IMMUNEBYTES

Security Assessment

Smart Contract Audit

plume

03rd Feb 2025

Plume Network
Staking | Airdrop
v1.0

Introduction

Name	Plume Network
Website	https://plumenetwork.xyz/
Repository/Source	https://github.com/CliqueOfficial/plume-contracts
Platform	L1
Network	Ethereum and EVM-compatible chains
Language	Solidity
Initial Commit	54013264e7755202c02f58884d7cf8e5218c5d16
Final Commit	NA
Timeline	09th Jan 2025 - 13th Jan 2025

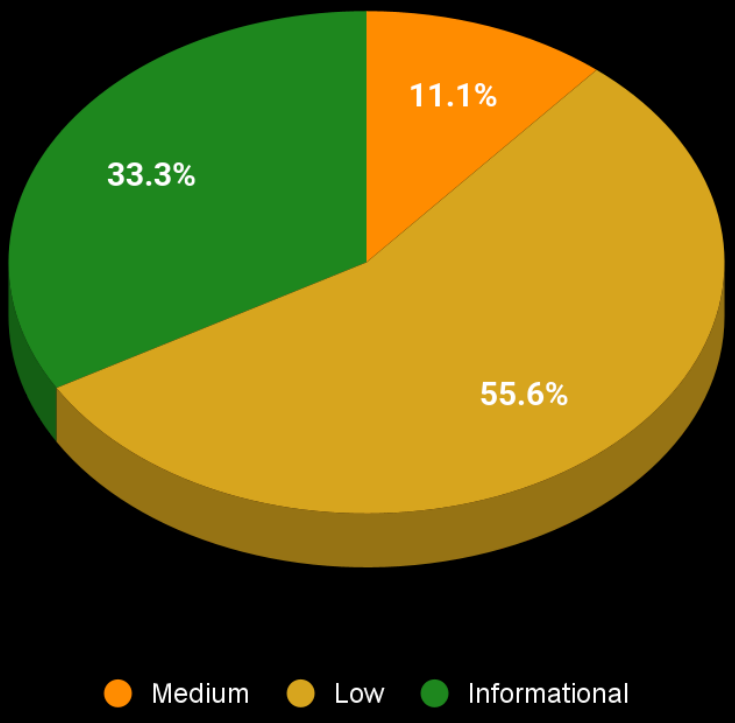
Table of Content

Executive Summary

This initial audit report provides a detailed analysis of the findings from the security assessment of the Plume Network's airdrop and staking system. The audit was conducted between January 09, 2025, and January 13, 2025. This report documents the results and observations from the final day of the review.

The Plume Network's smart contract suite implements an airdrop mechanism (via a Merkle tree) and an ERC20 staking protocol, designed to distribute tokens securely while incentivizing long-term staking with an unlocking-delay functionality. The audit's primary focus was on the security, correctness, and robustness of the system's functionalities, including the staking process, airdrop claim validations, and administrative controls.

Issues Overview



Project Overview

Key Components

1. **Airdrop Distribution**
    - **Merkle Tree Validation:** Ensures eligible users can claim their airdrop using Merkle proofs.
    - **Signature Verification:** Adds an additional layer of authentication to prevent fraudulent claims.
    - **Stake Integration:** Automatically stakes a portion of the claimed tokens to align incentives with network goals.
    - **Admin Controls:** Allows setting the Merkle root and signing authority for claim validation.
2. **Staking Protocol**
    - **Staking and Unstaking:** Users can stake tokens, earn rewards, and withdraw funds after a lockup period.
    - **Locking Period:** Implements delayed unstaking with configurable locking parameters for flexibility.
    - **Permissioned Staking:** Supports optional permissioning for controlled staking operations.
    - **State Validation:** Tracks staked balances and user-specific details to ensure data integrity.
3. **Admin Functions**
    - **Configurable Parameters:** Admins can set staking limits, unlocking parameters, and airdrop configurations.
    - **Pausability:** The system includes emergency pause functionality to mitigate risks during incidents.
    - **Access Control:** Enforces role-based access control for all sensitive operations.

Security Checklist

During the comprehensive audit of the Plume Network, the following potential vulnerabilities were assessed:

Access Control Issues	✅
Merkle Proof Manipulation	✅
Signature Spoofing or Replay Attacks	✅
Unintended State Manipulation	✅
Reentrancy Attacks	✅
Integer Overflow and Underflow	✅
Denial of Service (DoS) Scenarios	✅
Logic and Calculation Errors	✅
Event Emission for State Changes	✅
ERC20 Compliance	✅

Methodology

ImmuneBytes team has performed thorough testing of the project, starting with analyzing the code design patterns where the smart contracts and provided technical documentation were reviewed to ensure the architecture of the codebase aligns with the business specifications from a bird's eye view. During this phase, the invariants were identified along with the execution of the pre-existing test suite to ensure the smooth working of the implemented functionality and set up for the auditor's edge case testing in upcoming phases

Our team then performed a formal line-by-line inspection of the Smart Contract to find potential issues like Signature Replay Attacks, Unchecked External Calls, Merkle Tree Exploits, Reentrancy, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the next phase, the team ran curated test cases to verify the findings and issues identified during previous phases to provide a proof of concept. The team also runs automated tests over the provided code base using external and in-house-developed tools to identify vulnerabilities and security flaws of a static nature.

