Plume Staking v2

Security Assessment

July 8th, 2025 — Prepared by OtterSec

Nicholas R. Putra nicholas@osec.io

Zhenghang Xiao kiprey@osec.io

Robert Chen r@osec.io

Table of Contents

01 — Executive Summary

Overview

Plume Network engaged OtterSec to assess the `plume-staking-contracts` program. This assessment was conducted between May 1st and July 8th, 2025. For more information on our auditing methodology, refer to chapter 07.

Key Findings

We produced 28 findings throughout this audit engagement.

In particular, we identified several critical vulnerabilities in the staking and rewards system. Most notably, we found that new stakers could immediately claim unearned rewards due to improper reward tracking variable updates during staking operations (OS-PLM-ADV-00). Additionally, the claim functionality exhibited a critical flaw where it failed to update the global reward state before processing claims, enabling users to repeatedly claim more rewards than entitled by exploiting stale cumulative values (OS-PLM-ADV-01).

Furthermore, the commission system also showed precision-related issues, where rounding down in commission calculations could prevent validators from claiming their full commission due to discrepancies between tracked and actual commission amounts (OS-PLM-ADV-10). We also identified that the user-validator relationship might be prematurely removed due to `hasPendingReward` being cleared despite users having unclaimed rewards from removed `rewardToken` s (OS-PLM-ADV-11). The `claim` validation logic does not properly check whether users have pending rewards, preventing them from claiming rewards for removed `rewardToken` s (OS-PLM-ADV-12).

Moreover, the `claim` function cannot be used to claim rewards from slashed validators and does not clear the user-validator relationship state (OS-PLM-ADV-16). The `restakeRewards` function also fails to properly clean up pending rewards flags and user-validator relationships after claiming rewards (OS-PLM-ADV-17).

We also made suggestions to ensure adherence to coding best practices (OS-PLM-SUG-02) and advised implementing proper validation logic to mitigate potential security issues (OS-PLM-SUG-01).

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.

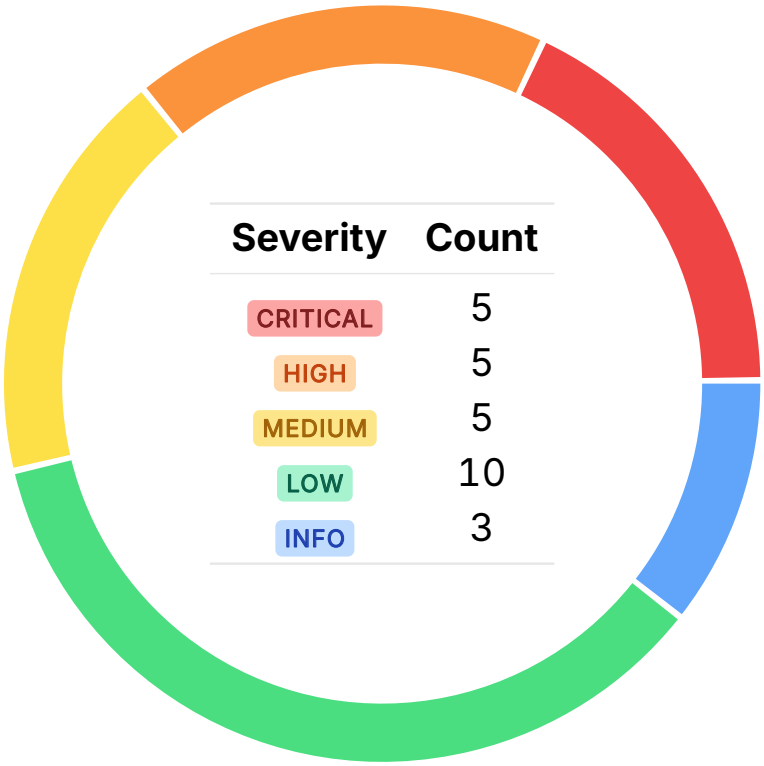
A brief description of the programs is as follows:

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 28 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	5
HIGH	5
MEDIUM	5
LOW	10
INFO	3

04 — Vulnerabilities

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

Rating criteria can be found in chapter 06.

