become Locked if Validator Owner Cannot Receive Nfts	High	Fixed
BEAM-2	Excessive Gas Usage From Uptime Updates Allows Delegator-Based Dos Attack	High	Fixed
BEAM-3	Delegation of Completed Validators Continue Accruing Rewards	High	Fixed
BEAM-4	Incorrect Mapping Means Users Keep Control over Previously Delegated Nfts	High	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
BEAM-5	Validators Removing Delegations without `includeUptimeProof` Can Withhold Earned Rewards	High	Fixed
BEAM-6	NFTs May Be Lost in Contract/delegators May Steal NFTs Back	High	Fixed
BEAM-7	Validators Continue Accruing Rewards in `PendingRemoved` State	Medium	Acknowledged
BEAM-8	Delegators Continue Accruing Rewards in `PendingRemoved` State	Medium	Fixed
BEAM-9	Validators May Be Incentivized to Remove Delegations	Medium	Fixed
BEAM-10	Inconsistent Handling of Delegations: Nft Delegation Cannot Skip Unlock Period for Completed Validators	Low	Fixed
BEAM-11	Native Redelegation Not Removing Delegation Entries	Low	Fixed
BEAM-12	Validator Tokens May Exceed `$._maximumNFTAmount`	Low	Fixed
BEAM-13	Unclear Division Between Churn Period and Delegation Unlock Period	Undetermined	Acknowledged
BEAM-14	No Max Stake for Nft Delegations	Undetermined	Fixed

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.

2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

- contracts/validator-manager/ERC721TokenStakingManager.sol
- contracts/validator-manager/PoSValidatorManager.sol

Operational Considerations

Uptime updates: The Beam team aims to operate a bot that regularly updates the uptime for all validators each epoch so that rewards can be accurately distributed. As the rewards can differ between epochs, and the rewarded uptime increases linearly, the timing of the updates within each epoch can have consequences. If the uptime updates are not called exactly once at the beginning of each epoch, it may result in allocating weights to the incorrect epoch, rolling over uptime to the next epoch, or double-counting uptime. Double-counting may occur because rewards are rounded up to 100% if the validator has achieved an uptime of 80%, it is possible for the remaining 20% to roll over into the next epoch. So long as the proper uptime update schedule is maintained, this scenario should be avoided.

Reward supply: The security of the validator network is determined by the total size of the stakes, which are incentivized using the reward mechanism. If the rewards are too low, or the supply runs out, validators may withdraw their stakes, lessening the chain's security.

Reasonable parameters: The contracts require values for certain protocol parameters to be supplied during initialization. Some, like the minimum stake amount, duration, delegation fee, and unlock period length impact the incentive to become a validator or delegator. Setting the epoch period incorrectly can break the reward mechanism.

Key Actors And Their Capabilities

The **Owner** of the `Native721TokenStakingManager` can call:

- `submitUptimeProof()` and `submitUptimeProofs()`: The Owner is responsible for submitting uptime proofs on a timely schedule for accurate reward calculation. The Owner does not have the ability to modify the uptimes themselves.
- `registerRewards()`: The Owner is responsible for sending and allocating token rewards to the contract.
- `cancelRewards()`: The Owner may revoke rewards for a certain epoch *if the epoch has not yet begun.*

Findings

BEAM-1

Delegations Can Become Locked if Validator Owner Cannot Receive Nfts

• High Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `7ad19c5e644d6f2054be78d0f5bc97754dfe843e` .

File(s) affected: `ERC721TokenStakingManager.sol`

Description: When a validator ends their validation period, their NFTs are returned via `safeTransferFrom` in the `_unlockNFTs` function. If the validator owner's address cannot handle ERC-721 tokens, the completion of the validation period will revert. This leaves the validator status as `PendingRemoved`, permanently locking both the validator's and delegators' funds since there is no way to complete the validation period without successfully transferring the NFTs.

A malicious actor could exploit this by creating a validator smart contract that conditionally reverts in onERC721Received, allowing them to hold the delegator's funds hostage until certain conditions are met.

Furthermore, a similar issue applies to delegators, who can designate an owner other than themselves during the creation step. If the new owner is incapable of receiving ERC721 tokens, the delegation cannot be completed currently.

Recommendation: Avoid this issue by using a regular transfer, not `safeTransfer`. Since the tokens were originally staked by the same address, it should be able to hold NFTs. Further, consider setting all related delegations to the `pendingRemoved` state or allow the initialization of delegator withdrawal if the validator is in that state. Alternatively, add a fallback mechanism that prevents a validator's status from being stuck in `PendingRemoved` if the owner is unable to receive ERC-721 tokens.

BEAM-2

Excessive Gas Usage From Uptime Updates Allows Delegator-Based Dos Attack

High Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `13758fcc9d44d7b6f189f143292eda4bfa6c53a2` .

