OpenZeppelin | security

Ava Cloud Staking Manager Audit

Ava Labs.

June 27, 2025

Table of Contents

Summary

Type	Staking
Timeline	From 2025-06-09 To 2025-06-13
Languages	Solidity

Total Issues	10 (10 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	2 (2 resolved)
Low Severity Issues	2 (2 resolved)
Notes & Additional Information	5 (5 resolved)
Client Reported Issues	1 (1 resolved)

Scope

OpenZeppelin audited the ava-labs/ac-staking-contracts repository at commit 26e4e9b.

In scope were the following files:

```
ac-staking-contracts
└── src
    ├── ERC20LicensedStakingManager.sol
    ├── LicensedStakingManager.sol
    ├── NativeTokenLicensedStakingManager.sol
    ├── StakingManager.sol
    └── interfaces
        ├── IERC20LicensedStakingManager.sol
        ├── ILicensedStakingManager.sol
        └── INativeTokenLicensedStakingManager.sol
```

Update: *All resolutions and the final state of the audited codebase mentioned in this report are contained at commit ee8bcdc.*

System Overview

AvaCloud is a fully managed, no-code blockchain platform developed by Ava Labs, designed to simplify the deployment of custom L1 blockchains on the Avalanche network. It enables businesses to launch scalable, high-performance blockchain networks without requiring deep technical expertise or significant infrastructure investment.

The AvaCloud staking manager contracts consist of smart contracts that manage the registration and termination of L1 validators and their delegators, as outlined in ACP-77. ACP-77 defines the mechanisms for creating subnets, transitioning them to an open proof-of-stake consensus model, and establishing communication between these subnets and the P-chain, which serves as the source of truth for L1 validators. These contracts allow staking for both delegators and validators using ERC-721 node license NFTs and ERC-20 tokens.

The core `StakingManager` abstract contract provides a foundation for validator lifecycle management, including registration, uptime proof submission, removal, and reward payout, with mechanisms to lock and unlock tokens securely using upgradeable patterns. Extended variants such as `NativeTokenLicensedStakingManager` and `ERC20LicensedStakingManager` incorporate native token and ERC-20 token staking respectively, along with license token NFTs (ERC-721) as a requirement for validator and delegation registration. The `LicensedStakingManager` abstract contract, which extends the `StakingManager` contract, is responsible for ERC-721 license token management-related operations such as calculating node license NFTs' staking amount and maintaining mappings between token IDs, validation IDs, and delegation IDs. Licensing logic, including token locking, credentials mapping, and stake calculation via conversion factors, is a key aspect, aiming to enforce licensing requirements in validator operations.

The `StakingManager` abstract contract is a modified version of the Avalanche ICM Validator Manager v2.0.0 directory's `StakingManager` contract to support node licenses. The `StakingManager` will act as the owner of the `ValidatorManager` contract, which is responsible for parsing and sending messages to the P-chain using the Avalanche Warp Messaging (AWM). Transferring the ownership of the `ValidatorManager` to the `StakingManager` contract is considered a migration from a proof-of-authority (PoA) to a proof-of-stake (PoS) consensus mechanism.

The `ValidatorManager` and `RewardCalculator` contracts, which are called by the `StakingManager` contract, are out of scope for this audit but were previously incrementally audited by OpenZeppelin at commit 8a8f78d.

Security Model and Trust Assumptions

This audit assumes that the following components and conditions function correctly:

- The `ValidatorManager` and `RewardCalculator` contracts function as documented in the ICM contracts directory. The churn tracking mechanism and its pitfalls in the `ValidatorManager` contract are taken into consideration during deployments, setting initial validator weights, and migration to the proof-of-stake mechanism.
- After transitioning from PoA to PoS, the system allows anyone to disable PoA validators, enforcing decentralization and reducing reliance on trusted actors. L1 operators on AvaCloud are assumed to handle this migration as per documentation and with sufficient stake, ensuring bad actors do not gain control of the L1.
- ICM messages are delivered securely and reliably, with off-chain relayers properly incentivized to forward messages between the P-chain and the L1.
- The `uptimeBlockchainID` initialized in the `StakingManager` is correct and must be validated by the L1 validator set that the contract manages.
- The uptime mechanism operates as intended, with accurate data delivered and verified across chains. Validators only sign valid ICM messages, ensuring that messages reflect accurate and authorized validator activity.
- The P-chain enforces a nonce-based replay protection mechanism and will only accept `L1ValidatorWeightMessage` messages that contain a nonce equal to or greater than the current `minNonce` tracked per validator. This prevents replays and ensures that validator weight updates occur in a consistent and monotonic fashion.
- The `weightToValueFactor` will be appropriately set based on the ERC-20 token's decimals, ensuring that significant dust amounts are not lost to the contract upon staking. For more details on this behavior, see the explanation.

Privileged Roles

- The proxy owner of both the `ValidatorManager` and the associated `StakingManager` contracts (`ERC20TokenStakingManager` or `NativeTokenStakingManager`) retains upgradeability control. These proxy owners are trusted entities whose actions can directly impact the security and behavior of the validator and staking logic.
- The `StakingManager` and `ValidatorManager` contracts may reside on a chain that is not validated by the L1 that the contract is managing. However, it is essential for the deployer of the contracts to ensure that the license tokens, as well as staking and reward tokens, reside on the same chain as the contracts to ensure consistent and secure staking operations.
- The node license NFT contract's admin roles are trusted entities who may have the ability to pause or upgrade the contract. Similarly, ERC-20 tokens are expected to function as standard ERC-20 tokens without any fees-on-transfer logic.

Medium Severity

M-01 Incorrect Key Used When Removing Delegation from `ValidatorDelegations`

In the `_returnDelegatorTokens` function, after returning the delegator's ERC-721 tokens, the contract attempts to remove the delegation record from the `_validatorDelegations` mapping. However, the mapping is defined to associate a validator's `validationID` with its `delegationIDs`, but it uses the `delegationID` as the key. This means that the intended validator's delegation list is not properly updated, leading to stale or inconsistent delegation records.

Consider updating the `_returnDelegatorTokens` function to retrieve the validator's `validationID` associated with the `delegationIDs`. Then, use that as the key to properly remove the delegation from `ValidatorDelegations`.

Update: *Resolved in pull request #13. The Ava Labs team stated:*

> *We updated the `_unlock` function to take one more `validationID` argument. It is set to `bytes32(0)` for validators.*

M-02 Inefficient Array Handling in Delegation Management

The `_returnDelegatorTokens` function is designed to process and clear delegations through the use of the `delete` operator on the `_validatorDelegations` mapping that has an array value data type within a for-loop. This approach effectively removes the specified elements from the array but does not reorganize or compact the array afterward. As a result, the array is left with "gaps" of default values where the deleted elements once resided.

This handling of the array can lead to inefficiencies, particularly as the array grows in size over time. Specifically, operations that require iterating over `_validatorDelegations`, such as looping to process or validate delegations, will encounter these gaps. Processing these gaps does not contribute to the function's logic but incurs computational overhead, thus leading to increased gas costs for transactions involving this function.

Moreover, as the array becomes increasingly sparse due to repeated deletions without recompaction, the cost of operations on this array could escalate, potentially making it expensive to perform delegation-related tasks. This not only affects the efficiency of the smart contract but could prevent its use for some delegators in the future.

Consider updating the `_returnDelegatorTokens` function to properly handle the rearrangement of the `_validatorDelegations` mapping's array value data type, such that it prevents any gaps.

Update: *Resolved in pull request #4.*

Low Severity

L-01 Setting `licenseToStakeConversionFactor` Variable High Could Lead to Churn Restriction Issues

The `ValidatorManager` contract, called by the `StakingManager` contract, includes a churn limit mechanism, which restricts the total weight that can be added or removed within a given period.