ID	Severity	Status	Description
OS-PLM-ADV-00	CRITICAL	RESOLVED	New stakers may immediately `claim` unearned rewards because the contract does not update reward tracking variables during staking.
OS-PLM-ADV-01	CRITICAL	RESOLVED	`claim` fails to update the global reward state before processing claims, allowing users to repeatedly claim more rewards than they should by exploiting stale cumulative values.
OS-PLM-ADV-02	CRITICAL	RESOLVED	The `restakeRewards` function only restakes reward deltas while nullifying existing unclaimed rewards.
OS-PLM-ADV-03	CRITICAL	RESOLVED	The `stake` function nullifies existing unclaimed rewards on new stake positions, even though users may have pending rewards from previous unstaking.
OS-PLM-ADV-04	CRITICAL	RESOLVED	`restakeRewards` fails to initialize the user's reward state when creating a new staking position.
OS-PLM-ADV-05	HIGH	RESOLVED	`slashValidator` function has not updated the `delegatedAmount` and `stakeInfo` states.

OS-PLM-ADV-06	HIGH	RESOLVED	`addRewards` incorrectly checks the treasury's balance without accounting for prior commitments, allowing multiple reward additions that exceed actual funds.
OS-PLM-ADV-07	HIGH	RESOLVED	The new commission collection scheme may result in precision loss that prevents validators from claiming their full commission.
OS-PLM-ADV-08	HIGH	RESOLVED	`removeRewardToken` does not properly update checkpoints and user states, allowing rewards to continue accruing after the `rewardToken` is removed.
OS-PLM-ADV-09	HIGH	RESOLVED	`forceSettleValidatorCommission` does not check the validator status, leading to possible commission accrual during an inactive state.
OS-PLM-ADV-10	MEDIUM	RESOLVED	`setValidatorStatus` fails to handle multiple edge cases that lead to invalid behavior.
OS-PLM-ADV-11	MEDIUM	RESOLVED	User-Validator Relationship might be prematurely removed due to `hasPendingReward` being cleared despite users having unclaimed rewards from removed `rewardToken`s.
OS-PLM-ADV-12	MEDIUM	RESOLVED	`_validateTokenForClaim` does not check whether the user has pending rewards, preventing users from claiming rewards for a removed `rewardToken`.
OS-PLM-ADV-13	MEDIUM	RESOLVED	If a user stakes before a reward rate is set, the system mistakenly applies the later rate retroactively, resulting in over-rewarding.

OS-PLM-ADV-14	MEDIUM	RESOLVED	The `restake` function double counts matured cooldown amounts, allowing users to restake more than their actual available balance.
OS-PLM-ADV-15	LOW	RESOLVED	`calculateRewardsWithCheckpoints` fallback logic has not considered a case where the `rewardToken` added timestamp is later than the start of the staking time.
OS-PLM-ADV-16	LOW	RESOLVED	`claim` maynot be used to claim rewards from slashed validators and does not clear the user-validator relationship state.
OS-PLM-ADV-17	LOW	RESOLVED	`restakeRewards` does not clear pending rewards flags and user-validator relationships after claiming rewards before restaking.
OS-PLM-ADV-18	LOW	RESOLVED	Setting `maxValidatorPercentage` below 100% when `totalStaked` is zero will block all future staking, and applying it after validators are already over the new cap may lead to inconsistent or unenforced limits.
OS-PLM-ADV-19	LOW	RESOLVED	In `updateRewardPerTokenForValidator`, the `validator.slashed` check is never reached because slashed validators are also inactive, and the function failts to cap rewards at the token's removal time, causing over-accrual of commissions.
OS-PLM-ADV-20	LOW	RESOLVED	Re-adding a previously removed token does not clear its removal timestamp in `addRewardToken`.

OS-PLM-ADV-21	LOW	RESOLVED	Users lose accrued rewards when reward tokens are removed and re-added due to incorrect timestamp handling.
OS-PLM-ADV-22	LOW	RESOLVED	`addValidator` does not initialize reward checkpoints, potentially resulting in inconsistent reward tracking.
OS-PLM-ADV-23	LOW	RESOLVED	`claim(address token)` and `claimAll()` incorrectly exclude rewards from inactive validators.
OS-PLM-ADV-24	LOW	RESOLVED	`_calculateActivelyCoolingAmount` and `_countActiveCooldowns` do not properly handle cooldowns from slashed validators.

