Plume Staking

Security Assessment

May 29th, 2025 — Prepared by OtterSec

Nicholas R. Putra nicholas@osec.io

Zhenghang Xiao kiprey@osec.io

Robert Chen r@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

Plume Network engaged OtterSec to assess the `plume-staking-contracts` program. This assessment was conducted between May 1st and May 29th, 2025. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 6 findings throughout this audit engagement.

In particular, we identified a critical-risk vulnerability in the logic. Unstaking after the cooldown ends resets the previously unlocked (cooled) amount, resulting in users losing unclaimed tokens (OS-PST-ADV-00). Additionally, we found an improper restaking implementation that might allow users to stake more than the validator capacity and trigger inconsistency in the stored state (OS-PST-ADV-01). We also found an incorrect commission calculation while handling the new staking position case (OS-PST-ADV-02).

Furthermore, there are missing validation checks and state updates in the staking-related functions that allow users to stake less than the minimum amount (OS-PST-ADV-03). We also found an improper in-place removal that might cause array elements to be skipped during the loop (OS-PST-ADV-04). Finally, we made suggestions to ensure adherence to coding best practices (OS-PST-SUG-00).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/plumenetwork/contracts. This audit was performed against commit d0fb773.

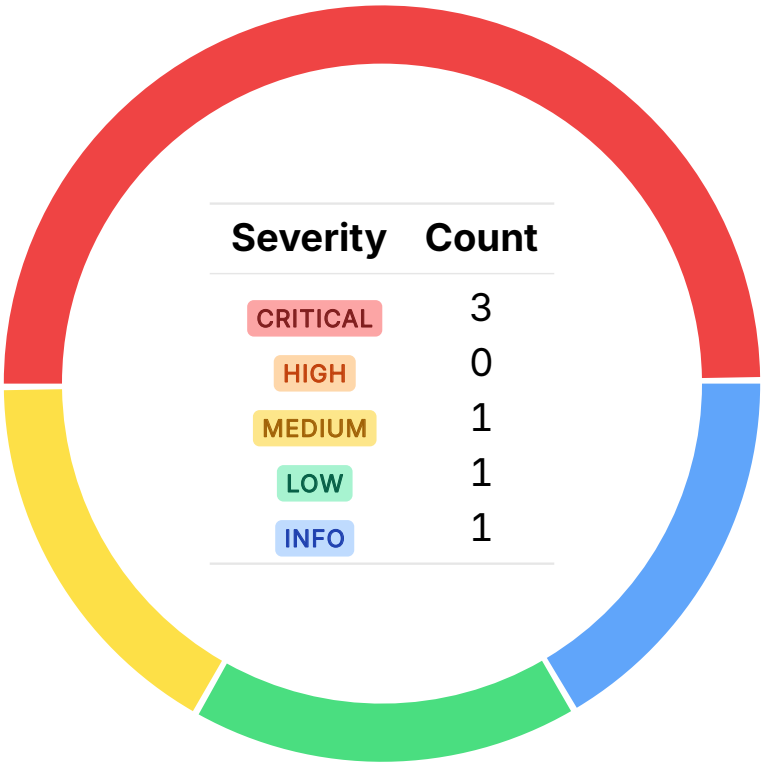
A brief description of the program is as follows:

Name	Description
plume-staking-contracts	An upgradeable staking system built using the Diamond Proxy (Facet-based architecture) and integrates with a dedicated reward treasury to manage and distribute staking incentives.

03 — Findings

Overall, we reported 6 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
CRITICAL	3
HIGH	0
MEDIUM	1
LOW	1
INFO	1

04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-PST-ADV-00	CRITICAL	RESOLVED	Unstaking after the cooldown ends resets the previously unlocked (`cooled`) amount, resulting in users losing unclaimed tokens.
OS-PST-ADV-01	CRITICAL	RESOLVED	The current implementation of `restake` function lacks validator capacity limit validation and contains missing logic that leads to invalid reward states.
OS-PST-ADV-02	CRITICAL	RESOLVED	The `_performStakeSetup` function incorrectly handles commission calculation for new stakes by updating stake amounts before updating reward state.
OS-PST-ADV-03	MEDIUM	RESOLVED	The current implementation of `StakingFacet` lacks enforcement of validation checks, proper state updates, and redundant code instances.
OS-PST-ADV-04	LOW	RESOLVED	The `withdraw` functions may skip processing array elements due to improper handling of `userValidators` removal.