File(s) affected: `ERC721TokenStakingManager.sol`

Description: In the `ERC721TokenStakingManager` contract, there is no minimum delegation amount enforced for delegators. The amount required to obtain one validation weight is determined by `_weightToValueFactor`, which in the current configuration allows delegations with extremely small amounts (approximately $0.00000002 USD worth of BEAM tokens).

Currently, once a validator's uptime is updated, the contract calls `_balanceTrackerHook()` for every delegator and the validator, recalculating rewards in a single transaction. This design causes prohibitive gas costs, especially when the validator has many delegators. As the system scales, this approach becomes increasingly expensive and unmanageable.

This particularly leaves the system open to DoS attacks in which many trivially small delegations flood the delegations arrays, making it impossible to complete a call to `updateUptime()` .

Exploit Scenario:

1. Alice is running a validator and has several legitimate delegators
2. Malicious Bob notices there is no minimum delegation amount
3. Bob creates thousands of delegate transactions with minimal amounts (e.g., $0.00000002 USD each)
4. When Alice's validator tries to update their uptime to claim rewards, the transaction reverts due to the gas cost of processing all of Bob's delegations
5. Alice's validator is effectively frozen out of receiving rewards since uptime can't be updated

Recommendation: Implement a minimum delegation amount high enough to make such attacks economically unfeasible. Note that the runtime complexity of the account weight calculation is very high (cubic), therefore further changes may be necessary to avoid running into gas constraints.

BEAM-3

Delegation of Completed Validators Continue Accruing Rewards

High Fixed

Update

Original
Marked as "Fixed" by the client.
Addressed in: `274b6e44f049441ac0342b98bb2953bf98dd51f9` .
The client provided the following explanation:

```
Removed the _calculateAccountWeight() function, instead the total reward is cached and added/subtracted
during uptime update and reward calculation. When submitted an uptime with 0 increase in next epoch,
rewards for all the delegators to a completed validator is set to 0.
```

Fix-Review
Marked as "Fixed" by the client.
Addressed in: `48b1bcffc487ded0df54bfdf43e09c918e66a93d` .

With the new reward system, the delegators are now rewarded based on the minimum of the delegation and validator uptime:

```
uint64 delegationUptime = uint64(Math.min(delegationEnd - delegationStart, validationUptime));
```

File(s) affected: `contracts/validator-manager/ERC721TokenStakingManager.sol`

Description: When a validator ends, its delegation is set to complete in its next interaction with the protocol. However, if such interaction is avoided, a delegation can remain marked as “Active” even after its validator has become inactive.

Since the staking manager’s `_calculateAccountWeight()` logic doesn’t check the validation status for delegators, the delegation’s weight is inadvertently included in the user’s total account balance when it is calculated as part of other validator uptime updates. As a result, the user’s reported weight is inflated, leading to an over-allocation of rewards.

The following test highlights this issue:

```
function testDelegatorEarnsRewardsForInactiveValidator() public {
    bytes32 validationID1 = _registerDefaultValidator();
    bytes32 delegationID = _registerDefaultDelegator(validationID1);

    _endValidationWithChecks({
        validationID: validationID1,
        validatorOwner: address(this),
        completeRegistrationTimestamp: DEFAULT_REGISTRATION_TIMESTAMP,
        completionTimestamp: DEFAULT_REGISTRATION_TIMESTAMP + DEFAULT_EPOCH_DURATION,
        validatorWeight: DEFAULT_WEIGHT,
        expectedNonce: 2,
        rewardRecipient: address(this)
    });

    uint256 balanceBefore = _claimReward(DEFAULT_DELEGATOR_ADDRESS);
    console.log("CLAIM WHEN VALIDATION ENDS", balanceBefore);

    vm.warp(block.timestamp + DEFAULT_EPOCH_DURATION);

    uint256 balanceAfter = _claimReward(DEFAULT_DELEGATOR_ADDRESS);
    console.log("CLAIM ONE EPOCH LATER", balanceAfter);
}
```

Exploit Scenario:

1. User Delegates to Validator X. Validator X acquires some uptime, so the delegator's active weight is tracked.
2. User Delegates to Validator Y with more assets.
3. Validator X Ends. Even though the `ValidatorStatus` is "Completed," the user's delegation remains with `status == Active` in storage.
4. Validator Y Updates Uptime. The calls to `_calculateAccountWeight()` and `balanceTrackerHook()` sums all "active" delegations (including the ghost one on Validator X).
5. The user's total reported balance is higher than it should be and is thus over-rewarded from Validator Y's uptime distribution.

Recommendation: In `_calculateAccountWeight()`, set related delegations to the `pendingRemoved` state and omit them from account weight aggregation when a validator ends, or add a check that the delegation's parent validator is also `Active`.