Invalid New Staking State CRITICAL OS-PLM-ADV-00

Description

The current implementation logic contains state inconsistencies where the contract fails to reflect the latest state. Specifically, when a user stakes tokens, it does not update `userValidatorRewardPerTokenPaid` and `lastPaidTimestamp`, which are essential for accurate reward tracking. As a result, new stakers are treated as if they have been staking from the start, allowing them to claim unearned rewards and potentially drain validator emissions.

Remediation

Ensure to update `userValidatorRewardPerTokenPaid` and `lastPaidTimestamp` on staking to properly reflect the current reward state.

Patch

Resolved in dd5df2e.

Failure to Update Reward State CRITICAL OS-PLM-ADV-01

Description

The current implementation of `claim` in `RewardsFacet` fails to update the global reward state (`validatorRewardPerTokenCumulative`, `validatorLastUpdateTimes`, and `totalClaimableByToken`) before computing user rewards. As a result, users may perform multiple sequential claims for the same elapsed time window, resulting in reward over-distribution. The function relies on up-to-date cumulative values, but without updating them, each call incorrectly assumes no prior claims were made. This allows users to claim more than they are entitled to by exploiting stale global state.

Remediation

Update the validator's reward state before processing user claims.

Patch

Resolved in dd5df2e.

Restake Rewards Loses Existing Rewards CRITICAL OS-PLM-ADV-02

Description

`restakeRewards` utilizes `getPendingRewardForValidator` to calculate rewards to restake, which only returns the reward delta since the last update. However, the function also nullifies `userRewards` which stores existing unclaimed rewards. This means that when a user calls `restakeRewards`, only the new reward delta will be restaked while any previously accumulated unclaimed rewards in `userRewards` will be lost since they are nullified but not included in the restake amount.

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

```
function restakeRewards(
    uint16 validatorId
) external nonReentrant returns (uint256 amountRestaked) {
[...]
    for (uint256 i = 0; i < currentUserValidators.length; i++) {
        uint16 userValidatorIdLoop = currentUserValidators[i];

        uint256 validatorReward = IRewardsGetter(address(this)).getPendingRewardForValidator(
            msg.sender, userValidatorIdLoop, tokenToRestake
        );

        if (validatorReward > 0) {
            amountRestaked += validatorReward;
            PlumeRewardLogic.updateRewardsForValidator($, msg.sender, userValidatorIdLoop);
            $.userRewards[msg.sender][userValidatorIdLoop][tokenToRestake] = 0;
[...]
}
```

Remediation

Modify `restakeRewards` to include both the existing unclaimed rewards from `userRewards` and the new reward delta when calculating the total amount to restake.

Patch

Resolved in e0e8134

Stake Function Incorrectly Nullifies User Rewards CRITICAL OS-PLM-ADV-03

Description

In the current implementation, when creating a new stake position with a validator, `stake` nullifies `userRewards` for that validator and token combination. However, this is incorrect because users may have unclaimed rewards from previous unstaking operations with that validator. By nullifying these rewards, users lose access to their legitimately earned but unclaimed rewards.

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

```
function stake(
    uint16 validatorId
) external payable returns (uint256) {
[...]
    if (isNewStakeForValidator) {
        address[] memory rewardTokens = $.rewardTokens; // Get all system reward tokens
        for (uint256 i = 0; i < rewardTokens.length; i++) {
            address token = rewardTokens[i];
            if ($.isRewardToken[token]) {
[...]
                // 3. Initialize any stored pending rewards for this specific validator/token to
                    ↪  zero.
                $.userRewards[msg.sender][validatorId][token] = 0;
[...]
}
```

Remediation

Remove the nullification of `userRewards` in `stake` when creating new stake positions.

Patch

Resolved in e0e8134

Missing Restake Reward State Initialization CRITICAL OS-PLM-ADV-04

Description

`restakeRewards` allows users to restake their pending rewards from all validators to a target validator.

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

```
function restakeRewards(
    uint16 validatorId
) external nonReentrant returns (uint256 amountRestaked) {
[...]
    // Calculate and claim all pending rewards
    amountRestaked = _calculateAndClaimAllRewards(user, tokenToRestake);

    // Validate restake amount
    if (amountRestaked == 0) {
        revert NoRewardsToRestake();
    }
    if (amountRestaked < $.minStakeAmount) {
        revert StakeAmountTooSmall(amountRestaked, $.minStakeAmount);
    }
    // Perform restaking workflow
    _performRestakeWorkflow(user, validatorId, amountRestaked, "rewards");
[...]
}
```

However, the current code does not handle the case where restaking accumulated rewards creates a new staking position with the target validator. In this case, the reward state for the new staking position is not initialized, which results in incorrect reward calculations for the user.

