IMMUNEFI AUDIT

Immunefi / plume

DATE	July 14, 2025
AUDITOR	Blockian, Security Researchers
REPORT BY	Immunefi

ABOUT IMMUNEFI

Immunefi is the leading onchain security platform, having directly prevented hacks worth more than $25 billion USD. Immunefi security researchers have earned over $120M USD for responsibly disclosing over 4,000 web2 and web3 vulnerabilities, more than the rest of the industry combined.

Through Magnus, Immunefi delivers a comprehensive suite of best-in-class security services through a single command center to more than 300 projects — including Sky (formerly MakerDAO), Optimism, Polygon, GMX, Reserve, Chainlink, TheGraph, Gnosis Chain, Lido, LayerZero, Arbitrum, StarkNet, EigenLayer, AAVE, ZKsync, Morpho, Ethena, USDT0, Stacks, Babylon, Fuel, Sei, Scroll, XION, Wormhole, Firedancer, Jito, Pyth, Eclipse, PancakeSwap and many more.

Magnus unifies SecOps across the entire onchain lifecycle, combining Immunefi's market leading products and community of elite security researchers with a curated set of the very best security products and technologies provided by top security firms — including Runtime Verification, Dedaub, Fuzzland, Nexus Mutual, Failsafe, OtterSec and others.

Magnus is powered by Immunefi's proprietary vulnerabilities dataset — the largest and most comprehensive in web3, ensuring that security leaders and teams have the best possible tools for identifying and mitigating life threats before they cause catastrophic harm, all while reducing operational overhead and complexity.

Learn how you can benefit too at immunefi.com.

TERMINOLOGY

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- **Likelihood** represents the likelihood of a finding to be triggered or exploited in practice
- **Impact** specifies the technical and business-related consequences of a finding
- **Severity** is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

LIKELIHOOD	IMPACT		
	HIGH	MEDIUM	LOW
CRITICAL	Critical	Critical	High
HIGH	High	High	Medium
MEDIUM	Medium	Medium	Low
LOW	Low		
NONE	None		

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

EXECUTIVE SUMMARY

Immunefi conducted two comprehensive audits of Plume Network's contracts repository between April 24th and June 3rd. Across both audits, 76 issues were identified.

SUMMARY

Name	Plume Network
Repository	https://github.com/plumenetwork/contracts
Audit Commit	f5332b2bbcc9f4a58ac818785ed11762968d5610
Type of Project	Stablecoin, RWA, Blockchain
Audit Timeline	● Audit 1: April 24th - May 8th ● Audit 2: June 1st - June 3rd
Fix Period	June 7th - July 9th

AUDIT 1

ISSUES FOUND

Severity	Count	Fixed	Acknowledged
Critical	14	13	1
High	6	6	0
Medium	9	8	1
Low	3	1	2
Insights	12	9	3

CATEGORY BREAKDOWN

Bug	32
Gas Optimization	0
Informational	12

AUDIT 2

ISSUES FOUND

Severity	Count	Fixed	Acknowledged
Critical	6	6	0
High	2	2	0
Medium	3	3	0
Low	9	6	0
Insights	12	8	4

CATEGORY BREAKDOWN

Bug	23
Gas Optimization	3
Informational	6

AUDIT 1 - FINDINGS

IMM-CRIT-01

Token Creator Can Upgrade ArcToken Implementation #33

Id	IMM-CRIT-01
Severity	Critical
Category	Bug
Status	Fixed in 20814d1b32321e2d1d26ff2e20239a79712714a5

Description

Code Affected: `ArcTokenFactory.sol::createToken`

When a new ArcToken is created using the createToken function, the UPGRADER_ROLE is granted to the msg.sender (the token creator):

```
None
token.grantRole(token.UPGRADER_ROLE(), msg.sender);
```

This allows the token creator to call `upgradeToAndCall` on the `ArcToken` contract, giving them full control over the token's implementation. As a result, the creator can upgrade the ArcToken contract at any time, bypassing the intended control of the ArcTokenFactory and potentially introducing malicious logic or vulnerabilities.

This undermines the security and trust assumptions of the ArcTokenFactory, as upgrades can occur without factory or governance oversight.

Recommendation

Grant the `UPGRADER_ROLE` to the ArcTokenFactory contract instead of the token creator.

IMM-CRIT-02

Unbounded Array Iteration in Yield Distribution Can Lead to No Yield Distribution #32

Id	IMM-CRIT-02
Severity	Critical
Category	Bug
Status	Acknowledged

Description

Code Affected: `ArcToken.sol::distributeYield`, `distributeYieldWithLimit`, preview functions

Both `distributeYield` functions (`distributeYield` and `distributeYieldWithLimit`), as well as the preview functions in `ArcToken.sol`, iterate over the entire `$.holders` array to determine `effectiveTotalSupply`.

Since the `$.holders` array is not bounded in size, if there are a large number of holders, these functions may revert due to exceeding the block gas limit.

Recommendation

- Avoid iterating over unbounded arrays in public or external functions.
- Consider using a mapping or checkpointing mechanism to track supply and yield distribution efficiently.

IMM-CRIT-03

No minOutAmount Protection in Token Purchase Can Result in Theft of Funds from Buyers #31

Id	IMM-CRIT-03
Severity	Critical
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `ArcTokenPurchase.sol::buy`

The buy function in `ArcTokenPurchase` does not include a `minOutAmount` parameter. This omission allows the seller to frontrun a buy transaction and manipulate the price, resulting in the buyer receiving fewer tokens than expected.

There are two main ways the seller can manipulate the outcome:

1. The seller can frontrun a `buy` transaction with an `enableToken` transaction and set a new price, causing the buyer to receive less than the expected amount.
2. The seller can frontrun a `buy` transaction with an `upgradeToAndCall` transaction on the `ArcToken` itself, changing the decimals and thus altering the calculation of `arcTokensBaseUnitsToBuy`.

For example:

```
TypeScript
uint8 tokenDecimals = token.decimals(); // Get decimals dynamically
uint256 scalingFactor = 10 ** tokenDecimals;

// Calculate ArcToken base units to buy, assuming tokenPrice is for 1 full ArcToken (scaled
by its decimals)
uint256 arcTokensBaseUnitsToBuy = (_purchaseAmount * scalingFactor) / info.tokenPrice;
```

Both `tokenDecimals` and `tokenPrice` can be manipulated by the seller, impacting the buyer's received amount.

Recommendation

Add a `minOutAmount` parameter to the `buy` function to protect buyers from receiving fewer tokens than expected due to price or decimal manipulation.

IMM-CRIT-04

Users Can Spin Multiple Times Before Randomness Callback #23

Id	IMM-CRIT-04
Severity	Critical
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `Spin.sol::startSpin`

TypeScript

```
    modifier canSpin() {

        uint256 _lastSpinTimestamp = userDataStorage.lastSpinTimestamp;

        // Retrieve last spin date components
        (uint16 lastSpinYear, uint8 lastSpinMonth, uint8 lastSpinDay) = (
            dateTime.getYear(_lastSpinTimestamp),
            dateTime.getMonth(_lastSpinTimestamp),
            dateTime.getDay(_lastSpinTimestamp)
        );

        // Retrieve current date components
        (uint16 currentYear, uint8 currentMonth, uint8 currentDay) =
            (dateTime.getYear(block.timestamp), dateTime.getMonth(block.timestamp),
dateTime.getDay(block.timestamp));

        // Ensure the user hasn't already spun today
        if (isSameDay(lastSpinYear, lastSpinMonth, lastSpinDay, currentYear, currentMonth,
currentDay)) {
            revert AlreadySpunToday();
        }

        _;
    }
```

The `lastSpinTimestamp` is validated against the current day, but, `lastSpinTimestamp` is only updated after the randomness callback is received from Supra.

This allows users to call `startSpin` multiple times before the callback is processed, resulting in multiple concurrent spins on the same day for the same user.

Recommendation

Update `lastSpinTimestamp` as soon as `startSpin` is called, not after the callback.

IMM-CRIT-05

Looping over unbounded arrays - Multiple issues including slashing evasion #11

Id	IMM-CRIT-05
Severity	Critical
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: Multiple locations

Several functions across the staking system iterate over arrays whose lengths are not explicitly bounded. This practice can lead to excessive gas consumption and, in some cases, cause the function to revert due to block gas limits. In some scenarios, malicious actors or even regular users can exploit these unbounded loops to prevent critical protocol actions, such as slashing, unstaking, or claiming rewards.

Below is a summary of the most significant unbounded loops, ranked by severity:

1. **ValidatorFacet.sol::slashValidator**
   CRITICAL — Iterates over `$.validatorIds`, `$.rewardTokens`, and `$.validatorStakers`. Since $.validatorStakers can be manipulated by anyone (anyone can stake), a malicious validator could inflate this array to prevent themselves from being slashed.

2. **PlumeValidatorLogic.sol::removeStakerFromValidator**
   MEDIUM — Iterates over all `validatorStakers` to pop a staker. If the array is too large, honest users may be unable to unstake.

3. **RewardsFacet.sol**
   a. Iterates over all `$.validatorIds` and `$.rewardTokens`.
   b. MEDIUM: May prevent adding or removing reward tokens.
   c. INSIGHT: Can cause `claimAll` or `claim(token)` to fail, though normal claims still work.

4. **StakingFacet.sol::restakeRewards**
   LOW — Iterates over all `userValidators`, which is user-controlled. Excessive entries may cause

user-initiated failures.

5. **PlumeRewardLogic.sol::updateRewardsForValidator**
   INSIGHT — Iterates over all `$.rewardTokens` (protocol-controlled), so risk is minimal.

6. **ManagementFacet.sol::adminCorrectUserStakeInfo**
   INSIGHT — Iterates over all `validatorIds` for a single user. If it reverts, it only affects syncing of stakeInfo and `userValidatorStakes`, which is not a real problem since `stakeInfo` and `userValidatorStakes` are synced anyways.