BEAM-4

Incorrect Mapping Means Users Keep Control over Previously Delegated Nfts

High Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `274b6e44f049441ac0342b98bb2953bf98dd51f9`.
The client provided the following explanation:

```
same delegatorStakes mapping is now used for both regular and NFT delegations.
  Distinction is done using the _lockedNFTs (delegationID -> tokenIDs) mapping.
```

File(s) affected: `ERC721TokenStakingManager.sol`

Description: The function `_completeEndNFTDelegation()` is called when a user is in the process of ending their delegation, and commits state changes that reflect that the delegation has ended. However, the function mistakenly uses the wrong delegator mapping in the following lines:

```
Delegator memory delegator = $._delegatorStakes[delegationID];
delete $._delegatorStakes[delegationID];
```

Given that the `_delegatorStakes[delegationId]` entry will be empty, this will have no effect and will not revert either. This will cause the zero address to be passed as the owner to the `_removeNFTDelegationFromAccount()` function. Therefore, the delegation will not be removed from the `_accountNFTDelegations` mapping of the owner, meaning they will continue earning rewards after ending their delegation. Not only does this keep the user accruing rewards, but the NFTs are also unlocked for the user, who could now sell them. Furthermore, if a different user delegates these NFTs, the original user still has a `_delegatorNFTStakes[delegationID]` entry, allowing them to call `completeEndNFTDelegation()` and reclaim the NFTs for themselves.

Exploit Scenario:

1. Alice delegates NFT Token ID 1.
2. Alice calls `completeEndNFTDelegation()`.

3. The function does not delete her mapping `_delegatorNFTStakes`. Alice continues to accrue rewards. She is also returned the NFT.
4. Alice sells the token to Bob.
5. Bob delegates NFT Token ID 1.
6. Alice calls `completeEndNFTDelegation()` again. There are no guards preventing this, and the entry will still exists.
7. Bob loses his NFT.

Recommendation: Ensure that the `_completeEndNFTDelegation()` function correctly caches the `_delegatorNFTStakes` mapping.

BEAM-5
Validators Removing Delegations without `includeUptimeProof` Can Withhold Earned Rewards

High · Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `bdb5552a60f4f9cb254d61965c2e9949dd5c7791`.

File(s) affected: `PoSValidatorManager.sol`

Description: The `_initializeEndDelegation` function can be called by the validator to remove a delegation if the minimum stake duration has passed. Here, an optional uptime update can be included, which forces a recalculation of the account weights associated with this validator and updates the reward streams accordingly. The delegation is moved to the `pendingRemoved` state and can be redelegated or withdrawn after the unlock period. However, if the validator chooses to omit this update the delegation may be omitted from accurate reward calculations, especially if there has not yet been an update for the delegator owner account during the current epoch.

Recommendation: Consider removing the ability for validators to remove delegations, or force the inclusion of an uptime proof.

BEAM-6
NFTs May Be Lost in Contract/delegators May Steal NFTs Back

High · Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `e9f53e6e25692d11aa02d4f4012e8d7d62e466d8`, `32b771abecb7381d4ff751918aec149b885ec352`, and `0f5dbbf738aaead6c15b98a5ca2a2054978f49d5`.

File(s) affected: `Native721TokenStakingManager.sol`

Description: Delegations involving NFTs have the token IDs of the locked tokens recorded in `$._lockedNFTs`. This mapping is what is checked when determining which NFTs to unlock when an NFT delegation is completed. However, the corresponding mapping entry is only deleted in `_updateTime()`. This results in a severe accounting error with multiple repercussions:

1. If an active delegator calls `registerNFTRedelegation()` and, before `_updateTime()` can be called, calls `completeNFTDelegatorRemoval()`, then a new delegation is created with the original delegation's NFTs, but the same NFTs are sent back to the delegator due to completion of the delegation removal;
2. If someone decides to call `initiateNFTDelegatorRemoval()`, after which `_updateTime()` is called, then the delegation's corresponding `$._lockedNFTs` entry will be deleted because the delegation is no longer active, and the NFTs will be forever lost in the contract.

Recommendation: Implement the following changes:

1. Create a new state for delegations such that when `_completeNFTDelegatorRemoval()` is called, the delegation's state transitions to this new state—this will prevent the stealing of NFTs when a new delegation has been created with them;
2. Do not delete the `$._lockedNFTs` entry—this will prevent the loss of the NFTs.

BEAM-7
Validators Continue Accruing Rewards in `PendingRemoved` State

Medium · Acknowledged

Update
Original
Marked as "Fixed" by the client.
Addressed in: `274b6e44f049441ac0342b98bb2953bf98dd51f9`.
The client provided the following explanation:

```
Removed the _calculateAccountWeight() function, instead the total reward is cached and added/subtracted
during uptime update and reward calculation. Moreover now there is one source of truth one source of
```

```
truth tracking the weight of a validator.
```

Fix-Review Update
Under the new design, `updateUptime()` makes no checks regarding the validators status. The validator will accrue rewards regardless of their status, so long as their uptime has actually increased.