Remediation

Follow the same reward state initialization logic used in the `stake` function to maintain consistent behavior.

Patch

Resolved in a17465b.

Failure to Update Stake Info State HIGH OS-PLM-ADV-05

Description

The current implementation logic for `slashValidator`, has logic to penalize the stakers of the slashed validator by nullifying their stakes. There are some states that are required to be updated due to this, but it has not updated the `delegatedAmount` and `stakeInfo` yet.

Remediation

Ensure to update `delegatedAmount` and `stakeInfo`.

Patch

Resolved in d408f63

Incorrect Treasury Balance Accounting HIGH OS-PLM-ADV-06

Description

`RewardsFacet::addRewards` checks the treasury's token balance to ensure sufficient funds but does not transfer the tokens when rewards are added. This creates a flaw, if the treasury has a balance `X` and `addRewards(token, X)` is called multiple times, each call passes the balance check because the balance remains unchanged. As a result, `rewardsAvailable[token]` may be inflated to values higher than the actual funds in the treasury. This results in over-accounting, resulting in the system promising more rewards than it is capable of paying.

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

```
function addRewards(
    address token,
    uint256 amount
) external payable virtual nonReentrant onlyRole(PlumeRoles.REWARD_MANAGER_ROLE) {
    [...]
    // Check if treasury has sufficient funds - direct balance check
    if (token == PLUME) {
        // For native PLUME, check the treasury's ETH balance
        if (treasury.balance < amount) {
            revert InsufficientBalance(token, treasury.balance, amount);
        }
    } else {
        // For ERC20 tokens, check the token balance
        uint256 treasuryBalance = IERC20(token).balanceOf(treasury);
        if (treasuryBalance < amount) {
            revert InsufficientBalance(token, treasuryBalance, amount);
        }
    }
    [...]
}
```

Remediation

Implement proper checks to ensure the treasury has enough funds for rewards.

Patch

Resolved in dd5df2e.

Commission Precision Loss HIGH OS-PLM-ADV-07

Description

In the current implementation, validator commission is calculated and accumulated through `commissionDelta` taken from each user's reward calculation. Due to precision loss from rounding down in these calculations, there may be a discrepancy between the total commission amount tracked in `validatorAccruedCommission` and the actual sum of all `commissionDelta` amounts taken from user rewards.

This discrepancy means that the treasury may not have sufficient funds to pay out the full commission amount tracked in `validatorAccruedCommission`, since the actual commission amounts deducted from user rewards will be lower than what is tracked. As a result, validator commission claims may fail due to insufficient funds in the treasury when validators attempt to claim their tracked commission amounts.

Remediation

Ensure that the commission rounding direction is implemented properly.

Patch

Resolved in f9a500b.

Improper Token Removal Process HIGH OS-PLM-ADV-08

Description

When a `rewardToken` is removed, the current code fails to clear validator and user states related to that token. Due to the checkpoint-based reward system, users continue to accrue rewards after `removeRewardToken` is called because the reward rate is still stored in the most recent checkpoint. Additionally, if a removed rewardToken is re-added, validators will retroactively accrue commission for the period when the token was removed.

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

```
function removeRewardToken(
    address token
) external onlyRole(PlumeRoles.REWARD_MANAGER_ROLE) {
[...]
    // Update rewards using the library before removing
    for (uint256 i = 0; i < $.validatorIds.length; i++) {
        // Needs to update the cumulative index, not user rewards
        PlumeRewardLogic.updateRewardPerTokenForValidator($, token, $.validatorIds[i]);
    }
    $.rewardRates[token] = 0;

    // Update the array
    $.rewardTokens[tokenIndex] = $.rewardTokens[$.rewardTokens.length - 1];
    $.rewardTokens.pop();

    // Update the mapping
    $.isRewardToken[token] = false;

    delete $.maxRewardRates[token];
    emit RewardTokenRemoved(token);
}
```

Remediation

Ensure that checkpoints, user reward states, and validator reward states are properly updated when removing a `rewardToken` to prevent unintended reward accrual.

Patch

Resolved in a17465b.