7. **ValidatorFacet.sol::_updateRewardsForAllValidatorStakers**
   INSIGHT — Iterates over all `$.validatorStakers`, but this function already reverts if the array is bigger than 100 (see IMM-CRIT-09).

Recommendation

- Avoid iterating over unbounded arrays wherever possible. Use mappings and checkpointing mechanisms to track state efficiently.
- If iteration over an array is unavoidable, enforce a strict upper bound (e.g., 100 elements) on the array's length. Always check the array length before allowing new entries.

IMM-CRIT-06

Reward Distribution Is Unrelated to Amount Added With `addRewards` #9

Id	IMM-CRIT-06
Severity	Critical
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `RewardsFacet.sol::addRewards`, `_transferRewardFromTreasury`

The amount of rewards distributed to users is not actually limited or affected by the amount added via `addRewards`. When `addRewards` is called, the amount is used to increment the `rewardsAvailable` variable:

TypeScript

```
function addRewards(
        address token,
        uint256 amount
    ) external payable virtual nonReentrant onlyRole(PlumeRoles.REWARD_MANAGER_ROLE) {
        PlumeStakingStorage.Layout storage $ = plumeStorage();
        if (!$.isRewardToken[token]) {
            revert TokenDoesNotExist(token);
        }

        address treasury = getTreasuryAddress();
        if (treasury == address(0)) {
            revert TreasuryNotSet();
        }

        // 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);
            }
        }

        uint16[] memory validatorIds = $.validatorIds;
        for (uint256 i = 0; i < validatorIds.length; i++) {
            // Use library function to update validator cumulative index
            PlumeRewardLogic.updateRewardPerTokenForValidator($, token, validatorIds[i]);
        }

        // Only update the accounting - actual funds remain in the treasury
        $.rewardsAvailable[token] += amount;
        emit RewardsAdded(token, amount);
    }
```

However, `rewardsAvailable` is not enforced as a limit when transferring rewards to users nor does it change the actual amount of rewards. The actual transfer logic is as follows:

TypeScript

```
function _transferRewardFromTreasury(address token, uint256 amount, address recipient)
internal {
    address treasury = getTreasuryAddress();
    if (treasury == address(0)) {
        revert TreasuryNotSet();
    }

    // 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;
}
```

If the amount transferred is greater than `rewardsAvailable`, the transfer will simply continue as normal and zero out `rewardsAvailable` so it has no effect on the actual rewards calculation nor does it prevent rewards bigger than `rewardsAvailable` from being transferred.

Recommendation

- Review all reward calculation logic to use the actual amount of rewards available via `rewardsAvailable`.
- Prevent transfers of rewards that would cause `rewardsAvailable` to go below zero.

IMM-CRIT-07

No Rewards Accrued Due to Zero Validator Stake #8

Id	IMM-CRIT-07
Severity	Critical
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `PlumeRewardLogic.sol::updateRewardPerTokenForValidator`

The `validatorRewardPerTokenCumulative` value is always zero because it depends on `$.validatorTotalStaked[validatorId]`, which is never incremented and remains at zero. As a result, the reward calculation logic never updates the cumulative reward per token:

TypeScript

```
uint256 totalStaked = $.validatorTotalStaked[validatorId];
if (totalStaked > 0) {
    uint256 lastUpdate = $.validatorLastUpdateTimes[validatorId][token];
    if (block.timestamp > lastUpdate) {
        uint256 timeDelta = block.timestamp - lastUpdate;
        uint256 effectiveRate = $.rewardRates[token];
        if (effectiveRate > 0) {
            uint256 numerator = timeDelta * effectiveRate * REWARD_PRECISION;
            uint256 reward = numerator / totalStaked;
            $.validatorRewardPerTokenCumulative[validatorId][token] += reward;
        }
    }
}
$.validatorLastUpdateTimes[validatorId][token] = block.timestamp;
```

Since `totalStaked` is always zero, the update block is never executed, and no rewards are ever accrued for any validator or staker.

Recommendation

- Ensure that `$.validatorTotalStaked[validatorId]` is correctly updated whenever users stake is updated, so that rewards can be properly accrued and distributed.
- Review all staking and reward logic to maintain consistency between stake tracking and reward calculations.

IMM-CRIT-08

Slashing Reverts Due to `totalStaked` being 0 #7

Id	IMM-CRIT-08
Severity	Critical
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `ValidatorFacet.sol::slashValidator`

The global staking total, `$.totalStaked`, is never incremented and remains at zero. If a nonzero `penaltyAmount` is ever calculated during slashing, the following code will revert due to an underflow:

TypeScript
```
uint256 penaltyAmount = $.validatorTotalStaked[validatorId];
if (penaltyAmount > 0) {
    $.totalStaked -= penaltyAmount; // Underflows and reverts if $.totalStaked is 0
    $.validatorTotalStaked[validatorId] = 0;
    address[] storage stakers = $.validatorStakers[validatorId];
    for (uint256 i = 0; i < stakers.length; i++) {
        $.userValidatorStakes[stakers[i]][validatorId].staked = 0;
    }
}
```

Because `$.totalStaked` is zero, subtracting any positive `penaltyAmount` will cause an underflow and revert the transaction, preventing slashing from executing as intended.

Since currently `penaltyAmount` is always 0, this issue is not currently exploitable. But it can be exploited in the future and needs to be addressed.

Recommendation

Ensure that `$.totalStaked` is correctly incremented whenever users stake, so it accurately reflects the total staked amount across all validators.

IMM-CRIT-09

Slashing Does Not Burn Staked Funds Due to Zeroed Validator Total #6

Id	IMM-CRIT-09
Severity	Critical
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `ValidatorFacet.sol::slashValidator`

The core slashing functionality is ineffective because `$.validatorTotalStaked[validatorId]` is always zero (there is no place where it's incremented) when `slashValidator` is called. As a result, the calculated `penaltyAmount` is zero, and the code that zeroes out individual user stakes is never executed:

TypeScript

```
        uint256 penaltyAmount = $.validatorTotalStaked[validatorId];
        if (penaltyAmount > 0) {
            $.totalStaked -= penaltyAmount;
            $.validatorTotalStaked[validatorId] = 0;

            address[] storage stakers = $.validatorStakers[validatorId];
            for (uint256 i = 0; i < stakers.length; i++) {
                $.userValidatorStakes[stakers[i]][validatorId].staked = 0;
            }
        }
```

Because `penaltyAmount` is always zero, the intended penalty — burning all staked funds for the slashed validator — is never applied, and user stakes remain untouched.

Recommendation

Increment `$.validatorTotalStaked[validatorId]` every time userValidatorStakes is updated.

IMM-CRIT-10

Uninitialized Reward Timestamp Allows Excessive Reward Claims for New Validators and Draining `PlumeStakingRewardTreasury` #5

Id	IMM-CRIT-10
Severity	Critical
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `ValidatorFacet.sol::addValidator`, `PlumeRewardLogic::updateRewardPerTokenForValidator`

When a new validator is added via `addValidator`, the contract does not initialize the `validatorLastUpdateTimes` mapping for the new validator and each reward token.

As a result, when `updateRewardPerTokenForValidator` is called for the first time, the `timeDelta` is calculated from block timestamp to the default value of zero (the Unix epoch), resulting in an excessively large `timeDelta`. (`timeDelta = block.timestamp - 0`):

```
TypeScript
uint256 lastUpdate = $.validatorLastUpdateTimes[validatorId][token];
if (block.timestamp > lastUpdate) {
    uint256 timeDelta = block.timestamp - lastUpdate; // block.timestamp - 0
    uint256 effectiveRate = $.rewardRates[token];
    if (effectiveRate > 0) {
        uint256 numerator = timeDelta * effectiveRate * REWARD_PRECISION;
        uint256 reward = numerator / totalStaked; // the reward will be huge
        $.validatorRewardPerTokenCumulative[validatorId][token] += reward;
    }
```

This `timeDelta` results in a huge `rewardPerTokenDelta`, allowing users to claim rewards as if the validator had existed since timestamp zero. This can lead to users draining all available reward funds from the protocol. Draining `PlumeStakingRewardTreasury`

Recommendation

In `addValidator`, ensure that `validatorLastUpdateTimes` is set to the current block timestamp for each reward token when a new validator is added.

IMM-CRIT-11

Slashing Penalty Evasion via Commission Claim #4

Id	IMM-CRIT-11
Severity	Critical
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `ValidatorFacet.sol::claimValidatorCommission`

A malicious validator can frontrun a slashing transaction by calling `claimValidatorCommission` immediately before being slashed. This allows the validator to withdraw their commission rewards before their balance is zeroed out, effectively evading the intended penalty and reducing the effectiveness of the slashing mechanism.

Recommendation

- Implement a cooldown period for `claimValidatorCommission` requests.
- Ensure the slashing vote duration is **strictly shorter** than the cooldown period, otherwise the malicious validator can claim their commission before the slashing vote is finalized.

IMM-CRIT-12

Stakers Can Bypass Slashing #3

Id	IMM-CRIT-12
Severity	Critical
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `StakingFacet.sol::unstake`, `ValidatorFacet.sol::slashValidator`

Stakers are currently able to evade slashing penalties by frontrunning the `slashValidator` function or by noticing the slashing vote before it is finalized. Once a slashing vote is initiated or even detectable on-chain, users have a window of opportunity to call `unstake`, which moves their funds into the `cooled` state:

TypeScript

```
amountUnstaked = amount > info.staked ? info.staked : amount;

// ... unrelevant code for this issue ...

$.validators[validatorId].delegatedAmount -= amountUnstaked;

// ... unrelevant code for this issue ...

globalInfo.cooled += amountUnstaked;
```

Because slashing currently does not affect funds that are cooling down, users can effectively protect their stake from being slashed by quickly unstaking during or immediately after a slashing vote starts, as long as they `unstake` before `slashValidator` is called.

Recommendation

1. Introduce a new storage mapping similar to `$.userValidatorStakes`, such as `$.userValidatorCooling`, to track cooldown balances on a per-validator basis.

2. During `unstake`, record the unstaked amount into this new mapping instead of marking it as globally cooled.
3. Modify `slashValidator` to also reduce funds from the `userValidatorCooling` mapping associated with the slashed validator.