The client provided the following explanation:

```
A validator should still accrue rewards if the uptime has increased (doesn't matter if active or not).
```

File(s) affected: `ERC721TokenStakingManager.sol`

Description: When a validator signals their intent to end validation via the `initializeEndValidation()` function, the weight of the `validationID` in question is reduced to zero in the `_setValidatorWeight()` function.

However when calculating user rewards in the `_calculateAccountWeight()` function the `_validationPeriods` mapping is not checked, and instead the `_posValidatorInfo` mapping's weight is used, which is identical to the `startingWeight` value of the other mapping. As a result, users will continue to accrue rewards based on their initial stake even after their validator becomes inactive. Further, the `_calculateAccountWeight()` function does not check the state of the validator before including it in the aggregation.

Recommendation: Ensure there is only one source of truth tracking the weight of a validator and that it is updated correctly after each state transition. Either query the actual validator weight during the account weight calculations, or exclude validators without an active state.

BEAM-8
Delegators Continue Accruing Rewards in `PendingRemoved` State

Medium | Fixed

Update

Original
Marked as "Fixed" by the client.
Addressed in: `274b6e44f049441ac0342b98bb2953bf98dd51f9` .
The client provided the following explanation:

```
Removed the _calculateAccountWeight() function, instead the total reward is cached and added/subtracted
during uptime update and reward calculation.
  Fix-Review
  Marked as "Fixed" by the client.
  Addressed in:  48b1bcffc487ded0df54bfdf43e09c918e66a93d .
```

With the new reward system, the delegators now have their `endTime` tracked in `_initiateNFTDelegatorRemoval()` .

```
$._delegatorStakes[delegationID].endTime = uint64(block.timestamp);
```

File(s) affected: `PoSValidatorManager.sol` , `ERC721TokenStakingManager.sol`

Description: When a delegator calls the function to remove an active delegation (moving it to `PendingRemoved`), it still appears in the user's delegation list until `_completeEndDelegation()` is called and removes the delegation from `_accountDelegations[]` . As a result, the staking manager's weight-calculation logic (`_calculateAccountWeight()`) may continue counting this "pending removal" delegation as if it were fully active.

The code comments and original design suggest rewards should cease immediately after triggering delegation removal. `IPoSValidatorManager.initializeEndDelegation()` mentions the following:

```
For the purposes of computing delegation rewards, the delegation period is considered ended when
this function is called.
```

Furthermore, unlike the original design, a user cannot complete ending their delegation until `_unlockDelegateDuration` has elapsed, making the impact of acrruing rewards for a pending removal more pronounced. A user can initialize ending their delegation soon after creating it and continue accruing rewards, with the option to end it instantly at their convenience once the unlock duration has elapsed.

Exploit Scenario:
1. The user's delegation is marked Active, and it properly earns rewards.
2. The user calls `initializeEndDelegation()` soon after, and the system moves the delegation status to `PendingRemoved` .

3. The delegation remains in `_accountDelegations[user]` until a final call (`completeEndDelegation()`) occurs, so `_calculateAccountWeight()`
4. The user does not finalize removal, continuing to accrue rewards despite having nominally ended the delegation.
5. Long after the unlock duration has elapsed, the user waits for a convenient moment to complete ending their delegation.

Recommendation: There are a few possible solutions:

- Exclude `PendingRemoved` Delegations: In the weight-calculation routines, skip any delegation not in `Active` status.
- Immediate Cleanup: Remove or finalize a delegation from the user's `_accountDelegations[]` as soon as it transitions to `PendingRemoved`, preventing it from inflating rewards between initialization and completion.

BEAM-9 Validators May Be Incentivized to Remove Delegations

Medium Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `274b6e44f049441ac0342b98bb2953bf98dd51f9` .
The client provided the following explanation:

```
Removed the feature where validators can remove delegations. (for noth native and NFT delegations)
```

File(s) affected: `PoSValidatorManager.sol`

Description: Validators are able to remove delegators after the global minimum stake duration has elapsed. This reduces the overall stake of the validator network. Since rewards are distributed according to the share of the total stake, this increases the rewards of the remaining stakers, which include the validator. Additionally a one-time delegation fee is transferred to the validator whenever a new delegation is registered.

Therefore, the validator is incentivized to remove delegations as soon as possible. This can leave delegations in a `pendingRemoved` state limbo until the delegator takes further actions. This incentive goes directly against goal of a PoS validation system to accrue a massive total stake to financially secure the network.

Recommendation: Consider removing this feature or change the reward mechanism to pay out relative to the stake size, not as a share of a limited pool.

BEAM-10 Inconsistent Handling of Delegations: Nft Delegation Cannot Skip Unlock Period for Completed Validators

Low Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `274b6e44f049441ac0342b98bb2953bf98dd51f9` .
The client provided the following explanation:

```
Consistent behaviour is adopted where all validators, delegators and NFT delegators have to wait for
unlockDuration
```

File(s) affected: `ERC721ValidatorManager.sol`

Description: The contracts allow the delegation of either native token assets or ERC721 NFTs. When a validator ends and acquires the `Completed` state, native token delegations can be withdrawn in a single step, skipping the unlock period. However, this cannot be done with NFT delegations, which are forced to wait for the unlock period to elapse before they can be withdrawn. However, redelegations of NFTs can still be done immediately.