Missing Settle Commission Validation HIGH OS-PLM-ADV-09

Description

In the current code, a validator may manually settle their commission by calling a function named `forceSettleValidatorCommission`.

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

```
function forceSettleValidatorCommission(
    uint16 validatorId
) external {
    PlumeStakingStorage.Layout storage $s = PlumeStakingStorage.layout();

    // Perform validator existence check directly
    if (!$s.validatorExists[validatorId]) {
        revert ValidatorDoesNotExist(validatorId);
    }

    PlumeRewardLogic._settleCommissionForValidatorUpToNow($s, validatorId);
}
```

However, the function does not check whether the validator is currently in an inactive state. Due to this, a validator may still accrue commission during their inactive period.

Remediation

Ensure that validators maynot accrue commission during an inactive state.

Patch

Resolved in a17465b.

Improper Set Validator Status Implementation MEDIUM OS-PLM-ADV-10

Description

In the current code, a validator's status may be changed through the function `setValidatorStatus`.

>_ plume/src/facets/ValidatorFacet.sol SOLIDITY

```
function setValidatorStatus(
    uint16 validatorId,
    bool newActiveStatus
) external onlyRole(PlumeRoles.ADMIN_ROLE) _validateValidatorExists(validatorId) {
    PlumeStakingStorage.Layout storage $ = PlumeStakingStorage.layout();
    PlumeStakingStorage.ValidatorInfo storage validator = $.validators[validatorId];

    // Prevent activating an already slashed validator through this function
    if (newActiveStatus && validator.slashed) {
        revert ValidatorAlreadySlashed(validatorId);
    }

    validator.active = newActiveStatus;
    // $.validators[validatorId].slashed should remain false unless explicitly slashed

    emit ValidatorStatusUpdated(validatorId, newActiveStatus, validator.slashed);
}
```

However, there are several edge cases that have not been properly handled during status changes:

1. When transitioning from inactive to active state, the function does not reinitialize `validatorLastUpdateTimes` to prevent reward accrual during the inactive period. Additionally, the user's reward state needs to be updated to prevent accrual during this inactive period.
2. When transitioning from active to inactive state, the function does not preserve users' accrued rewards up to the point of deactivation.

Remediation

Ensure that `setValidatorStatus` properly handles all state transitions by:

1. Reinitializing timestamps and reward states when activating validators
2. Preserving accrued rewards when deactivating validators

Patch

Resolved in a17465b.

Premature User-Validator Relationship Removal MEDIUM OS-PLM-ADV-11

Description

In the existing implementation, a removed `rewardToken` may still have pending rewards that have not been claimed by users. However, currently, `clearPendingRewardsFlagIfEmpty` only iterates through active `rewardTokens` and may clear the `hasPendingReward` flag prematurely.

>_ *plume/src/lib/PlumeRewardLogic.sol* SOLIDITY

```
function clearPendingRewardsFlagIfEmpty(
    PlumeStakingStorage.Layout storage $,
    address user,
    uint16 validatorId
) internal {
[...]
    // Check if user still has any pending rewards
    address[] memory currentRewardTokens = $.rewardTokens;
    for (uint256 i = 0; i < currentRewardTokens.length; i++) {
        if ($.userRewards[user][validatorId][currentRewardTokens[i]] > 0) {
            return; // Still has pending rewards, don't clear flag
        }
    }

    // No pending rewards found - clear the flag
    $.userHasPendingRewards[user][validatorId] = false;
}
```

Due to this, it is possible for the user-validator relationship to be cleared prematurely, leading to users being unable to `claim` their unclaimed rewards from a removed `rewardToken`. This occurs because `_validateTokenForClaim` only iterates through the latest list of `userValidators` when checking for existing rewards if the target token is in a removed state.

Remediation

Ensure that the `hasPendingReward` flag is cleared only when both removed and active `rewardToken` rewards for a user are zero.

Patch

Resolved in a17465b.

Invalid Claim Validation MEDIUM OS-PLM-ADV-12

Description

In the current validation of `claim`, the `_validateTokenForClaim` function implements a check to ensure that users may still claim rewards from a removed `rewardToken`.