The last recommendation is very important:

4. Ensure the slashing vote duration is **strictly shorter** than the cooldown period, otherwise users will be able to unstake and get their funds back before the slashing vote is finalized.

IMM-CRIT-13

Incorrect Reward Accounting Allows Multiple Claims #2

Id	IMM-CRIT-13
Severity	Critical
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `RewardsFacet.sol::claim`, `claimAll`

When a user claims rewards, the contract sets `userValidatorRewardPerTokenPaid` to the current value of `validatorRewardPerTokenCumulative` to track the latest reward per token amount claimed:

```
TypeScript
$.userValidatorRewardPerTokenPaid[user][validatorId][token] =
    $.validatorRewardPerTokenCumulative[validatorId][token];
```

However, during reward calculation in `calculateRewardsWithCheckpoints`, if `validatorRewardPerTokenCumulative` is outdated, the function computes an up-to-date value locally (using `currentCumulativeIndex`) but does not update the stored `validatorRewardPerTokenCumulative`:

```
TypeScript
uint256 currentCumulativeIndex = $.validatorRewardPerTokenCumulative[validatorId][token];
uint256 lastUpdateTime = $.validatorLastUpdateTimes[validatorId][token];
if (block.timestamp > lastUpdateTime && $.validatorTotalStaked[validatorId] > 0) {
    uint256 timeDelta = block.timestamp - lastUpdateTime;
    uint256 effectiveRate = $.rewardRates[token];
    if (effectiveRate > 0) {
        uint256 numerator = timeDelta * effectiveRate * REWARD_PRECISION;
        currentCumulativeIndex += numerator / $.validatorTotalStaked[validatorId];
    }
}
```

As a result, the user's `userValidatorRewardPerTokenPaid` is set to the old, un-updated value. On subsequent claims, the `rewardPerTokenDelta` is calculated using this stale value, allowing users to repeatedly claim additional rewards they are not entitled to.

Recommendation

Ensure that `validatorRewardPerTokenCumulative` is updated before any reward claim action, either by calling `updateRewardPerTokenForValidator` directly or by using a modifier to enforce this update.

IMM-CRIT-14

Theft of Funds From `PlumeStakingRewardTreasury` #1

Id	IMM-CRIT-14
Severity	Critical
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `StakingFacet.sol::stake`, `restake`, `stakeOnBehalf`, `restakeRewards`

The staking functions do not update `userValidatorRewardPerTokenPaid` when a user stakes. As a result, this variable remains at its default value (zero), which causes the reward calculation to treat the user as if they had been staked since the beginning of the reward period.

As a result, when the user immediately calls `claim`, the reward calculation uses the default value (zero) for `userValidatorRewardPerTokenPaid`:

TypeScript

```
uint256 lastPaidCumulativeIndex =
$.userValidatorRewardPerTokenPaid[user][validatorId][token];

uint256 rewardPerTokenDelta = currentCumulativeIndex - lastPaidCumulativeIndex; //
currentCumulativeIndex - 0
```

An attacker can exploit this by staking and then immediately calling `claim`, receiving the full accumulated rewards they were never entitled to. They can then even immediately unstake — all in a single transaction, putting no attacker funds at risk — and then repeat this process draining the `PlumeStakingRewardTreasury`.

Recommendation

- Ensure `userValidatorRewardPerTokenPaid` and `userValidatorRewardPerTokenPaidTimestamp` are updated when staking.

Better yet:

- Call `updateRewardsForValidator` before any staking action to ensure rewards are properly accounted for.
- Consider abstracting this into a modifier or internal helper to enforce consistency across all staking-related functions.

IMM-HIGH-01

Changing Purchase Token Does Not Invalidate Existing Sales or Prices #35

Id	IMM-HIGH-01
Severity	High
Category	Bug
Status	Fixed in 77d09cf9c18a25b3f82c0a4a83a908c5c99d355e

Description

Code Affected: `ArcTokenPurchase.sol::setPurchaseToken`

The `setPurchaseToken` function in `ArcTokenPurchase.sol` allows the purchase token to be changed. However, changing the purchase token does not invalidate or update existing sales or prices, which may lead to inconsistencies or unexpected behavior for sellers who have posted sales.

Recommendation

Ensure that changing the purchase token properly invalidates any existing sales or prices.

IMM-HIGH-02

No Validation of Token Origin in `enableToken` #34

Id	IMM-HIGH-02
Severity	High
Category	Bug
Status	Fixed in c3acabfbc8fe626a9febda09619231302a76557c

Description

Code Affected: `ArcTokenPurchase.sol::enableToken`

When calling `enableToken` on the `ArcTokenPurchase` contract, there is no check to ensure that the token being enabled was actually deployed by the factory and is a valid `ArcToken`. The only check performed is that the caller is the token admin:

TypeScript

```
    modifier onlyTokenAdmin(
        address _tokenContract
    ) {
        address adminRoleHolder = msg.sender;
        bytes32 adminRole = ArcToken(_tokenContract).ADMIN_ROLE();
        if (!ArcToken(_tokenContract).hasRole(adminRole, adminRoleHolder)) {
            revert NotTokenAdmin(adminRoleHolder, _tokenContract);
        }
        _;
    }
```

This check is insufficient, as anyone can deploy a contract with a `hasRole` function that always returns true, allowing them to call `enableToken` and list fake or malicious tokens for sale.

Recommendation

- Add a check in enableToken to ensure that the token was deployed by the official factory and is a valid `ArcToken`.
- Maintain a registry of valid tokens deployed by the factory and verify against it before enabling a token for sale.

IMM-HIGH-03

Unused `jackpotThreshold` in Reward Probability Calculation #24

Id	IMM-HIGH-03
Severity	High
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `Spin.sol::determineReward`

When determining the reward, the `jackpotThreshold` field in `rewardProbabilities` appears to be unused:

```
TypeScript
function determineReward(uint256 randomness, address user) internal view returns (string
memory, uint256) {
      // ... non relevant code for this issue ...
        uint256 jackpotThreshold = jackpotProbabilities[dayOfWeek];

        if (probability < jackpotThreshold) { // no use of
`rewardProbabilities.jackpotThreshold`
            return ("Jackpot", jackpotPrizes[weekNumber]);
        }

        // ... non relevant code for this issue ...
    }
```

As a result, the calculation for jackpot chances relies solely on `jackpotProbabilities`, which is set to very low values and does not reflect intended game dynamics.

Recommendation

- Either remove `jackpotThreshold` if it is unnecessary, or ensure it is properly integrated into the jackpot probability calculation.
- Review and adjust jackpot probability logic to match intended odds.

IMM-HIGH-04

Reward Rate Updates Are Applied Retroactively #13

Id	IMM-HIGH-04
Severity	High
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `RewardsFacet.sol::setRewardRates`, `PlumeRewardLogic.calculateRewardsWithCheckpoints`

When the `setRewardRates` function is called to update the reward rate for a token, it does not update users' `userValidatorRewardPerTokenPaid` values. As a result, when users later call `claim`, the reward calculation uses the new reward rate retroactively for the entire period since their last claim, rather than only for the period after the rate was updated.

For reference, in `setRewardRates`:

TypeScript

```
for (uint256 j = 0; j < validatorIds.length; j++) {
    uint16 validatorId = validatorIds[j];
    PlumeRewardLogic.updateRewardPerTokenForValidator($, token, validatorId);
    PlumeRewardLogic.createRewardRateCheckpoint($, token, validatorId, rate);
}
$.rewardRates[token] = rate;
```

No update is made to users' `userValidatorRewardPerTokenPaid` values, so the new rate is applied to all unclaimed rewards, regardless of when they were accrued.
This can result in users receiving more (or less) rewards than intended, depending on whether the reward rate was increased or decreased.

Recommendation

When updating the reward rate, ensure that users' reward accounting is updated so that the new rate only applies to rewards accrued after the change.

This can be done in two ways:

1. Update all users' `userValidatorRewardPerTokenPaid` to the current cumulative value before the rate change (not recommended as it requires more gas and risks running out of gas for large numbers of users)
2. Implement checkpoints for rewards calculations. This is the recommended approach.

IMM-HIGH-05

Cannot Claim Accrued Rewards for Deactivated Tokens #12

Id	IMM-HIGH-05
Severity	High
Category	Bug
Status	Fixed in 59c81c5f2a52f89fba1ed77bc3b37450f57d5097

Description

Code Affected: `RewardsFacet.sol::removeRewardToken`, `RewardsFacet.sol::claim`, `claimAll`

When a reward token is deactivated via `removeRewardToken`, the contract updates the reward state for that token using `updateRewardPerTokenForValidator`, ensuring that all rewards accrued up to that point are properly accounted for:

TypeScript

```
function removeRewardToken(
    address token
) external onlyRole(PlumeRoles.REWARD_MANAGER_ROLE) {
    PlumeStakingStorage.Layout storage $ = plumeStorage();
    if (!$.isRewardToken[token]) {
        revert TokenDoesNotExist(token);
    }

    // Find the index of the token in the array
    uint256 tokenIndex = _getTokenIndex(token);

    // 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);
    }
```

However, after deactivation, `$.isRewardToken[token]` is set to false. As a result, any attempt to claim rewards for the deactivated token will revert, since all claim functions check this flag:

TypeScript

```
    function claim(address token, uint16 validatorId) external nonReentrant returns (uint256)
{
        PlumeStakingStorage.Layout storage $ = plumeStorage();
        if (!$.isRewardToken[token]) {
            revert TokenDoesNotExist(token);
        }
    }
```

This prevents users from claiming rewards they rightfully earned before the token was removed.

Recommendation

- Allow users to claim any accrued rewards for tokens that have been deactivated, even after `removeRewardToken` is called.
- Consider tracking a separate state for "historical" reward tokens that allows claims but prevents new rewards from accruing.