Further, if a validator completes while the delegation is `pendingRemoved` , the completion of the delegation still enforces the unlock period.

Recommendation: Decide which behavior is intended and adopt a consistent delegation flow.

BEAM-11 Native Redelegation Not Removing Delegation Entries

Low Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `8c342dae4f0a02cfec65a64d29440e82e24a8691` .

File(s) affected: `PoSValidatorManager.sol`

Description: `PoSValidatorManager.initializeRedelegation()` uses the funds of an existing delegation to create a new delegation. Though it deletes the old `_delegatorStakes[]` entry, it does not make calls to `_removeDelegationFromValidator(validationID, delegationID);` or `_removeDelegationFromAccount(delegator.owner, delegationID);`.

This leaves hanging references to old delegation entries. While no vulnerability was identified, these entries increase the attack surface as well as the gas costs of calculating an account's weight.

Recommendation: Add the following calls to `initializeRedelegation()`:

```
_removeDelegationFromValidator(validationID, delegationID);
_removeDelegationFromAccount(delegator.owner, delegationID);
```

BEAM-12 Validator Tokens May Exceed `$._maximumNFTAmount`

• Low Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `81635580e1239008d9dd97397fea4eb2ecdad9e6`.

File(s) affected: `Native721TokenStakingManager.sol`

Description: When registering a validator or a delegation with NFTs, it is checked that the number of NFTs staked to a given validator do not exceed a predetermined `_maximumNFTAmount`. However, the check does not get triggered when "redelegating" via `registerNFTRedelegation()`. Thus, it is possible to exceed the intended number of NFTs staked to a validator by first delegating to different validators, and then calling `registerNFTRedelegation()` to delegate tokens to the desired validator.

Recommendation: Check the `_maximumNFTAmount` while redelegating.

BEAM-13 Unclear Division Between Churn Period and Delegation Unlock Period

• Undetermined Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
We have this extra deleay from preventing the users from constantly moving their delegations to game
the reward system.
```

File(s) affected: `PoSValidatorManager.sol`

Description: A delegation cannot be completed until it has elapsed a minimum duration, as indicated by the following code in `_completeEndDelegation()`:

```
// To prevent churn tracker abuse, check that one full churn period has passed,
// so a delegator may not stake twice in the same churn period.
if (block.timestamp < delegator.startedAt + _getChurnPeriodSeconds()) {
    revert MinStakeDurationNotPassed(uint64(block.timestamp));
}
```

On top of requiring a minimum duration, the protocol will also hold the funds for a delegation unlock period **after** ending the delegation has been initialized, as seen in the following code in `completeEndDelegation()`:

```
if(block.timestamp < delegator.endedAt + $._unlockDelegateDuration) {
    revert UnlockDelegateDurationNotPassed(uint64(block.timestamp));
}
```

It is unclear why the protocol must both enforce that a delegation lasts a minimum period, as well as keep the funds locked after a sufficient delegation period. There is additional confusion given that users continue to accrue rewards whilst their delegation is in status `PendingRemoved` since the stake does not contribute to the security weight of the validator anymore.

Recommendation: Enforcing the churn period should be sufficient for protecting the protocol against MEV, and a length lock period may push away some users. Consider consolidating these two requirements into one.

BEAM-14 No Max Stake for Nft Delegations

• Undetermined Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `d85a3f86c739f8320e0cfc4ed9875873704d8e3d` .

File(s) affected: `ERC721ValidatorManager.sol`

Description: The protocol enforced a maximum native token stake amount for validators and delegators, forcing some distribution across multiple nodes. Contrary to that, similar limits are not enforced on NFTs. Therefore validators can receive an unlimited amount of NFT delegations.

While NFTs do not contribute to the security weight of the validator, other consequences may arise. Firstly, this leads to an unbounded size for the account weight aggregation calculation during the `_updateUptime()` function in its current form. Further, a higher share of rewards is dependent on a single validator. This could increase the power of its owner to do harm to the protocol or make the validator a target for other malicious actors. Additionally, a max NFT stake can incentivize the creation of new validators to facilitate further NFT delegations, which in turn increases the security factor of the network.

Recommendation: Enforce a maximum total validator stake amount for NFTs. Note that this maximum must be balanced according to the expected value of the NFTs, which is influenced by the rewards, and the required validator stake of native Beam tokens. Otherwise, NFTs may be used to blockade all available validators to limit the ability of other users to access the NFT reward stream pool.

Auditor Suggestions

S1 Remove Deprecated Reward Mechanism

Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `274b6e44f049441ac0342b98bb2953bf98dd51f9` .
The client provided the following explanation:

```
Removed the deprecated reward system
```

File(s) affected: `PoSValidatorManager.sol`

Related Issue(s): SWC-131, SWC-135

Description: The codebase contains many references to the old rewards mechanism. While this does not have a direct security threat, it increases gas costs, code readability, developer onboarding, and overall tech debt.

A non-exhaustive list of the dead code includes:

- `_withdrawValidationRewards()`
- `_withdrawDelegationRewards()`
- `_calculateAndSetDelegationReward()`
- `claimDelegationFees()`
- `_rewardCalculator.calculateReward()`
- `$._rewardRecipients[]`
- Duplicate code on `#L477`

Recommendation: We recommend removing any references to the old reward mechanism. The tests should also be updated to accurately reflect the intended reward mechanism.

S2 Add `initializeRedelegation()` to `IPoSValidatorManager`

Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `5935869d86010d47d9e0474eb9d5cbb5779afe76` .

File(s) affected: `interfaces/IPoSValidatorManager.sol`

Description: The `IPoSValidatorManager` is missing a declaration for `initializeRedelegation()` . We recommend adding an entry to complete the interface.

S3 Improve Error Messages for Non-Existent Validators

Acknowledged

> **Update**
> Marked as "Acknowledged" by the client.
> The client provided the following explanation:
>
> ```
> The error message is sufficient as only PoS validators will be in concern to be used with
> implementations of PoSStakingManager
> ```

File(s) affected: `PoSValidatorManager.sol`, `ERC721TokenStakingManager.sol`

Description: The contracts `PosValidatorManager` and `ERC721TokenStakingManager` do not properly distinguish between non-existent validators and non-PoS validators. When calling `getValidator()` with an invalid `validationID`, the function returns default values rather than reverting. The subsequent check `_isPoSValidator()` will fail with `ValidatorNotPoS` error since `owner` is `address(0)`, which does not accurately reflect that the validator does not exist.

Recommendation: Add a specific check and error message for non-existent validators to improve error handling clarity.

S4 Used Cached Storage Variables to Reduce Gas Consumption

Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `5935869d86010d47d9e0474eb9d5cbb5779afe76`.
> The client provided the following explanation:
>
> ```
> Removed the _calculateAccountWeight() function, instead the total reward is cached and added/subtracted
> during uptime update and reward calculation.
> ```

File(s) affected: `ERC721TokenStakingManager`

Description: Multiple times within the codebase unnecessary storage reads are made when it would be more gas efficient to first cache the variable in memory.

As an example, in `_calculateAmountWeight` the for loops make repeated calls to storage to read the `length` of the necessary array rather than first caching the value in memory. This greatly increases the gas consumption of the function as the length of the loop grows.

Recommendation: Consider making the recommended changes.

S5 Missing Input Validation

Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `42554d0d2f1e576eb353ceea2286012cf4846822`.

File(s) affected: `PoSValidatonManager.sol`

Description: Input validation of contract parameters is crucial for maintaining security and preventing invalid or malicious values from being processed which could lead to unexpected behavior or attacks. Without proper validation, contracts may operate on incorrect assumptions about their inputs, potentially leading to security vulnerabilities or contract malfunctions.

1. The owner of a delegator can be set to the zero address
2. `minimumNFTAmount` should be less than `maximumNFTAmount`
3. `epochDuration` should be within the limit designated by the reward streams contract.

Recommendation: Implement the described validation.

S6 Incorrect Natspec

Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `274b6e44f049441ac0342b98bb2953bf98dd51f9`.
> The client provided the following explanation:

> `The mentioned natspec comments and functions have been removed`

File(s) affected: `PoSValidatorManager`

Description: The natspec for the `_removeValidationFromAccount()` and `_removeDelegationFromAccount()` functions has been copied without being adapted and is therefore inconsistent with the actual behavior.

Recommendation: Make sure that natspec properly explains the function behaviors.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Files

- `994...e1f ./ERC721TokenStakingManager.sol`
- `3fe...3c0 ./validator-manager/PoSValidatorManager.sol`

Tests

- `79a...7b5 ./tests/ValidatorManagerTests.t.sol`
- `c61...9bd ./tests/ExamplesRewardCalculatorTests.t.sol`
- `9fa...cbe ./tests/NativeTokenStakingManagerTests.t.sol`
- `4bf...87a ./tests/ERC721TokenStakingManagerTests.t.sol`
- `63f...d41 ./tests/PoSValidatorManagerTests.t.sol`

Test Suite Results

```
Ran 110 tests for contracts/validator-
manager/tests/Native721TokenStakingManagerTests.t.sol:Native721TokenStakingManagerTest
[PASS] testCancelRewards() (gas: 45106)
[PASS] testCompleteDelegatorRegistration() (gas: 786063)
[PASS] testCompleteDelegatorRegistrationAlreadyRegistered() (gas: 798683)
[PASS] testCompleteDelegatorRegistrationForCompletedDelegatorRegistrationWhileValidatorPendingRemoved()
(gas: 941450)
[PASS] testCompleteDelegatorRegistrationImplicitNonce() (gas: 914447)
[PASS] testCompleteDelegatorRegistrationValidatorCompleted() (gas: 818295)
[PASS] testCompleteDelegatorRegistrationWrongNonce() (gas: 868258)
[PASS] testCompleteDelegatorRegistrationWrongValidationID() (gas: 739287)
```