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

```
function _validateTokenForClaim(address token, address user) internal view returns (bool
    ↪  isActive) {
[...]
    if (!isActive) {
        // If token is not active, check if there are previously earned/stored rewards
        uint16[] memory validatorIds = $.userValidators[user];
        bool hasExistingRewards = false;

        for (uint256 i = 0; i < validatorIds.length; i++) {
            if ($.userRewards[user][validatorIds[i]][token] > 0) {
                hasExistingRewards = true;
                break;
            }
        }

        if (!hasExistingRewards) {
            revert TokenDoesNotExist(token);
        }
    }
}
```

However, the current check only examines rewards that have been stored in the `userRewards` mapping. There is an edge case where a user's rewards might not be in storage yet and are still in a pending state.

Remediation

Ensure that the validation also checks whether the user has pending rewards.

Patch

Resolved in a17465b.

Miscalculation of Retroactive Reward Rate MEDIUM OS-PLM-ADV-13

Description

The contract does not properly handle the case when users stake before any `rewardRate` has been set. If a user stakes at time T1 and the `rewardRate` rate is only set later at T2, the system incorrectly utilizes the default `rewardRate` (which is set to the most recent `rewardRate`), assuming this new rate applies retroactively to the entire period from T1 to T2. This implies rewards are accrued for past time intervals where the intended rate was actually zero. As a result, users may unfairly collect rewards for periods when no incentives should have existed.

Remediation

Set an explicit zero reward rate checkpoint at the time of token addition (T1). When the admin sets a rate (T2), and the rewards are calculated, any time delta before T2 should be multiplied by zero reward rate.

Patch

Resolved in a17465b.

Double Counting of Matured Cooldowns MEDIUM OS-PLM-ADV-14

Description

The current implementation of `restake` double counts matured cooldown amounts. When a cooldown has matured, the amount is counted both in `coolingFromTarget` and `withdrawableAmount`, allowing users to restake more tokens than they actually have available.

Remediation

Modify the restake function to ensure matured cooldown amounts are only counted once, either in `coolingFromTarget` or `withdrawableAmount`, but not both.

Patch

Resolved in a17465b.

Improper Effective Time Delta Calculation LOW OS-PLM-ADV-15

Description

In the current fallback logic of `calculateRewardsWithCheckpoints`, when the `userValidatorRewardPerTokenPaidTimestamp` is zero, it will initialize the fallback value with the initial stake time.

>_ *plume/src/lib/PlumeRewardLogic.sol* SOLIDITY

```
function calculateRewardsWithCheckpoints(
    PlumeStakingStorage.Layout storage $,
    address user,
    uint16 validatorId,
    address token,
    uint256 userStakedAmount
) internal returns (uint256 totalUserRewardDelta, uint256 totalCommissionAmountDelta, uint256
    ↪  effectiveTimeDelta) {
[...]
    uint256 lastUserRewardUpdateTime =
        ↪  $.userValidatorRewardPerTokenPaidTimestamp[user][validatorId][token];

    if (lastUserRewardUpdateTime == 0) {
        lastUserRewardUpdateTime = $.userValidatorStakeStartTime[user][validatorId];
[...]
}
```

However, there is a possible case where the token was added after the initial stake time, which implies the `lastUserRewardUpdateTime` should be the time when the `rewardToken` was added instead of the initial stake time. Note that the calculated reward is not impacted since there are no checkpoints between the initial stake and when the token was added, but the returned `effectiveTimeDelta` in the function is incorrect.

Remediation

Ensure the fallback logic uses the correct timestamp.

Patch

Resolved in a17465b.

Inconsistent Claim Behavior LOW OS-PLM-ADV-16

Description

In the current implementation of `claim(address token, uint16 validatorId)`, when a user wants to claim rewards from a slashed validator, it fails due to the check of `_validateValidatorForClaim`.

Furthermore, at the end of the function, it is possible that the `claim` leaves the user with no remaining involvement with the validator, in which case the relationship state needs to be cleared.

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

```
function claim(address token, uint16 validatorId) external nonReentrant returns (uint256) {
[...]
    _validateValidatorForClaim(validatorId);

    // Process rewards for this specific validator
    uint256 reward = _processValidatorRewards(msg.sender, validatorId, token);

    // Finalize claim if there are rewards
    if (reward > 0) {
        _finalizeRewardClaim(token, reward, msg.sender);
    }

    // Clear pending flags for this validator
    PlumeStakingStorage.Layout storage $ = PlumeStakingStorage.layout();
    PlumeRewardLogic.clearPendingRewardsFlagIfEmpty($, msg.sender, validatorId);

    return reward;
}
```

Remediation

Ensure that `claim` may be utilized to claim rewards from slashed validators and that it clears the relationship state at the end of the function.