Improper Reset of Cooled Amount CRITICAL OS-PST-ADV-00

Description

In `StakingFacet::_unstake`, if a user's previous cooldown period has ended (`globalInfo.cooldownEnd != 0` and `block.timestamp >= globalInfo.cooldownEnd`), their cooled amount is overwritten by the newly unstaked amount. This creates an issue where any previously unlocked (but unclaimed) `PLUME` tokens are effectively lost. Instead of accumulating the new unstaked amount on top of the existing cooled amount, the function resets `globalInfo.cooled` and starts a fresh cooldown. Users who forget to withdraw their cooled tokens before unstaking again will lose their previous cooled balance

Remediation

Accumulate cooled amounts if the previous cooldown has already expired.

Patch

Resolved in 892661c.

Improper Restaking Mechanism CRITICAL OS-PST-ADV-01

Description

The `restake` function is used to restake a user's `cooling` amount back into PLUME. However, the function does not validate whether the restaking amount would exceed the validator's capacity limit. Additionally, the restaking process fails to properly initialize the reward state when the restake represents a new stake.

>_ *plume/src/facets/StakingFacet.sol* SOLIDITY

```
function _performRestakeWorkflow(
    address user,
    uint16 validatorId,
    uint256 amount,
    string memory fromSource
) internal {
    // Use consolidated validation
    _validateStaking(validatorId, amount);

    // Update rewards before any balance changes
    PlumeRewardLogic.updateRewardsForValidator(PlumeStakingStorage.layout(), user, validatorId);

    // Update stake amounts
    _updateStakeAmounts(user, validatorId, amount);

    // Ensure staker is properly listed for the validator
    PlumeValidatorLogic.addStakerToValidator(PlumeStakingStorage.layout(), user, validatorId);
}
```

Furthermore, the current implementation of `restake` could reuse the `stake` logic to ensure consistent staking behavior.

Remediation

Ensure that `restake` follows the implementation defined in `stake` to maintain consistency.

Patch

Resolved in 8bb6933.

Incorrect Commission Calculation for New Stakes CRITICAL OS-PST-ADV-02

Description

When a user creates a new stake position with a validator, the `_performStakeSetup` function incorrectly updates the stake amounts before updating the validator's reward state by calling the helper function `_initializeRewardStateForNewStake`. This causes the validator to incorrectly receive commission on the newly staked amount, rather than having the reward state updated based on the previous total staked amount.

>_ *plume/src/facets/StakingFacet.sol* SOLIDITY

```
function _performStakeSetup(
    address user,
    uint16 validatorId,
    uint256 stakeAmount
) internal returns (bool isNewStake) {
[...]
    // Check if this is a new stake for this specific validator
    isNewStake = $.userValidatorStakes[user][validatorId].staked == 0;

    // If user is adding to an existing stake with this validator, settle their current rewards
        ↪  first
    if (!isNewStake) {
        PlumeRewardLogic.updateRewardsForValidator($, user, validatorId);
    }

    // Update stake amount
    _updateStakeAmounts(user, validatorId, stakeAmount);
[...]
    // Initialize reward state for new stakes
    if (isNewStake) {
        _initializeRewardStateForNewStake(user, validatorId);
    }
}
```

Remediation

For new stakes, update the validator's reward state before updating the stake amounts. This ensures commission is calculated correctly based on the previous total staked amount.

Patch

Resolved in f9a500b

Discrepancies in StakingFacet Implementation MEDIUM OS-PST-ADV-03

Description

`StakingFacet` lacks limit enforcement, fails to update protocol state in some cases, and contains redundant logic that should be consolidated. Specifically, `restake` do not enforce a `minStakeAmount`, allowing users to restake arbitrarily small amounts.