The code was audited by a team of independent auditors, which included -

- Testing the functionality of the Smart Contracts to determine proper logic has been followed throughout.
- Thorough, line-by-line review of the code, done by a team of security researchers with experience across Blockchain security, cybersecurity, software engineering, game theory, economics and finance.
- Deploying the code on localnet using open-source testing frameworks and clients to run live tests.
- Analyzing failing transactions to check whether the Smart Contracts implementation handles bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest secure version.
- Analyzing the security of the on-chain data and components.

Scope

In-Scope Files

The following files and their respective locations were included in the scope of the audit:

1. **Staking Contract**:
   Location: /contracts/Staking.sol
   This contract implements the ERC20 staking mechanism with delayed unstaking, permissioned staking, and reward tracking.

2. **Distributor Contract**:
   Location: /contracts/Distributor.sol
   This contract handles airdrop distribution, integrating Merkle tree validation and staking functionality.

Critical Area of Focus

1. **Airdrop Distribution**
    - Validation of Merkle proofs for eligible claims.
    - Ensuring the signer mechanism securely authenticates claims.
    - Proper handling of staking percentages and unclaimed tokens.
2. **Staking Mechanism**
    - Validation of locking and unlocking parameters.
    - Security of permissioned staking logic.
    - Ensuring correct accounting of staked and unstaked balances.
3. **Admin Controls**
    - Validation of access control for admin functions.
    - Ensuring safe updates to critical parameters such as Merkle root, signer, and staking configurations.
4. **Reentrancy and Security**
    - Verifying all state-changing functions are protected against reentrancy attacks.
    - Ensuring correct usage of nonReentrant and safe external call patterns.
5. **ERC20 Compliance**
    - Compatibility with standard ERC20 tokens, ensuring edge cases such as fee-on-transfer tokens are handled correctly.

Out of Scope

The following areas were excluded from the scope of this audit:

- **Frontend Integrations:** The audit excluded any user interface or frontend applications interacting with the smart contracts.
- **Third-Party Libraries:** External libraries used for Merkle proofs, ECDSA operations, or other utilities were not reviewed beyond their integration with the contracts.
- **Off-chain Components:** The audit did not cover review of the off-chain components or data integrated within the smart contracts listed in the scope.

Security Concerns

1. **Over-Reliance on a Centralized Admin Entity**

   The Plume Network contracts currently rely heavily on a centralized admin for critical operations, such as setting the signer, claimRoot, and pausing or unpausing the system. This dependency creates a single point of failure, as the misuse or compromise of the admin's private key could lead to unauthorized modifications or operational disruptions. A decentralized approach, such as using a multisig wallet requiring approvals from multiple parties, can significantly reduce the risk of malicious actions and enhance trust in the system.

2. **Avoid Manual Deployment Setup**

   The deployment process involves manually configuring critical parameters like signer, claimRoot, and stakingToken, which introduces risks of human error and misconfiguration. An automated deployment script, thoroughly validated in test environments before use, ensures consistency, reduces the chances of errors, and improves the overall reliability of the deployment process while maintaining auditability of all setup steps.

Security Review Report

Severity	Open	Acknowledged	Partially Resolved	Resolved	**TOTAL**
Critical	-	-	-	-	-
High	-	-	-	-	-
Medium	-	1	-	-	1
Low	-	5	-	-	5
Informational	-	3	-	-	3
TOTAL	-	9	-	-	9

Severity Reference

1. **Critical**
   Issues causing protocol insolvency, unauthorized theft, or irreversible loss or alteration of assets, or governance outcomes.

2. **High**
   Issues involving theft or freezing of unclaimed yield, royalties, or temporary locking of assets.

3. **Medium**
   Operational failures, inefficiencies, or exploitations causing delays, excessive gas usage, or disruption without direct loss.

4. **Low**
   Minor deviations in promised functionality without impact on underlying value.

5. **Informational**
   Recommendations for code readability, maintainability, or best practices.

Status Reference

1. **Open**
   The issue has been identified and is pending resolution.