IMM-HIGH-06

Unbounded Staker Count Can Prevent Validator Commission Claims #10

Id	IMM-HIGH-06
Severity	High
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `ValidatorFacet.sol::claimValidatorCommission`

The `claimValidatorCommission` function allows validators to claim their commission rewards. However, if a validator has more than 100 stakers, this function will revert due to its reliance on the underlying `_updateRewardsForAllValidatorStakers` function, which contains the following check:

TypeScript

```
if (stakers.length > 100) {
    revert TooManyStakers();
}
```

Currently, there is no restriction on the number of stakers that can delegate to a validator — only a limit on the total delegated amount (`maxCapacity`). This means a validator can accumulate more than 100 stakers, after which it becomes impossible to claim commission rewards, effectively bricking the validator's commission functionality.

Recommendation

- In general, avoid iterating over unbounded arrays, as this can lead to reverts due to gas limitations and denial of service.
- Implement a mechanism such as working checkpoints to track rewards without requiring iteration over the entire stakers array.
- If iteration is unavoidable, enforce an upper bound (e.g., 100) on the number of stakers per validator by checking the array length before allowing new stakers to join.

IMM-MED-01

Seller Does Not Receive Revenue from ArcToken Sales #37

Id	IMM-MED-01
Severity	Medium
Category	Bug
Status	Acknowledged

Description

Code Affected: `ArcTokenPurchase.sol::buy`

When an `ArcToken` is bought, the seller does not receive the revenue from the sale. Instead, the buyer's tokens are transferred to the `ArcTokenPurchase` contract itself, rather than being forwarded to the intended seller or beneficiary.

The only functions to withdraw tokens from the `ArcTokenPurchase` contract are controlled by the `ArcTokenPurchase` admin and not the token seller.

This results in the seller not being compensated for the tokens sold, and the funds remaining locked in the contract.

Recommendation

Ensure that, upon purchase, the payment is correctly forwarded to the seller or designated beneficiary.

IMM-MED-02

Unsold ArcToken Withdrawal Restricted to Contract Admin #36

Id	IMM-MED-02
Severity	Medium
Category	Bug
Status	Fixed in 57595d2c983af9db583cc48b52aac5339af1a4bc

Description

Code Affected: `ArcTokenPurchase.sol::withdrawUnsoldArcTokens`

The `withdrawUnsoldArcTokens` function in `ArcTokenPurchase` is only callable by the contract admin, not by the `ArcToken` admin. This restricts the ability to withdraw unsold `ArcTokens`, potentially preventing the `ArcToken` admin from managing their own token supply after a partial sale.

Recommendation

- Provide a mechanism for the `ArcToken` admin to withdraw unsold `ArcTokens`.
- In general, provide a mechanism for the `ArcToken` admin to manage their token sale (such as setting `isEnabled` to `false`, etc).

IMM-MED-03

Missing `minStakeAmount` Enforcement in `restakeRewards` #20

Id	IMM-MED-03
Severity	Medium
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `StakingFacet.sol::restakeRewards`

The `restakeRewards` function does not enforce the `minStakeAmount` constraint for validators. As a result, users can restake rewards that are less than the minimum stake amount, bypassing the restriction that is properly enforced in other staking functions such as stake, restake, and `stakeOnBehalf`.

Recommendation

- Add a validation check in `restakeRewards` to ensure that the amount being restaked is greater than or equal to `minStakeAmount`.
- Refactor staking-related functions to share core logic.

IMM-MED-04

Single Admin Can Only Vote With One Validator Due to Mapping Overwrite #19

Id	IMM-MED-04
Severity	Medium
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `ValidatorFacet.sol::addValidator`, `ValidatorFacet.sol::voteToSlashValidator`

The `addValidator` function does not check whether `adminToValidatorId[l2AdminAddress]` is already set. This allows the same admin address to be assigned to multiple validators. However, each new assignment overwrites the previous mapping.

This has a critical impact on the slashing process. The `voteToSlashValidator` function relies on the `adminToValidatorId` mapping to determine the identity of the voting validator:

TypeScript
```
address voterAdmin = msg.sender;
uint16 voterValidatorId = $.adminToValidatorId[voterAdmin];
// ...
if ($.validators[voterValidatorId].l2AdminAddress != voterAdmin ||
!$.validators[voterValidatorId].active) {
    revert NotValidatorAdmin(voterAdmin);
    }
if (voterValidatorId == maliciousValidatorId) {
    revert CannotVoteForSelf();
}
```

If an honest admin controls multiple validators, they will only be able to vote using one of their validators, not all of them. This prevents unanimity from being reached in `slashValidator`, potentially allowing a malicious validator to evade slashing.

Recommendation

In `addValidator`, check that `adminToValidatorId[l2AdminAddress]` is not already set to prevent assigning the same admin to multiple validators.
Or alternatively, allow multiple validators per admin, maybe via a mapping new `adminToValidatorIds` address => uint16[] array.

B `bondETH`

IMM-MED-05

Users Lose Rewards When Calling restakeRewards #18

Id	IMM-MED-05
Severity	Medium
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `StakingFacet.sol::restakeRewards`

During the `restakeRewards` process, the function retrieves the pending reward delta using `getPendingRewardForValidator`, but then zeroes out the entire reward balance for the user, causing users to lose any previously accrued (but unclaimed) rewards:

```
TypeScript
uint256 validatorReward =
    RewardsFacet(payable(address(this))).getPendingRewardForValidator(msg.sender,
userValidatorId, token);

if (validatorReward > 0) {
    amountRestaked += validatorReward;

    PlumeRewardLogic.updateRewardsForValidator($, msg.sender, userValidatorId);

    $.userRewards[msg.sender][userValidatorId][token] = 0; // Zero out the entire reward
balance without using it for restaking

    emit RewardClaimedFromValidator(msg.sender, token, userValidatorId, validatorReward);
}
```

As a result, any rewards accrued prior to the current delta are lost when a user calls `restakeRewards`.

Recommendation

Ensure that all unclaimed rewards (not just the current delta) are included when restaking, so users do not lose previously accrued rewards.

IMM-MED-06

Validator Commission Rate Can Be Set to 100% #17

Id	IMM-MED-06
Severity	Medium
Category	Bug
Status	Fixed in a192bbcdc4009c3f48d23d2840a99695785b6031

Description

Code Affected: `ValidatorFacet.sol::setValidatorCommission`, `ValidatorFacet.sol::addValidator`

There is currently no restriction preventing a validator's commission rate from being set to 100%. Both the `addValidator` and `setValidatorCommission` functions allow any value up to and including 100% (represented as `1e18`), meaning a validator can claim all rewards for themselves and leave stakers with nothing.

If the commission rate is set to 100%, users who stake with that validator will not receive any rewards and thus the validator won't receive any rewards either (since the commission is taken only when the user claims their rewards and the reward is not 0).

Recommendation

Impose an upper limit on the validator commission rate (e.g., less than 100%, for example 30%) in both `addValidator` and `setValidatorCommission`.

IMM-MED-07

Slashed Validators Are Still Useable In the Protocol #16

Id	IMM-MED-07
Severity	Medium
Category	Bug
Status	Fixed in 3d899831cf556f4d7d3311f830c439d0fa6f2b16

Description

Code Affected: Multiple locations (e.g., `StakingFacet.sol::stake`, `ValidatorFacet.sol::claimValidatorCommission`)

After a validator is slashed, its status is set to `validator.slashed = true` and `validator.active = false`. However, many protocol functions do not consistently check whether a validator is slashed or inactive before allowing further actions.

For example:
Users can still stake a slashed validator, even though it is no longer considered active.
A slashed validator can still call `claimValidatorCommission` and withdraw accrued commission rewards.
This undermines the intended consequences of slashing and can lead to unexpected or insecure protocol behavior.

Recommendation

- Ensure that all functions which interact with validators (such as staking, claiming rewards, or updating validator data) include checks to prevent actions involving slashed or inactive validators.
- Consider centralizing these checks in modifiers or internal helpers to enforce consistent validation across the codebase.

IMM-MED-08

Temporary Slashing Evasion via Admin Assignment #15

Id	IMM-MED-08
Severity	Medium
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `ValidatorFacet.sol::setValidatorAddresses`

The `setValidatorAddresses` function allows updating a validator's admin address without verifying that the `newL2AdminAddress` has consented to this role. If the `newL2AdminAddress` is already the admin of another validator, this operation will overwrite the adminToValidatorId mapping.

This has a critical impact on the slashing process. The `voteToSlashValidator` function relies on the `adminToValidatorId` mapping to determine the identity of the voting validator. A malicious validator can assign another validator's admin as their own, preventing that admin from voting to slash the malicious validator due to the following check:

TypeScript

```
address voterAdmin = msg.sender;
uint16 voterValidatorId = $.adminToValidatorId[voterAdmin];

// ...

if (voterValidatorId == maliciousValidatorId) {
    revert CannotVoteForSelf();
}
```

This prevents unanimity from being reached in `slashValidator`, effectively allowing the malicious validator to evade slashing.

Since the victim admin can restore their voting rights by reassigning their admin address through an intermediate step, this issue is not critical.

Recommendation

Require explicit consent from the `newL2AdminAddress` before assigning them as a validator admin. This can be achieved by requiring a signature from the new admin or by implementing an `acceptAdmin` function that the new admin must call to confirm the assignment.

IMM-MED-09

Missing `maxCapacity` Enforcement in `restakeRewards` #14

Id	IMM-MED-09
Severity	Medium
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `StakingFacet.sol::restakeRewards`

The `restakeRewards` function does not enforce the `maxCapacity` constraint for validators. As a result, users can restake rewards and exceed the intended maximum delegation limit for a validator, bypassing the restriction that is properly enforced in other staking functions such as stake, restake, and `stakeOnBehalf`.

Recommendation

- Add a validation check in `restakeRewards` to ensure that if `maxCapacity` is not zero, the new delegated amount does not exceed `maxCapacity`.
- Refactor staking-related functions to share core logic.