Patch

Resolved in a17465b.

Missing Pending Reward Cleanup LOW OS-PLM-ADV-17

Description

In the current implementation of `restakeRewards`, the function claims all available rewards via `_calculateAndClaimAllRewards` and uses the total claimed amount as the stake amount for the target validator.

However, the code does not clear pending rewards flags or user-validator relationships in cases where the claim process results in zero rewards or removes all user involvement with the claimed validator.

Remediation

Add logic to clear pending rewards flags and user-validator relationships when appropriate after claiming rewards.

Patch

Resolved in a17465b.

Improper Validator Cap Enforcement LOW OS-PLM-ADV-18

Description

The current implementation of `setMaxValidatorPercentage` may break the app if set below 100% while `totalStaked` is zero, as any staking attempt would violate the limit, effectively halting all staking. Additionally, if invoked after validators have already received delegations, existing validators may have `delegatedAmountwithout` that exceed the new max percentage limit without any enforcement, resulting in an inconsistent state.

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

```
function setMaxValidatorPercentage(
    uint256 newPercentage
) external onlyRole(PlumeRoles.ADMIN_ROLE) {
    PlumeStakingStorage.Layout storage $ = PlumeStakingStorage.layout();
    // A percentage must not exceed 100% (10,000 basis points).
    if (newPercentage > 10_000) {
        revert InvalidPercentage(newPercentage);
    }
    uint256 oldPercentage = $.maxValidatorPercentage;
    $.maxValidatorPercentage = newPercentage;
    emit MaxValidatorPercentageUpdated(oldPercentage, newPercentage);
}
```

Remediation

Re-evaluate the logic to determine whether this is intended or not.

Patch

Resolved in a17465b.

Inconsistencies in Reward Update Logic LOW OS-PLM-ADV-19

Description

In `PlumeRewardLogic::updateRewardPerTokenForValidator`, the check for `validator.slashed` is placed after the `!validator.active` check, and as a result, it will never be executed since slashed validators are always marked inactive. This skips important logic, such as capping the last update timestamp at the slashing time and ensuring no rewards accrue post-slash. Also, the function does not check whether the given reward token has already been removed. This allows rewards and validator commissions to continue accruing beyond the token's intended removal time, resulting in inflated commission values.

Remediation

Move the `validator.slashed` check before the `!validator.active` check to ensure correct handling. Additionally, creating a zero-rate checkpoint at both token addition and removal would provide clearer boundaries

Patch

Resolved in a17465b.

Token Re-Addition Handling Flaw LOW OS-PLM-ADV-20

Description

`addRewardToken` fails to clear a token's previous removal timestamp if it is re-added after it has been removed. This results in the system treating the token as removed despite the fact that it is active, resulting in inconsistencies in reward accrual and claiming logic.

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

```
function addRewardToken(
    address token
) external onlyRole(PlumeRoles.REWARD_MANAGER_ROLE) {
    PlumeStakingStorage.Layout storage $ = PlumeStakingStorage.layout();
    if (token == address(0)) {
        revert ZeroAddress("token");
    }
    if ($.isRewardToken[token]) {
        revert TokenAlreadyExists();
    }
    uint256 additionTimestamp = block.timestamp;
    $.rewardTokens.push(token);
    $.isRewardToken[token] = true;
    [...]
}
```

Remediation

Add logic to clear the stale removal timestamp to delete the removal timestamp during re-addition.

Patch

Resolved in a17465b.

Lost Rewards Due to Token Re-addition LOW OS-PLM-ADV-21

Description

The new token addition/removal timestamp tracking may result in users losing previously accrued rewards. Consider a scenario where:

1. User stakes at T0
2. Reward token added at T1
3. Token removed at T2
4. Token re-added at T3

Since `tokenAdditionTimestamps` is set to T3 which is after the user's stake start time (T0), the user loses rewards accrued during T1→T2.

Remediation

Consider relying solely on reward rate checkpoints rather than addition/removal timestamps.

Patch

Resolved in a17465b.

Missing Reward Checkpoint Initialization LOW OS-PLM-ADV-22

Description

`addValidator` currently does not initialize reward checkpoints when creating new validators. This omission may result in inconsistent reward tracking since the checkpoint history will be incomplete from the validator's creation.