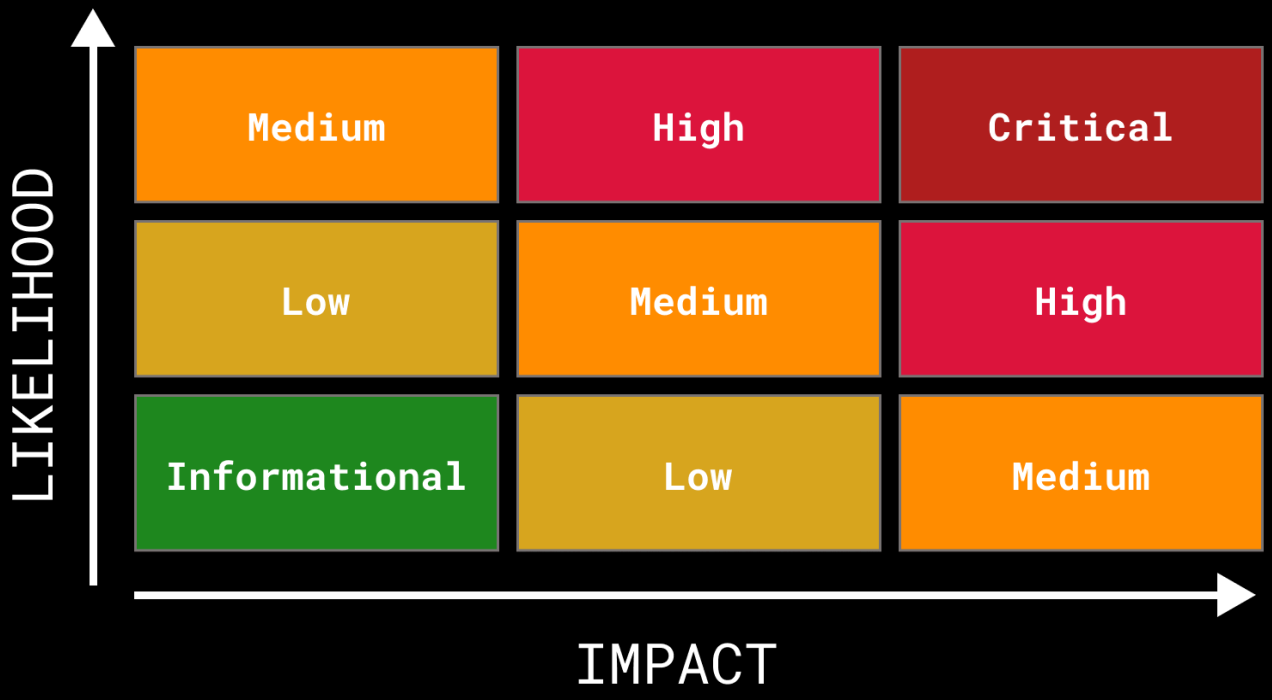
2. **Acknowledged**
   The issue has been reviewed and accepted by the team but remains unresolved.

3. **Partially Resolved**
   The issue has been mitigated to some extent, but residual risks or concerns persist.

4. **Resolved**
   The issue has been fully addressed, with no further action required.

5. **Closed**
   The issue is no longer relevant or actionable, regardless of resolution.

Risk Matrix


LIKELIHOOD
Medium
High
Critical
Low
Medium
High
Informational
Low
Medium
IMPACT

Summary of Findings

CODE	ISSUE	SEVERITY	STATUS
PNSA01	Temporary contract halting due to invalidated setUnlockParams	Medium	ACK
PNSA02	Signatures bypassed if signer is set to zero address	Low	ACK
PNSA03	Missing input validation for zero values in constructor	Low	ACK
PNSA04	Zero address check missing in setPermittedStaker	Low	ACK
PNSA05	Missing input validation for zero values in Distributor contract methods	Low	ACK
PNSA06	Missing input validation for zero values in withdrawTokens	Low	ACK
PNSA07	Misleading error message in _signatureCheck	Info	ACK
PNSA08	Constants and immutables not following naming conventions	Info	ACK
PNSA09	Lack of event emission for some critical functions	Info	ACK

Findings Overview

Severity: Medium	Impact: High	Likelihood: Low
PNSA01	**Temporary contract halting due to invalidated setUnlockParams**	
File Name	contracts/Staking.sol	
Status	Acknowledged	

Description

The setUnlockParams function does not enforce sanity checks on _unstakingStartBlock and _lockingPeriodBlocks. Also, it does not ensure that the parameters are set to times equal to or in the future. If set to unrealistic values either during the start of the staking campaign or in between a running campaign, the results can be as severe as functionality halting leading to reputation loss for the protocol.

Additionally, the protocol's documentation does not define any locking period campaigns according to the business charter which reduces consistency from user perspective. User documentation is an essential part of a smart contract specification along the lines of which the implementation itself is validated.