IMM-LOW-01

Restriction Contracts Are Not Used as Upgradeable Proxies #39

Id	IMM-LOW-01
Severity	LOW
Category	Bug
Status	Fixed in 23bd73bde60fe5aef38d383f492dee1cb7b24d86

Description

Code Affected: `ArcTokenFactory.sol::createToken`, `WhitelistRestrictions.sol`, `YieldBlacklistRestrictions.sol`

When creating a new token, the `ArcTokenFactory` also creates new instances of `WhitelistRestrictions` and `YieldBlacklistRestrictions` as upgradeable proxies. However, the `ArcToken` contract uses the implementation contracts directly, rather than deploying and interacting with proxy instances.

As a result, the implementation contracts are initialized and used directly. Due to the `onlyProxy` modifier from `UUPSUpgradeable`, upgrade attempts on the implementation contracts will revert:

TypeScript

```
function _checkProxy() internal view virtual {
    if (
        address(this) == __self || // Must be called through delegatecall
        ERC1967Utils.getImplementation() != __self // Must be called through an active proxy
    ) {
        revert UUPSUnauthorizedCallContext();
    }
}
```

Since the implementations are not used as proxies, the `_checkProxy` function will revert due to the `address(this) == __self check`.

Recommendation

Use upgradeable proxies for the `WhitelistRestrictions` and `YieldBlacklistRestrictions` instances as intended.

IMM-LOW-02

Distributor Can Direct Yield to Specific Holders #38

Id	IMM-LOW-02
Severity	LOW
Category	Bug
Status	Acknowledged

Description

Code Affected: `ArcToken.sol::distributeYieldWithLimit`

The `YIELD_DISTRIBUTOR_ROLE` in the `ArcToken` contract can send arbitrary yield to any specific holder by using the `distributeYieldWithLimit` function. This allows the distributor to allocate all yield to a single holder, rather than distributing it fairly among all holders.

For example, the distributor can call:

TypeScript

```
distributeYieldWithLimit(totalAmount, startIndex, maxHolders)
// where totalAmount = (x * effectiveTotalSupply) / holderBalance, startIndex = i, maxHolders = 1
```

and transfer `x` yield tokens to the contract beforehand. This results in only the selected holder receiving the yield, while other holders may receive nothing.

This undermines the fairness and intended distribution of yield among all token holders.

Recommendation

Consider using checkpoint mechanisms to ensure fair distribution of yield.

IMM-LOW-03

Spin Does Not Auto-End After 12 Weeks #27

Id	IMM-LOW-03
Severity	LOW
Category	Bug
Status	Acknowledged

Description

Code Affected: `Spin.sol` - Spin duration and reward logic

The spin game does not automatically end after 12 weeks, allowing non-jackpot rewards to continue being farmed indefinitely.

Recommendation

Implement logic to automatically end the spin game after the intended duration (e.g., 12 weeks).

IMM-INSIGHT-01

Code Redundancy, Dead Code, and Code Duplication in Arc #44

Id	IMM-INSIGHT-01
Severity	INSIGHT
Category	Informational
Status	Fixed in 303694c82cad48cc5788d73b200159887a71318f

Description

Code Affected: Multiple locations (e.g., `ArcToken.sol`)

There are various instances of code redundancy, dead code, and duplication throughout the project, which can lead to maintenance challenges, increased risk of bugs, and unnecessary contract size.

Redundancy

1. `previewYieldDistributionWithLimit` has `amounts = new uint256[](batchSize);` twice
2. In `ArcTokenFactory.sol` the `restrictionsRouter` storage variable is redundant, just use `fs.restrictionsRouter` and create a view function to return it.

Dead Code

`RestrictionsFactory` is not used at all anywhere.

Duplication

1. The underlying login in all of `ArcToken.sol::previewYieldDistribution`, `ArcToken.sol::previewYieldDistributionWithLimit`, `ArcToken.sol::distributeYield`, and `ArcToken.sol::distributeYieldWithLimit` is the same.
2. `TRANSFER_RESTRICTION_TYPE` and `YIELD_RESTRICTION_TYPE` are defined both in `ArcToken.sol` and `ArcTokenFactory.sol`.

Recommendation

Remove duplicated and redundant code to improve maintainability and reduce contract size.

IMM-INSIGHT-02

ArcToken Implementation Can Be Initialized by Anyone #43

Id	IMM-INSIGHT-02
Severity	INSIGHT
Category	Informational
Status	Acknowledged

Description

Code Affected: `ArcTokenFactory.sol::createToken`, `ArcToken.sol`

In `ArcTokenFactory.sol`, the implementation contracts for `ArcTokens` are never initialized. This allows an attacker to call initialize on the implementation contract directly. Thanks to the But, the `onlyProxy` modifier from the `UUPSUpgradeable` contract, implementations cannot be upgraded.
The `onlyProxy` modifier `calls _checkProxy` function:

```TypeScript
    function _checkProxy() internal view virtual {
        if (
            address(this) == __self || // Must be called through delegatecall
            ERC1967Utils.getImplementation() != __self // Must be called through an active proxy
        ) {
            revert UUPSUnauthorizedCallContext();
        }
    }
```

Since the implementations are not used as proxies, the `_checkProxy` function will revert due to the `address(this) == __self` check.
This check prevents unauthorized upgrades or self-destruction on the implementation contract. Which makes it safe to not initialize the implementation contract.

But, it is best practice to prevent any initialization of the implementation contract for safety.

Recommendation

- Either `Call _disableInitializers()` in the constructor of the `ArcToken` implementation (and in general for all upgradeable contracts) - This is the best practice.
- Or, call `initialize` on the implementation contract after deployment.

IMM-INSIGHT-03

Use of `initializer` Instead of `reinitializer` in Upgradeable Contract #42

Id	IMM-INSIGHT-03
Severity	INSIGHT
Category	Informational
Status	Acknowledged

Description

Code Affected: Upgradeable contract initialization logic

All of the upgradeable contracts use the `initializer` modifier to protect its initialization function. While this prevents re-initialization, it does not support future upgrades that may require additional initialization steps

Since the contract is upgradeable, using `reinitializer(version)` is recommended to allow safe, versioned initialization for future upgrades.

Recommendation

Replace the `initializer` modifier with `reinitializer(version)` in upgradeable contracts to support safe, versioned initialization during upgrades.

IMM-INSIGHT-04

Preview Functions Are Not Marked as `view` #41

Id	IMM-INSIGHT-04
Severity	INSIGHT
Category	Informational
Status	Fixed in 2360d5e6d0124e1b806b7ce7b77f0790ef924d47

Description

Code Affected: `ArcToken.sol::previewYieldDistribution`, `previewYieldDistributionWithLimit`

In `ArcToken.sol`, the functions `previewYieldDistribution` and `previewYieldDistributionWithLimit` are intended to provide a read-only preview of yield distribution outcomes. However, these functions are not marked as views.

Recommendation

Mark `previewYieldDistribution` and `previewYieldDistributionWithLimit` as view functions to clearly indicate that they do not modify contract state.

IMM-INSIGHT-05

Deployer Cannot Manage Roles After Token Creation #40

Id	IMM-INSIGHT-05
Severity	INSIGHT
Category	Informational
Status	Fixed in 0dd791f5a8b37ce85f7bb8d8f0e09c05c8d5ac15

Description

Code Affected: `ArcTokenFactory`, role assignment logic

When creating a new token, the `ArcTokenFactory` grants all major roles to the deployer `(msg.sender`):

```TypeScript
token.grantRole(token.ADMIN_ROLE(), msg.sender);
token.grantRole(token.MANAGER_ROLE(), msg.sender);
token.grantRole(token.YIELD_MANAGER_ROLE(), msg.sender);
token.grantRole(token.YIELD_DISTRIBUTOR_ROLE(), msg.sender);
token.grantRole(token.MINTER_ROLE(), msg.sender);
token.grantRole(token.BURNER_ROLE(), msg.sender);
token.grantRole(token.UPGRADER_ROLE(), msg.sender);
```

However, the deployer is not granted the `DEFAULT_ADMIN_ROLE`, which is required to grant or revoke roles. As a result, the deployer cannot delegate or renounce these roles, leading to a risk of centralization and lack of flexibility in role management.

Recommendation

- Grant the DEFAULT_ADMIN_ROLE to the deployer so they can manage other roles.
- Alternatively, set the ADMIN_ROLE as the admin of the other roles to allow for proper delegation and management.

IMM-INSIGHT-06

Code Redundancy, Dead Code, and Code Duplication #30

Id	IMM-INSIGHT-06
Severity	INSIGHT
Category	Informational
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: Multiple locations (e.g., `PlumeStaking.sol::initialize`, reward logic functions, staking functions)

There are various instances of code redundancy, dead code, and duplication throughout the project, which can lead to maintenance challenges, increased risk of bugs, and unnecessary contract size.

Redundancy

1. In `PlumeStaking.sol::initialize`, the following snippet is duplicated within the same function:

TypeScript
```
$.minStakeAmount = minStake;
$.cooldownInterval = cooldown;
$.initialized = true;
```

2. In reward rate updates, `updateRewardPerTokenForValidator` is called before `createRewardRateCheckpoint`, but the latter already calls the former internally, making the first call redundant:

TypeScript
```
PlumeRewardLogic.updateRewardPerTokenForValidator($, token, validatorId);
PlumeRewardLogic.createRewardRateCheckpoint($, token, validatorId, rate); // Use library
```

Dead Code

1. The function `createRewardRateCheckpoint` updates `$.validatorRewardRateCheckpoints`, but this variable is never used elsewhere, making the update redundant and the function practically dead

code.