```
[PASS] testCompleteEndDelegation() (gas: 1008972)
[PASS] testCompleteEndDelegationChurnPeriodSecondsNotPassed() (gas: 900316)
[PASS] testCompleteEndDelegationImplicitNonce() (gas: 1306245)
[PASS] testCompleteEndDelegationValidatorCompleted() (gas: 1017809)
[PASS] testCompleteEndDelegationValidatorPendingRemoved() (gas: 1054564)
[PASS] testCompleteEndDelegationWhileActive() (gas: 794020)
[PASS] testCompleteEndDelegationWrongNonce() (gas: 1238891)
[PASS] testCompleteEndDelegationWrongValidationID() (gas: 940025)
[PASS] testCompleteEndValidation() (gas: 653444)
[PASS] testCompleteEndValidationWithNonValidatorRewardRecipient() (gas: 656132)
[PASS] testCompleteInvalidatedValidation() (gas: 637822)
[PASS] testCompleteRegisterDelegatorValidatorPendingRemoved() (gas: 838294)
[PASS] testCompleteValidatorRegistration() (gas: 537908)
[PASS] testCompleteValidatorRegistrationUnauthorizedCaller() (gas: 12971)
[PASS] testCompleteValidatorRemovalUnauthorizedCaller() (gas: 13058)
[PASS] testCompleteValidatorWeightUpdateUnauthorizedCaller() (gas: 13237)
[PASS] testCumulativeChurnRegistration() (gas: 599597)
[PASS] testCumulativeChurnRegistrationAndEndValidation() (gas: 993873)
[PASS] testDefaultAndNFTDelegationRewards() (gas: 1607031)
[PASS] testDelegationFeeBipsTooHigh() (gas: 66527)
[PASS] testDelegationFeeBipsTooLow() (gas: 63611)
[PASS] testDelegationOverWeightLimit() (gas: 561825)
[PASS] testDelegationRewards() (gas: 1237321)
[PASS] testDelegationRewardsForSameValidatorAndDelegator() (gas: 1125210)
[PASS] testDelegationToPoAValidator() (gas: 50195)
[PASS] testDelegatorNFTRemoval() (gas: 831474)
[PASS] testDisableInitialization() (gas: 8129120)
[PASS] testDoubleDelegationRewards() (gas: 1483271)
[PASS] testEndDelegationNFTBeforeUnlock() (gas: 799508)
[PASS] testEndNFTDelegationRevertBeforeMinStakeDuration() (gas: 789845)
[PASS] testEndValidationPoAValidator() (gas: 180145)
[PASS] testForceInitiateDelegatorRemoval() (gas: 856141)
[PASS] testForceInitiateDelegatorRemovalInsufficientUptime() (gas: 856203)
[PASS] testForceInitiateValidatorRemoval() (gas: 581585)
[PASS] testForceInitiateValidatorRemovalInsufficientUptime() (gas: 585380)
[PASS] testInitialWeightsTooLow() (gas: 16635567)
[PASS] testInitializeEndValidation() (gas: 581619)
[PASS] testInitializeEndValidationNotOwner() (gas: 570589)
[PASS] testInitializeEndValidationWithoutNewUptime() (gas: 768880)
[PASS] testInitializeValidatorRegistrationExcessiveChurn() (gas: 257)
[PASS] testInitializeValidatorRegistrationExcessiveStake() (gas: 276)
[PASS] testInitializeValidatorRegistrationInsufficientDuration() (gas: 322)
[PASS] testInitializeValidatorRegistrationInsufficientStake() (gas: 234)
[PASS] testInitializeValidatorRegistrationPChainOwnerAddressesUnsorted() (gas: 145948)
[PASS] testInitializeValidatorRegistrationPChainOwnerThresholdTooLarge() (gas: 145313)
[PASS] testInitializeValidatorRegistrationSuccess() (gas: 597894)
[PASS] testInitializeValidatorRegistrationZeroPChainOwnerThreshold() (gas: 145329)
[PASS] testInitiateDelegatorRegistration() (gas: 665000)
[PASS] testInitiateDelegatorRegistrationInvalidAmount() (gas: 561111)
[PASS] testInitiateDelegatorRegistrationValidatorCompleted() (gas: 672596)
[PASS] testInitiateDelegatorRegistrationValidatorPendingRemoved() (gas: 604790)
[PASS] testInitiateDelegatorRemoval() (gas: 856183)
[PASS] testInitiateDelegatorRemovalMinStakeDurationNotPassed() (gas: 801066)
[PASS] testInitiateDelegatorRemovalNotRegistered() (gas: 677915)
[PASS] testInitiateDelegatorRemovalValidatorCompleted() (gas: 945549)
[PASS] testInitiateDelegatorRemovalValidatorPendingRemoved() (gas: 604770)
[PASS] testInitiateDelegatorRemovalWrongSender() (gas: 799476)
[PASS] testInitiateValidatorRegistrationUnauthorizedCaller() (gas: 27514)
[PASS] testInitiateValidatorRemovalUnauthorizedCaller() (gas: 12674)
[PASS] testInitiateValidatorWeightUpdateUnauthorizedCaller() (gas: 13403)
[PASS] testInvalidInitializeEndTime() (gas: 570576)
[PASS] testInvalidMinStakeDuration() (gas: 65857)
[PASS] testInvalidStakeAmountRange() (gas: 8178251)
[PASS] testInvalidTokenAddress() (gas: 8425893)
[PASS] testInvalidUptimeSenderAddress() (gas: 551363)
[PASS] testInvalidUptimeValidationID() (gas: 593122)
[PASS] testInvalidUptimeWarpMessage() (gas: 548235)
[PASS] testInvalidValidatorManager() (gas: 16256660)
[PASS] testMaxMinimumDelegationFee() (gas: 8179553)
[PASS] testMinStakeDurationTooLow() (gas: 8184035)
[PASS] testNFTDelegationOverWeightLimit() (gas: 2543108)
[PASS] testNFTDelegationRewards() (gas: 1221414)
```