According to the documentation, a scenario may arise where "a PoS validator manager with a high proportion of the L1's weight is not able to exit or enter the validator set due to churn restrictions. Additional validators or delegators will need to be registered to more evenly distribute weight across the L1 validator set."

These churn limitations depend on the current total validator weight. Over time, if the total validator weight falls below (licenseToStakeConversionFactor × 100) / (weightToValueFactor × churnPercent), no new validators or delegators can be added to the system since even one NFT would require a minimum stake amount equal to `licenseToStakeConversionFactor`.

For example:

Assume churnPercent is 20%, the current total weight is 1000e6, `weightToValueFactor` is 1e12, and `licenseToStakeConversionFactor` is 100e18.

If the total weight drops below 500e6, no new registrations can occur because staking a single NFT requires 100e18 (plus 1e12 for the ERC-20 token requirement), which exceeds the 20% churn limit of the current weight (500e6 × 20% = 100e6, i.e., 100e18 stake amount).

This issue is similar to the design pitfall mentioned in the documentation. However, once the system reaches this state, no new validators or delegators can be added, even to contribute smaller proportions of stake—contradicting the documentation's mitigation suggestion. This effectively blocks the system.

Consider documenting this scenario when configuring the `licenseToStakeConversionFactor` variable and ensuring initial weights are set correctly so that this edge case is never achieved unless L1 becomes inactive. Additionally, having mitigation steps in place to restore the system externally is advisable.

Update: *Resolved in pull request #5 and pull request #14. The Ava Labs team stated:*

> *We decided to allow registering new delegators without a license. An explanation of the potential churn issues has been added to `CONSIDERATIONS.md`. In addition, the AvaCloud team will work on preparing recommended configuration options to ensure a safe transition to PoS for the clients.*

L-02 Missing and Incomplete Docstrings

Certain instances across the codebase lack documentation. This hinders reviewers' understanding of the code's intention, which is fundamental to correctly assess not only security but also correctness. Additionally, docstrings improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned, and the events emitted. Instances of such were identified:

- In `ERC20LicensedStakingManager.sol`, the return value in `erc20` function
- In `IERC20LicensedStakingManager.sol`, the return value in `erc20` function, the `ERC20TokensUnlocked` event
- In `INativeTokenLicensedStakingManager.sol`, the `NativeTokensUnlocked` event

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

Update: *Resolved in pull request #6.*

Notes & Additional Information

N-01 Missing Reentrancy Warning in `NativeTokenLicensedStakingManager`'s `_unlock` Function

The `NativeTokenLicensedStakingManager` contract contains an internal function named `_unlock`, tasked with releasing license tokens and native stakes back to users. This function utilizes OpenZeppelin's `Address` library's `sendValue` method to transfer native tokens. The `ERC20LicensedStakingManager` contract, which manages ERC-20 staked tokens and license tokens, features a similar `_unlock` function. Notably, the latter includes a docstring cautioning against potential reentrancy attacks when invoking it, specifically advising the use of a reentrancy guard due to the inherent risks associated with safe transfer operations.

However, the `_unlock` function within `NativeTokenLicensedStakingManager` lacks this critical docstring, despite being equally susceptible to reentrancy risks. This omission is particularly concerning given the explicit warning in the `Address` library's documentation, which underscores the importance of exercising caution to prevent reentrancy vulnerabilities, suggesting the adoption of the `ReentrancyGuard` or the checks-effects-interactions pattern as preventive measures.

Consider incorporating a warning docstring for `NativeTokenLicensedStakingManager`'s `_unlock` function similar to that found in the `ERC20LicensedStakingManager`'s `_unlock` function.