2. In `Raffle.sol` the `onlyAdmin`, and `onlySupra` modifiers are not used. The `onlyRole` modifier is used instead.
3. The `_isRewardToken` in `RewardsFacets.sol` is not used.
4. In `PlumeRewardLogic.sol` the `findCheckpointIndex` function is not used.
5. In `PlumeRewardLogic.sol` the `calculateRewardsByIteratingCheckpoints` function is not used.
6. In `PlumeRewardLogic.sol` the `createCommissionRateCheckpoint` function is used to create a commission rate checkpoint. But the checkpoint is not used anywhere else.
7. In `PlumeRewardLogic.sol` the `createRewardRateCheckpoint` function is used to create a `validatorRewardRateCheckpoints` checkpoint. But:
    a. The `validatorRewardRateCheckpoints` is used to set `userLastCheckpointIndex` and in 2 view functions.
    b. The `userLastCheckpointIndex` is used in `getUserLastCheckpointIndex` which is a get function not used anywhere else.
    c. So to summarize, the `createRewardRateCheckpoint` is not really affecting any logic.
8. In `PlumeValidatorLogic.sol` the `getValidatorInfo` function is not used.
9. In `PlumeValidatorLogic.sol` the `isValidatorActive` function is not used.
10. In `PlumeValidatorLogic.sol` the `getValidatorTotalStaked` function is not used.

The following storage variables are not used:

- `rewardPerTokenCumulative`
- `totalCooling`
- `totalWithdrawable`
- `userRewardPerTokenPaid`
- `validatorTotalCooling`
- `validatorTotalWithdrawable`
- `usingEpochs`
- `currentEpochNumber`
- `epochValidatorAmounts`
- `maxValidatorCommission`
- `maxValidatorPercentage`
- `userValidatorStakeStartTime`
- `hasRole`
- `slashVoteCounts`

The following storage variables are "used" but not actually affecting any logic:

- lastUpdateTimes
- rewardsAvailable
- totalClaimableByToken
- rewards
- validatorTotalStaked
- userLastCheckpointIndex

Duplication

1. The TREASURY_STORAGE_POSITION is being declared in multiple facets. This is an unnecessary risk and all facets should use the same centralized source of truth.
2. PLUME_NATIVE and PLUME addresses are being declared in multiple facets.
3. REWARD_PRECISION is being declared in multiple facets.
4. The onlyRole modifier is being used in multiple facets, instead declaring it one in a common library or a contract that is inherited by all facets.
5. _getPlumeStorage is being used in multiple facets, instead using the layout function from the PlumeStakingStorage library.

Recommendation

- Remove duplicated and redundant code to improve maintainability and reduce contract size.
- Eliminate dead code and unused variables, such as validatorRewardRateCheckpoints, to prevent confusion and potential errors.
- Any common logic should be extracted to a helper function.

IMM-INSIGHT-07

No way to deactivate a whitelisted account #29

Id	IMM-INSIGHT-07
Severity	INSIGHT
Category	Informational
Status	Fixed in 8c295373bfd6c57602539f7add856af1280cf576

Description

Code Affected: `Spin.sol`

After a user is whitelisted, there is no way to remove them from the whitelist.
This can be problematic if a user's private key is compromised or if they want to leave the whitelist or any other reason.

Recommendation

Add a function to remove a user from the whitelist.

IMM-INSIGHT-08

Deactivated Prizes via `setPrizeActive` Remain in Prize List #28

Id	IMM-INSIGHT-08
Severity	INSIGHT
Category	Informational
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `Raffle.sol::setPrizeActive`

When a prize is deactivated using `setPrizeActive`, it is not removed from the `prizeIds` array or list. This can result in deactivated prizes still appearing in prize selection logic or user interfaces, potentially causing confusion or unintended behavior.

Recommendation

- When deactivating a prize via `setPrizeActive`, also remove its ID from the `prizeIds` array or list.
- Remove the `setPrizeActive` function entirely.

IMM-INSIGHT-09

No Retry Mechanism for Failed Prize Transfers #26

Id	IMM-INSIGHT-09
Severity	INSIGHT
Category	Informational
Status	Acknowledged

Description

Code Affected: `Spin.sol::_safeTransferPlume`

If `_safeTransferPlume` reverts, there is no retry mechanism in place.
It can lead to lost rewards if a transfer fails due to temporary issues.

Recommendation

Consider implementing a retry mechanism or a way for users to manually claim failed transfers.

IMM-INSIGHT-10

Multiple Randomness Requests Possible in Raffle #25

Id	IMM-INSIGHT-10
Severity	INSIGHT
Category	Informational
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `Raffle.sol::requestWinner`, `Raffle.sol::handleWinnerSelection`

The `requestWinner` function in the Raffle contract can be called multiple and only stops being callable after a `handleWinnerSelection` callback is processed.

This allows the creation of multiple randomness requests and can potentially overwrite the winner.

Since the owner is the only one who can call requestWinner, this is not a huge problem in the current implementation.

But it could be a problem if the owner is not careful.

Recommendation

Add a flag that indicates that `requestWinner` was called for a specific `prizeId` and prevent calling it again if that is the case.

IMM-INSIGHT-11

Unrestricted Initialization of `minStakeAmount` #22

Id	IMM-INSIGHT-11
Severity	INSIGHT
Category	Informational
Status	Fixed in 297b5a76dd31c1d97c625bdb89e83560595ce6f7

Description

Code Affected: `PlumeStaking.sol::initializePlume`

The `initializePlume` function currently allows the `minStakeAmount` parameter to be set to zero during contract initialization. This can lead to undesirable behavior, such as allowing users to register as stakers in a validator context without being included in the `$.stakers` array.

While the initialization is restricted to the contract owner, the absence of a minimum value check introduces unnecessary risk and could result in subtle bugs or inconsistencies if `minStakeAmount` is ever set to zero, either accidentally or due to a misconfiguration.

Recommendation

Add a validation check in the `initializePlume` function to ensure that `minStakeAmount` is strictly greater than zero.

IMM-INSIGHT-12

Inconsistent Storage Access in AccessControlFacet #21

Id	IMM-INSIGHT-12
Severity	INSIGHT
Category	Informational
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `AccessControlFacet.sol`

The `AccessControlFacet` currently defines and uses its own storage variable `_initializedAC`, which is not part of the shared `PlumeStaking` storage layout located at slot `keccak256("plume.storage.PlumeStaking")`.

This practice is unsafe in the context of diamond storage. Any future facet or modification that assumes a consistent storage layout could unintentionally overwrite this slot or use it with a different meaning - leading to hard-to-detect bugs and unexpected behavior.

Recommendation

Define `_initializedAC` inside the PlumeStaking storage struct and access it via the shared storage layout. This ensures safe and consistent access across all facets in compliance with the diamond storage pattern.

AUDIT 2 - FINDINGS

IMM-CRIT-01

No `isNewStake` Handling in `restakeRewards` Enables Theft #48

Id	IMM-CRIT-01
Severity	Critical
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `StakingFacet.sol::restakeRewards`

No `isNewStake` flow in `restakeRewards` allows theft of funds. A user can stake to a new validator via `restakeRewards` but since there is no `isNewStake` check, they skip updating their reward checkpoint update. This can allow the user to steal rewards they don't deserve.

(Since during the reward calculation `lastUserRewardUpdateTime` falls back to `block.timestamp` the flow to exploit this issue is a bit more complicated, but it is still possible so this is very important to fix)

Recommendation

Add `isNewStake` handling in the `restakeRewards` function

IMM-CRIT-02

Slashing Can Be Prevented via Gas Exhaustion #47

Id	IMM-CRIT-02
Severity	Critical
Category	Bug
Status	Fixed in `3a745d505563afaf5093025f65d1195d0955a772`

Description

Code Affected: `ValidatorFacet.sol::slashValidator`

The `slashValidator` function loops over all of `validatorStakers` and then internally every reward token. A malicious validator can use a large number of stakers to exploit this to consume all gas for slashing attempts. Consider enforcing a max number of `validatorStakers`.

Recommendation

Implement a maximum limit on the number of stakers per validator

IMM-CRIT-03

Duplicate timestamp Commission Checkpoints Can Lead to Theft #46

Id	IMM-CRIT-03
Severity	Critical
Category	Bug
Status	Fixed in 9f1e5483e289c9d45d3ebedd3b62c4e88ea3417c

Description

Code Affected: `PlumeRewardLogic.sol::getEffectiveCommissionRateAt`, `PlumeRewardLogic.sol::findCommissionCheckpointIndexAtOrBefore`

When creating a new checkpoint of any kind, there is no check to see if the timestamp already exists. This means that a validator can create multiple checkpoints with the same timestamp, which will cause the binary search to return a different checkpoint for the same timestamp for different users.

The `getEffectiveCommissionRateAt` function relies on `findCommissionCheckpointIndexAtOrBefore`, which uses binary search under the assumption that `validatorCommissionCheckpoints` contains distinct timestamps. However, a validator can insert multiple commission changes within the same block, all sharing the same timestamp. This causes inconsistent search results, potentially returning a 0% commission for users and the maximum for the validator—allowing them to claim an unfair share of rewards.

Side Note: `validatorRewardRateCheckpoints` can also contain entries with the same timestamp, but this is controlled by you guys so it's less of an issue, but should be fixed as well.

Recommendation

If creating a new checkpoint in the same block, overwrite the last checkpoint instead of creating a new one.

IMM-CRIT-04

Excessive Commission Changes Can Lock Funds #45

Id	IMM-CRIT-04
Severity	Critical
Category	Bug
Status	Fixed in 9d49b09f7a6591bbf0aa5bf9e2131869e2186f81

Description

Code Affected: `PlumeRewardLogic.sol::calculateRewardsWithCheckpoints`

In `calculateRewardsWithCheckpoints`, the `distinctTimestamps` array is built from all commission checkpoints and reward checkpoints and later iterated over.

A malicious validator can create many commission checkpoints to bloat this array and effectively block users from claiming rewards. This can also happen by accident with time.

Recommendation

- Implement a maximum limit on the number of commission checkpoints a validator can create
- Consider implementing a cleanup mechanism for old checkpoints

IMM-CRIT-05

Users Can Bypass Slashing Penalties #69

Id	IMM-CRIT-05
Severity	Critical
Category	Bug
Status	Fixed in b1f7364252f28d7eb342a26b846fedf34c8c0ace

Description

Code Affected: `StakingFacet.sol::restake`

The `restake` function allows users to move funds that are in the cooling or parked state to a different validator. This opens up a loophole where users can effectively avoid slashing.