```
[PASS] testNFTRedelegation() (gas: 1407396)
[PASS] testRedelegation() (gas: 1576544)
[PASS] testRemoveValidatorTotalWeight5() (gas: 16757010)
[PASS] testResendDelegatorRegistration() (gas: 688884)
[PASS] testResendEndDelegation() (gas: 879906)
[PASS] testResendEndDelegationWhileActive() (gas: 790618)
[PASS] testResendEndValidation() (gas: 598263)
[PASS] testResendRegisterValidatorMessage() (gas: 628017)
[PASS] testRevertDoubleCompletion() (gas: 927996)
[PASS] testRevertRemovalDelgationNFTForNonOwner() (gas: 908493)
[PASS] testRewardCancellationNonOwner() (gas: 15623)
[PASS] testRewardCancellationTooLate() (gas: 25101)
[PASS] testRewardRegistrationNonOwner() (gas: 15983)
[PASS] testRewardsTooEarly() (gas: 828070)
[PASS] testStakeAmountTooHigh() (gas: 70088)
[PASS] testStakeAmountTooLow() (gas: 67906)
[PASS] testStakingManagerStorageSlot() (gas: 10188)
[PASS] testSubmitUptimeNonOwner() (gas: 546459)
[PASS] testSubmitUptimeProofPoaValidator() (gas: 13955)
[PASS] testSubmitUptimes() (gas: 1224328)
[PASS] testSubmitUptimesInvalidInput() (gas: 545088)
[PASS] testUnsetValidatorManager() (gas: 8175310)
[PASS] testValidationRegistrationWithoutNFT() (gas: 46635)
[PASS] testValidatorManagerStorageSlot() (gas: 10145)
[PASS] testValueToWeight() (gas: 17059)
[PASS] testValueToWeightExceedsUInt64Max() (gas: 12585)
[PASS] testValueToWeightTruncated() (gas: 12501)
[PASS] testWeightToValue() (gas: 17184)
[PASS] testZeroMinimumDelegationFee() (gas: 8178398)
[PASS] testZeroWeightToValueFactor() (gas: 8183488)
Suite result: ok. 110 passed; 0 failed; 0 skipped; finished in 244.53ms (1.63s CPU time)
```

Code Coverage

The coverage data was gathered by running forge coverage.

We highly recommend increasing the Branch coverage metric, ideally to 100%. While this metric is an important indicator of security, we would like to note that, even at a score of 100%, it is essential to view it as one component of a comprehensive security strategy.

File	% Lines	% Statements	% Branches	% Funcs
contracts/validator-manager/Native721TokenStakingManager.sol	95.61% (196/205)	96.38% (266/276)	82.50% (33/40)	89.29% (25/28)
contracts/validator-manager/StakingManager.sol	75.21% (182/242)	75.84% (226/298)	52.73% (29/55)	86.36% (19/22)
contracts/validator-manager/ValidatorManager.sol	82.41% (164/199)	86.61% (207/239)	33.33% (13/39)	89.29% (25/28)

Changelog

- 2025-02-04 - Initial report
- 2025-03-06 - Fix Review

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and and may not be represented as such. No third party is entitled to rely on the report in any any way, including for the purpose of making any decisions to buy or sell a product, product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or or any open source or third-party software, code, libraries, materials, or information to, to, called by, referenced by or accessible through the report, its content, or any related related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.