Update: *Resolved in pull request #7.*

N-02 For Loops Can Be Gas Efficient

Throughout the codebase, multiple instances where `for` loops can be optimized were identified:

- Within the `_returnValidatorTokens` function, the loop iterates over the length of the `validatorTokens` storage variable, which can be cached into a memory variable.
- Within the `_returnDelegatorTokens` function, the loop iterates over the length of the `delegatorTokens` storage variable, which can be cached into a memory variable.
- Within the `_returnDelegatorTokens` function, the second loop iterates over the length of the `_validatorDelegations` storage variable, which can be cached into a memory variable.
- All instances of the `i` variable in `for` loops are initiated with a default value. Skipping initialization of these variables will save gas.
- The prefix increment operator `++i` instead of the post-increment operator `i++` can be applied to all `for` loop instances.

Consider implementing the above optimizations to make the `for` loops more gas efficient.

Update: *Resolved in pull request #8.*

N-03 Missing Named Parameters in Mapping

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

In the `_validatorDelegations` state variable of the `LicensedStakingManager` contract, the mapping does not have any named parameters. Similarly,

- The `_validatorStakedTokens` state variable
- The `_delegatorStakedTokens` state variable
- The `_tokenToValidator` state variable
- The `_tokenToDelegation` state variable
- The `_validatorDelegations` state variable

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

Update: *Resolved in pull request #9.*

N-04 Use `calldata` Instead of `memory`

When dealing with the parameters of `external` functions, it is more gas-efficient to read their arguments directly from `calldata` instead of storing them in `memory`. `calldata` is a read-only region of memory that contains the arguments of incoming `external` function calls. This makes using `calldata` as the data location for such parameters cheaper and more efficient compared to `memory`. Thus, using `calldata` in such situations will generally save gas and improve the performance of a smart contract.

Throughout the codebase, multiple instances where function parameters should use `calldata` instead of `memory` were identified:

- In `ERC20LicensedStakingManager.sol`, the `nodeID` parameter
- In `ERC20LicensedStakingManager.sol`, the `blsPublicKey` parameter
- In `ERC20LicensedStakingManager.sol`, the `remainingBalanceOwner` parameter
- In `ERC20LicensedStakingManager.sol`, the `disableOwner` parameter
- In `NativeTokenLicensedStakingManager.sol`, the `nodeID` parameter
- In `NativeTokenLicensedStakingManager.sol`, the `blsPublicKey` parameter
- In `NativeTokenLicensedStakingManager.sol`, the `remainingBalanceOwner` parameter
- In `NativeTokenLicensedStakingManager.sol`, the `disableOwner` parameter

Consider using `calldata` as the data location for the parameters of `external` functions to optimize gas usage.

Update: *Resolved in pull request #10.*

N-05 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it allows the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, there are contracts that do not have a security contact:

- The `ERC20LicensedStakingManager` contract
- The `LicensedStakingManager` abstract contract
- The `NativeTokenLicensedStakingManager` contract
- The `IERC20LicensedStakingManager` interface
- The `ILicensedStakingManager` interface
- The `INativeTokenLicensedStakingManager` interface

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

Update: *Resolved in pull request #11.*

Client Reported

CR-01 Duplicate Entries Allowed in `_validatorDelegations` Mapping

The `_validatorDelegations` mapping stores an array of `delegationId` values for a particular `validationId` key to track all the delegations for a validator. However while initiating registration, in the `_lockERC721s` function the `delegationId` is pushed into the array for every node license NFT that a delegator intends to stake through the loop. This creates duplicate entries with the same `delegationId` in the array for the particular validator if there are multiple NFTs to stake.

The Ava labs team resolved this issue with pull request #5 by pushing the `delegationId` value into the array once and outside the for loop.

Update: *Resolved in pull request #5.*

Conclusion

The AvaCloud Staking Manager contract extends the functionality of Ava Labs' ICM Validator Manager system to support ERC-721 Node Licenses, which are distributed via node sales. The audit focused on this extended functionality to ensure the robustness of the system. Multiple medium- and low-severity issues were identified, along with opportunities to improve gas efficiency. The codebase was well written and comprehensively documented. Throughout the audit, the Ava Labs team was highly responsive, providing valuable insights into the modifications made to the existing contracts.