Example Scenario

Suppose Validator A is about to be slashed in the next block. A user with staked funds on Validator A can:

1. Initiate an `unstake` from Validator A.
2. Immediately call `restake` to move the funds to Validator B.

Even though the `unstake` request hasn't completed the cooldown period and is not yet withdrawable, the funds can still be restaked elsewhere. This allows users to move their funds away from slashing risk and effectively shield them, rendering the slashing mechanism ineffective.

Recommendation

Addressing this issue likely requires a redesign of the `restake` logic to ensure that slashing applies consistently, even during cooldown periods.

IMM-CRIT-06

Validator Can Bypass Slashing Penalties #70

Id	IMM-CRIT-06
Severity	Critical
Category	Bug
Status	Fixed in 747f7156836522d908efa25feb532565c7df58b7

Description

Code Affected: `ValidatorFacet.sol::finalizeCommissionClaim`

The `finalizeCommissionClaim` function is intended to prevent slashed validators from withdrawing commission rewards that were initiated after the voting against them started. However, it reverts under the following condition:

TypeScript

```
if (validator.slashed && claim.requestTimestamp >= validator.slashedAtTimestamp) {
    revert ValidatorInactive(validatorId);
}
```

`claim.requestTimestamp` refers to when the claim was initiated-not when the funds become withdrawable.

As a result, a validator can initiate a commission claim at any point before being slashed (even during the voting period), and then simply wait for the cooldown to pass before finalizing the claim and withdrawing funds. This effectively allows them to escape the consequences of slashing.

Recommendation

Update the condition to ensure that slashing applies retroactively to any pending commission claims. Specifically, check that:

TypeScript

```
claim.requestTimestamp + PlumeStakingStorage.COMMISSION_CLAIM_TIMELOCK <
validator.slashedAtTimestamp
```

This ensures that only claims that completed their timelock before the validator was slashed can be finalized.

IMM-HIGH-01

No New Reward Checkpoint on Token Remove #50

Id	IMM-HIGH-01
Severity	High
Category	Bug
Status	Fixed in 69b384873c6c9aeabf27c064248699f271eb248e

Description

Code Affected: `RewardsFacet.sol::removeRewardToken`

When a reward token is removed, no new `RewardRateCheckpoint` is created. This allows users to continue accumulating rewards for the removed token, which should not be possible.

Recommendation

Create a final checkpoint when removing a reward token

IMM-HIGH-02

Staker Not Removed After Claim #49

Id	IMM-HIGH-02
Severity	High
Category	Bug
Status	Fixed in `3a745d505563afaf5093025f65d1195d0955a772`

Description

Code Affected: `RewardsFacet.sol::claim`

In `claim(address token, uint16 validatorId)`, the staker is not removed from the validator. This can lead to stale data in the validator's staker list and potential issues with future operations.

Recommendation

Implement proper staker removal after successful claim

IMM-MED-01

Winner Selection State Not Reset on Handler Failure #66

Id	IMM-MED-01
Severity	Medium
Category	Bug
Status	Fixed in 64be0a1e4876c0e9bd56cecd0584b119f45e4f31

Description

Code Affected: `Raffle.sol::handleWinnerSelection`

If `handleWinnerSelection` fails, there can never be a winner for that prizeId since `isWinnerRequestPending` will remain true and prevent calls to `requestWinner`.

Recommendation

Consider implementing a timeout mechanism for pending winner selections

IMM-MED-02

Spin State Not Reset on Randomness Handler Failure #65

Id	IMM-MED-02
Severity	Medium
Category	Bug
Status	Fixed in 08fa565d261fbe2874a8bf2aac7070f77fee6773

Description

Code Affected: `Spin.sol::handleRandomness`

If `handleRandomness` fails (for example the plume transfer reverts) `isSpinPending[msg.sender]` will remain true forever and the user will never be able to spin again.

Recommendation

Consider implementing a timeout mechanism for pending spins

IMM-MED-03

Commission Might Be Able to Claim Before Slashing Period Ends #51

Id	IMM-MED-03
Severity	Medium
Category	Bug
Status	Fixed in `f9c053b2b1683691ce35c36ba7e138e7a2807369`

Description

Code Affected: `RewardsFacet.sol::requestCommissionClaim` and `RewardsFacet.sol::finalizeCommissionClaim`

`maxSlashVoteDurationInSeconds` can exceed `COMMISSION_CLAIM_TIMELOCK`, letting validators claim commission even when under active slashing vote. This could allow them to escape slashing consequences.

Recommendation

Ensure `maxSlashVoteDurationInSeconds` is always less than `COMMISSION_CLAIM_TIMELOCK`

IMM-LOW-01

Validator Commission Checks Not Enforced on `maxAllowedValidatorCommission` Update #61

Id	IMM-LOW-01
Severity	LOW
Category	Bug
Status	Fixed in 0523300203430aa7b7610102d9afe13b20307ef7

Description

Code Affected: `ManagementFacet.sol::setMaxAllowedValidatorCommission`

When `maxAllowedValidatorCommission` is changed, there is no check that all validator commissions are less than the newly set `maxAllowedValidatorCommission`, allowing the existence of validators with a commission higher than `maxAllowedValidatorCommission`.

Recommendation

During `setMaxAllowedValidatorCommission`, update all violating validators to the new maximum

IMM-LOW-02

Incorrect `totalCooling` Decrease on Slashing #57

Id	IMM-LOW-02
Severity	LOW
Category	Bug
Status	Fixed in 86af6d4c51b426f0adf621244f117bbe8d1d4d0d

Description

Code Affected: `PlumeStaking.sol::slashValidator`

When slashing a validator, `totalCooling` is being decreased by the entire amount of `validatorTotalCooling` for the malicious validator. But, if a user unstaked before the slashing vote period even started (in other words - unstaked in time) their unstake is valid and once they withdraw their funds, `totalCooling` will decrease again, effectively decreasing twice for the same amount.

This also has the side effect of making `validatorTotalCooling` inaccurate for the same reason.

Recommendation

Implement proper tracking of cooling amounts for slashed validators

IMM-LOW-03

Expired Votes Not Properly Filtered #55

Id	IMM-LOW-03
Severity	LOW
Category	Bug
Status	Fixed in aebcbd127a932bed3bb6c3e6dfc4cd0df7de5902

Description

Code Affected: `ValidatorFacet.sol::_cleanupExpiredVotes`

`_cleanupExpiredVotes` doesn't validate that the `voterValidatorId` is active and not slashed, allowing stale or invalid votes to linger.

Recommendation

Add validation to check validator status before processing votes

IMM-LOW-04

Votes Can Be Cleaned Up to Avoid Slashing #54

Id	IMM-LOW-04
Severity	LOW
Category	Bug
Status	Fixed in 93dd3e5bdbff73b5245ce8f3adbf4511708feaae

Description

Code Affected: `ValidatorFacet.sol::cleanupExpiredVotes`, `ValidatorFacet.sol::voteToSlashValidator`

If slashing isn't triggered before the voting window expires, a malicious validator can call `cleanupExpiredVotes` to remove stale votes and evade punishment. Consider auto-triggering slashing during `voteToSlashValidator` if the vote count is sufficient.

Recommendation

Implement auto-triggering of slashing when vote threshold is reached

IMM-LOW-05

Validator Creation Can Be Blocked From Being Created #53

Id	IMM-LOW-05
Severity	LOW
Category	Bug
Status	Fixed in 38292d32a941d786a45eef8ae5ca67bdc4f7e6bc

Description

Code Affected: `ValidatorFacet.sol::addValidator`, `ValidatorFacet.sol::setValidatorAddresses`

A malicious validator can frontrun an `addValidator` transaction by calling `setValidatorAddresses`, setting the new validator's admin to their own. This causes the `addValidator` call to revert due to `isAdminAssigned`.

Recommendation

Implement a "acceptAdmin" function that allows the new admin to accept the admin role before switching it to the new admin.

IMM-LOW-06

Claim Reverts on Slashed Validator #52

Id	IMM-LOW-06
Severity	LOW
Category	Bug
Status	Fixed in 3a745d505563afaf5093025f65d1195d0955a772

Description

Code Affected: `RewardsFacet.sol::claim`, `RewardsFacet.sol::_validateValidatorForClaim`

When calling `claim(address token, uint16 validatorId)`, `_validateValidatorForClaim reverts` if the validator is slashed, even though rewards may still be claimable. (the other claim function makes this less of an issue, but still)

Recommendation

Modify `_validateValidatorForClaim` to allow claims for slashed validators

IMM-LOW-07

Incorrect Handling of Slashed Validators in `updateRewardPerTokenForValidator` #71

Id	IMM-LOW-07
Severity	LOW
Category	Bug
Status	Fixed in 7edae49a1afdb948b227f3d80ab592b7b7d4f761

Description

Code Affected: `PlumeRewardLogic.sol::updateRewardPerTokenForValidator`

The function currently checks for inactive validators before checking for slashed ones:

TypeScript
```
if (!validator.active) {
    ...
    return;
}
// --- END INACTIVE CHECK ---

// --- BEGIN SLASH CHECK ---
if (validator.slashed) {
    ...
}
```

However, since slashed validators are also marked as inactive, the slashed check is never reached. This results in slashed validators being treated as merely inactive.

Recommendation

Reorder the conditional checks so that slashing is handled before inactivity.

IMM-LOW-08

`clearPendingRewardsFlagIfEmpty` Only Checks for `PLUME_NATIVE` #72

Id	IMM-LOW-08
Severity	LOW
Category	Bug
Status	Fixed in a17465bec3c22d915556ff44ecde6c882213bba4

Description

Code Affected: `PlumeRewardLogic.sol::clearPendingRewardsFlagIfEmpty`

The `clearPendingRewardsFlagIfEmpty` function currently checks only for removed pending rewards in `PLUME_NATIVE`. If a user has pending rewards in other tokens and those are removed, the function fails to account for them. As a result, it may incorrectly clear the user's `hasPendingRewards` flag, even though rewards from other tokens may still be pending.