Remediation

Initialize reward checkpoints during validator creation in `addValidator` to ensure complete and accurate reward tracking from the start.

Patch

Resolved in a17465b.

Inconsistent Reward Claims for Inactive Validators LOW OS-PLM-ADV-23

Description

The current implementation inconsistently handles reward claims from inactive validators. While `claim(address token, uint16 validatorId)` allows users to claim rewards from inactive validators, the batch functions, `claim(address token)` and `claimAll()` exclude them, potentially preventing users from accessing legitimately earned rewards through batch claims.

Remediation

Update the batch claim functions to include rewards from inactive validators.

Patch

Resolved in 56b9c93.

Incorrect Handling of Slashed Validator Cooldowns LOW OS-PLM-ADV-24

Description

The current implementation logic of `_calculateActivelyCoolingAmount` and `_countActiveCooldowns` does not account for cooldowns from slashed validators that should still be considered active. This inconsistency may result in incorrect calculations of cooling amounts and active cooldown counts when validators are slashed.

Remediation

Update the cooldown calculation functions to properly handle cooldowns from slashed validators according to the new logic.

Patch

Resolved in cee50e9.

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-PLM-SUG-00	`claim` does not properly increase `totalClaimable` when calculating pending rewards.
OS-PLM-SUG-01	There are several instances where proper validation is not performed, resulting in potential security issues.
OS-PLM-SUG-02	Suggestions regarding inconsistencies in the codebase and ensuring adherence to coding best practices.

Missing Total Claimable State Update OS-PLM-SUG-00

Description

In the current implementation of `claim`, the function calculates the sum of existing and pending rewards through the helper function `_processValidatorRewards`. This helper function returns the accumulated rewards without updating the `totalClaimable` state.

However, the function `_finalizeRewardClaim` incorrectly assumes that `totalClaimable` has been updated. This causes the code to attempt decreasing `totalClaimable` by more than it should. Note that this currently does not cause any degradation in functionality due to the safety check performed before subtracting the accumulated reward.

Remediation

Ensure `totalClaimable` is increased and decreased properly.

Patch

Resolved in 3a745d5.

Missing Validation Logic OS-PLM-SUG-01

Description

1. `RewardsFacet::_transferRewardFromTreasury` violates the Checks-Effects-Interactions (CEI) pattern by executing the external reward transfer before updating internal state. This ordering may expose the contract to reentrancy risks. To mitigate this, update the internal state before initiating the transfer.

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

```
function _transferRewardFromTreasury(address token, uint256 amount, address recipient)
    ↪  internal {
    [...]
    // Make the treasury send the rewards directly to the user
    IPlumeStakingRewardTreasury(treasury).distributeReward(token, amount, recipient);
    // Update accounting
    PlumeStakingStorage.Layout storage $ = plumeStorage();
    $.rewardsAvailable[token] = ($.rewardsAvailable[token] > amount) ?
        ↪  $.rewardsAvailable[token] - amount : 0;
}
```

Remediation

Incorporate the above validations.

Patch

Resolved in 8801360

Code Maturity OS-PLM-SUG-02

Description

1. `claim` and `claimAll` in `RewardsFacet` duplicate nearly identical logic for reward distribution, rendering the code redundant. It will be appropriate to refactor the shared logic into a re-utilizable internal function to improve clarity and reduce redundancy.
2. `RewardsFacet::setRewardRates` redundantly calls `updateRewardPerTokenForValidator` for the same validator-token pairs, resulting in unnecessary gas costs. Remove the duplicate function call.
3. The `previousAccrued` variable in `updateRewardPerTokenForValidator` is not used anywhere and can be safely removed.
4. `getMaxRewardRate` should check whether the passed token is an active `rewardToken`.
5. `tokenRewardInfo` returns `$.lastUpdateTimes` which is never updated or used in the codebase.
6. The `_validateValidatorForStaking` check inside `restake` is redundant since `_performStakeSetup` already performs this validation.
7. The `requestCommissionClaim` function should settle any pending commission first before executing the claim request.
8. The `setMaxRewardRate` function should create a new reward checkpoint for existing validators whose rewards exceed the new max reward.

Remediation

Implement the above-mentioned suggestions.

Patch

Resolved in 91e0c6d.

06 — 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.

07 — 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.