Also, `stakeOnBehalf` currently updates the user's stake and the validator's `delegatedAmount`, but fails to update `validatorTotalStaked`, which tracks the total staked to each validator, and `totalStaked`, which tracks the total `PLUME` staked across all validators. This results in inconsistencies between global and per-validator staking records.

>_ *plume/src/facets/StakingFacet.sol* SOLIDITY

```
function stakeOnBehalf(uint16 validatorId, address staker) external payable returns (uint256) {
    PlumeStakingStorage.Layout storage $ = _getPlumeStorage();
    uint256 stakeAmount = msg.value;
    if (stakeAmount < $.minStakeAmount) {
        revert StakeAmountTooSmall(stakeAmount, $.minStakeAmount);
    }
    if (!$.validatorExists[validatorId]) {
        revert ValidatorNotActive(validatorId);
    }
    if (staker == address(0)) {
        revert ZeroRecipientAddress();
    }
    [...]
}
```

Furthermore, the current implementations of `stake`, `restake`, and `stakeOnBehalf` share substantial overlapping logic affecting efficiency and maintainability.

Remediation

Ensure that `restake` enforces the `minStakeAmount`, and that `stakeOnBehalf` updates both `validatorTotalStaked` and `totalStaked`. Additionally, consolidate the redundant code in `stake`, `restake`, and `stakeOnBehalf` to improve maintainability and clarity.

Patch

Resolved in dd5df2e.

Array Elements Skipped Due To In-Place Removal LOW OS-PST-ADV-04

Description

In the current implementation of `withdraw`, there is logic that loops through `userValidators` while also removing validators that are no longer needed.

However, since the for-loop directly references the modified `userValidators` array, removing elements during iteration causes elements that are swapped into removed positions to be skipped. This happens because when an element is removed, the last element is swapped into its position, but the loop counter continues forward, missing the newly swapped element.

Remediation

Create a memory copy of `userValidators` to iterate over instead of directly accessing and modifying the storage array during iteration. This ensures all validators are properly processed regardless of removals.

Patch

Resolved in d408f63.

05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-PST-SUG-00	Suggestions regarding inconsistencies in the codebase and ensuring adherence to coding best practices.

Code Maturity OS-PST-SUG-00

Description

1. The comment in `adminWithdraw` explicitly states that the function requires `ADMIN_ROLE`, implying that only addresses with this role should be allowed to call `adminWithdrawstates`. However, the code currently enforces `TIMELOCK_ROLE`. Update the code to reflect the comment.

>_ *plume/src/facets/ManagementFacet.sol* SOLIDITY

```
/**
 * @notice Allows admin to withdraw ERC20 or native PLUME tokens from the contract balance
 * @dev Primarily for recovering accidentally sent tokens or managing excess reward funds.
 * Requires ADMIN_ROLE.
 * @param token Address of the token to withdraw (use PLUME address for native token)
 * @param amount Amount to withdraw
 * @param recipient Address to send the withdrawn tokens to
*/
function adminWithdraw(
    address token,
    uint256 amount,
    address recipient
) external onlyRole(PlumeRoles.TIMELOCK_ROLE) nonReentrant {
    // <-- Use ADMIN_ROLE
    [...]
}
```

2. The `stake` function does not set `$.userValidatorStakeStartTime` when staking for the first time.
3. The `restake` comment states that the restaking mechanism will take from the user's `cooling` and `parked` amounts, yet the current implementation only takes from the `cooling` amount.
4. The `restake` function does not clean up validator relationships in cases where it consumes the entire cooled amount from other validators, leaving no remaining involvement with those validators.
5. The `_processMaturedCooldowns` `$.userValidators` array copy can be simplified to a direct assignment instead of iteratively copying through a loop.

Remediation

Implement the above-mentioned suggestions.

Patch

Resolved in 3a6c78c.

A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

CRITICAL

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

HIGH

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

MEDIUM

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.