Recommendation

Create a historical array of all reward tokens that ever existed in the protocol. Use this list within `clearPendingRewardsFlagIfEmpty` to ensure that the function accurately checks all potential reward tokens, even removed ones, before clearing the pending flag.

IMM-LOW-09

Admin May Improperly Clear Valid `userValidatorCooldowns` #73

Id	IMM-LOW-09
Severity	LOW
Category	Bug
Status	Fixed in 7a2fde4ace99430bb2bcb9939218ec415f0e55d2

Description

Code Affected:

- `ManagementFacet.sol::adminClearValidatorRecord`
- `ManagementFacet.sol::adminBatchClearValidatorRecords`

Both admin functions clear entries in `userValidatorCooldowns` without verifying their validity. This can lead to the removal of legitimate cooldown requests that users are still entitled to withdraw after the cooldown period.

Recommendation

Only remove cooldown entries if they are invalid (if they were created during the voting period for a slashing event). This ensures legitimate user cooldowns remain intact and prevents unintended loss of withdrawal rights.

IMM-INSIGHT-01

Redundant Winner Request State Management #68

Id	IMM-INSIGHT-01
Severity	INSIGHT
Category	Informational
Status	Fixed in 10d3f3888735bde0f913ab8eaaf1e02bba1925ae

Description

Code Affected: `Raffle.sol::requestWinner`, `Raffle.sol::handleWinnerSelection`

`requestWinner` can be called again after `handleWinnerSelection` has been called because `isWinnerRequestPending` was set to false during `handleWinnerSelection`. It has no actual effect because `handleWinnerSelection` will revert due to the prize being inactive.

Recommendation

Prevent calling `requestWinner` if a winner has already been selected

IMM-INSIGHT-02

Unused Code in Spin Contract #67

Id	IMM-INSIGHT-02
Severity	INSIGHT
Category	Informational
Status	Fixed in 6b203943f77d1f7fb43da2bf5008daf7f65106df

Description

Code Affected: `Spin.sol::_getWeeklyJackpotDetails`

The `_getWeeklyJackpotDetails` function is defined but never used in the contract.

Recommendation

Remove the unused `_getWeeklyJackpotDetails` function and any other unused code

IMM-INSIGHT-03

Unnecessary Storage Reads #64

Id	IMM-INSIGHT-03
Severity	INSIGHT
Category	Gas Optimization
Status	Fixed in 155942c25f9de53b2d34b64ab515afd055b4dbe3

Description

Code Affected: Multiple functions including `_processMaturedCooldowns`

In functions like `_processMaturedCooldowns`, storage variables are copied into storage references (e.g., `CooldownEntry` storage `cooldownEntry = ...`) even when only reading. Use memory variables instead where writes aren't needed to save gas.

Recommendation

Replace storage references with memory variables when only reading

IMM-INSIGHT-04

Duplicate Storage Variables #63

Id	IMM-INSIGHT-04
Severity	INSIGHT
Category	Gas Optimization
Status	Fixed in 12fb6349e9966cf1dd3ac678945a054bba458577

Description

Code Affected: Multiple storage variables across facets

Variables like `validatorTotalStaked` and `validators[validatorId].delegatedAmount` store the same value and are used the same `way. Removing` redundancy can improve gas efficiency.

Recommendation

Remove redundant storage variables and use a single source of truth.

IMM-INSIGHT-05

Different Reward Rates Used for The Same Segment #62

Id	IMM-INSIGHT-05
Severity	INSIGHT
Category	Informational
Status	Acknowledged

Description

Code Affected: `PlumeRewardLogic.sol::updateRewardPerTokenForValidator`

During `updateRewardPerTokenForValidator` the timestamp used for `getEffectiveRewardRateAt` is the end of the time segment, while `getEffectiveCommissionRateAt` uses the start of the time segment. Theoretically, this can lead to different reward rates for the same time segment, leading to inconsistent reward distributions.

But, since `updateRewardPerTokenForValidator` is called before every reward checkpoint, this is not a real issue.

Recommendation

Make sure to keep this fact in mind in the future. Currently, this is not a real issue.

IMM-INSIGHT-06

High Validator Count May Be Costly #60

Id	IMM-INSIGHT-06
Severity	INSIGHT
Category	Gas Optimization
Status	Acknowledged

Description

Code Affected: Multiple functions that iterate over validators

Since the system is expected to have only 10–20 validators it is not a real problem, but several functions iterate over all validators, increasing gas costs. Consider limiting how many validators a single user can stake with (e.g., max 10) and management functions that iterate over all validators can iterate in batches.

Recommendation

- Implement a maximum limit on the number of validators a user can stake with
- Add batch processing for functions that iterate over all validators

IMM-INSIGHT-07

`maxValidatorPercentage` Is Ineffective #59

Id	IMM-INSIGHT-07
Severity	INSIGHT
Category	Informational
Status	Acknowledged

Description

Code Affected: `StakingFacet.sol` validatorPercentage checks

Currently always set to zero and immutable, making `validatorPercentage` checks non-functional.

Note: If this value is set to anything other than 0 and no one has staked yet, no one will be able to stake, since for the first staker, `newDelegatedAmount == totalStaked`.

Recommendation

- Implement proper initialization of `maxValidatorPercentage`
- Add special handling for the first staker case
- Consider making `maxValidatorPercentage` configurable rather than immutable

IMM-INSIGHT-08

Missing Constant Initialization #58

Id	IMM-INSIGHT-08
Severity	INSIGHT
Category	Informational
Status	Fixed in 27b9cd45a1e25c813bb3f444cb658021fca62124

Description

Code Affected: `PlumeStaking.sol::initializePlume`

Critical protocol constants such as `maxSlashVoteDurationInSeconds` and `maxAllowedValidatorCommission` are not initialized in `initializePlume`. If `maxAllowedValidatorCommission` remains zero, validators cannot earn commission.

Recommendation

Initialize all critical protocol constants in `initializePlume`

IMM-INSIGHT-09

Single Validator Can Be Slashed Without Any Votes #56

Id	IMM-INSIGHT-09
Severity	INSIGHT
Category	Informational
Status	Acknowledged

Description

Code Affected: `ValidatorFacet.sol::slashValidator`

If only one validator is active, it can be slashed without any voting. There shouldn't be a single validator so it's not a huge problem.

Recommendation

N/A

IMM-INSIGHT-10

`userLastCheckpointIndex` Becomes Inaccurate After Checkpoint Pruning #74

Id	IMM-INSIGHT-10
Severity	INSIGHT
Category	Informational
Status	Fixed in 152e497926bba305662c0128a9ad44002ecf8593

Description

Code Affected: `ManagementFacet.sol::pruneRewardRateCheckpoints`

The `pruneRewardRateCheckpoints` function removes validator reward checkpoints by shifting the checkpoint array and popping elements. However, if a user's `userLastCheckpointIndex` was pointing to one of the pruned entries (for example the latest checkpoint), it will become invalid and refer to a non-existent index.

Since `userLastCheckpointIndex` is only used in view contexts and doesn't affect state changes or protocol logic, this issue poses no security risk.

Recommendation

Consider whether this variable is still necessary. If it's not providing meaningful utility, it might be worth removing.

IMM-INSIGHT-11

The view function `getSlashVoteCount` May Count Votes from Ineligible Validators #75

Id	IMM-INSIGHT-11
Severity	INSIGHT
Category	Informational
Status	Fixed in aebcbd127a932bed3bb6c3e6dfc4cd0df7de5902

Description

Code Affected: `ValidatorFacet.sol::getSlashVoteCount`

The `getSlashVoteCount` function does not verify whether the vote was cast by an eligible validator (i.e., active and not slashed). As a result, votes from inactive or already-slashed validators may be incorrectly counted. Additionally, there is inconsistency in how vote expiration is handled:

- `_cleanupExpiredVotes` treats votes expiring at the current block as expired:

TypeScript
```
bool voteHasExpired = block.timestamp >= voteExpiration;
```

- `getSlashVoteCount`, on the other hand, treats them as still valid:

TypeScript
```
if (voteExpiration > 0 && voteExpiration >= currentTime) {
    validVoteCount++;
}
```

Recommendation

Ensure `validVoteCount` is incremented only when:

1. The voting validator is active and not slashed.

2. The vote has not expired (`voteExpiration > block.timestamp`).

Align the expiration logic with `_cleanupExpiredVotes` to maintain consistency across the codebase.

IMM-INSIGHT-12

Off-by-One Error Prevents Re-Voting for a Single Block #76

Id	IMM-INSIGHT-12
Severity	INSIGHT
Category	Informational
Status	Fixed in 0804ea4ba73a22c608d00d638369392578c6eeb3

Description

Code Affected: `ValidatorFacet.sol::voteToSlashValidator`

In `voteToSlashValidator`, a validator is prevented from voting again if their previous vote has not yet expired:

TypeScript

```
       uint256 currentVoteExpiration =
$.slashingVotes[maliciousValidatorId][voterValidatorId];
        if (currentVoteExpiration >= block.timestamp) {
            revert AlreadyVotedToSlash(maliciousValidatorId, voterValidatorId);
        }
```

This logic treats a vote as valid if its expiration is greater than or equal to the current block timestamp. However, `_cleanupExpiredVotes` uses the following logic to determine vote validity:

TypeScript

```
bool voteHasExpired = block.timestamp >= voteExpiration;
```

This means votes that expire in the current block (`voteExpiration == block.timestamp`) are treated as expired in `_cleanupExpiredVotes`, but still prevent re-voting in `voteToSlashValidator`.

Impact

This creates a one-block window where a validator cannot re-vote even though their vote is considered expired for cleanup and counting purposes. It's a minor logic inconsistency that might lead to confusion or temporary vote blocking.

Recommendation

Update the voting logic in `voteToSlashValidator` to match the expiration logic used in `_cleanupExpiredVotes`. Specifically, change the condition to:

TypeScript

```typescript
if (currentVoteExpiration > block.timestamp) {
    revert AlreadyVotedToSlash(maliciousValidatorId, voterValidatorId);
}
```

This ensures that votes expiring in the current block are consistently treated as expired across all code paths.