Impact

The admin can set parameters to invalid or past values, rendering unstaking impossible or breaking the contract's logic.

Proposed Solution

Add checks to validate that _unstakingStartBlock is in the future and _lockingPeriodBlocks is non-zero and enforces the locking period specification as per the user documentation defined by the protocol.

Developer's Response

Acknowledged. Unlock params are optional params. Users shall unlock instantly if it's set to zero.

Severity: Low	Impact: Medium	Likelihood: Low
PNSA02	**Signatures bypassed if signer is set to zero address**	
File Name	contracts/Distributor.sol	
Status	Acknowledged	

Description

The _signatureCheck method does not validate that the signer address is non-zero. If the signer is set to the zero address, any signature can be bypassed since ECDSA.recoverCalldata returns the zero address for all malleable signatures. This results in a complete bypass of signature-based authentication.

Impact

An attacker could forge claims and maliciously withdraw tokens without any valid authentication. This could result in a total loss of tokens within the Distributor contract.

Code Affected

```
Line of Code: Distributor.sol
if (recoveredSigner != signer) revert InvalidSignature();
```

Proposed Solution

Add a check to ensure the signer is a valid non-zero address in the _signatureCheck function and during the setSigner method.

Developer's Response

Acknowledged.
Signer is optial configuration. Owner should make sure all params are as expected before unpausing it.

Severity: Low	Impact: Medium	Likelihood: Low
PNSA03	**Missing input validation for zero values in constructor**	
File Name	contracts/Staking.sol, contracts/Distributor.sol	
Status	Acknowledged	

Description

The constructors of both ERC20Staking and Distributor contracts do not validate the input parameters _stakingToken, _minStakingAmount, _signer, and _staking. This could result in unexpected contract behavior if these parameters are set to zero addresses or invalid values.

Impact

- A zero address for _stakingToken would result in broken token transfers.
- A zero _minStakingAmount allows users to stake trivial amounts.
- A zero _signer in Distributor would allow signature bypass.

Code Affected

```
  Lines of Code:
  ERC20Staking.sol:
  stakingToken = _stakingToken;
  minStakingAmount = _minStakingAmount;
 Distributor.sol:
  signer = _signer;
  staking = _staking;
```

Proposed Solution

Validate all constructor inputs to ensure they are non-zero and within acceptable ranges.

Developer's Response

Acknowledged. Owner is responsible for this.

Severity: Low	Impact: Medium	Likelihood: Low
PNSA04	**Zero address check missing in setPermittedStaker**	
File Name	contracts/Staking.sol	
Status	Acknowledged	

Description

The setPermittedStaker function does not validate that the _permittedStaker address is non-zero.

Impact

Setting the permittedStaker to a zero address could break permissioned staking logic, allowing unintended accounts to stake.

Proposed Solution

Add a check to ensure _permittedStaker is a non-zero address.

Developer's Response

Acknowledged. Intended

Severity: Low	Impact: Medium	Likelihood: Low
PNSA05	**Missing input validation for zero values in Distributor contract methods**	
File Name	contracts/Distributor.sol	
Status	Acknowledged	

Description

The setClaimRoot and setSigner methods do not validate the incoming inputs _claimRoot and _signer being a valid non-zero value.

Impact

Setting a zero claimRoot or _signer would render all claims and unlocks invalid, halting airdrop functionality.

Proposed Solution

Add a check to ensure _claimRoot is non-zero.

Developer's Response

Acknowledged. Intended

Severity: Low	Impact: Medium	Likelihood: Low
PNSA06	**Missing input validation for zero values in withdrawTokens**	
File Name	contracts/Distributor.sol	
Status	Acknowledged	

Description

The withdrawTokens function does not validate the receiver address, allowing it to be set to the zero address. Token withdrawals to the zero address result in irreversible fund loss for the protocol revenue.

Code Affected

```
Line: Distributor.sol
IERC20(token).safeTransfer(receiver, amount);
```

Proposed Solution

Add a check to ensure receiver is non-zero.

Developer's Response

Acknowledged. Intended

Severity: Info	
PNSA07	**Misleading error message in _signatureCheck**
File Name	contracts/Distributor.sol
Status	Acknowledged

Description

The error message in _signatureCheck uses InvalidSignature even when the issue is an incorrect signer. Misleading error messages make debugging and troubleshooting more difficult.

Code Affected

```
Line: Distributor.sol
if (recoveredSigner != signer) revert InvalidSignature();
```

Proposed Solution

Use a separate error message like InvalidSigner when the recovered address does not match the expected signer.

Developer's Response

Acknowledged.

Severity: Info	
PNSA08	**Constants and immutables not following naming conventions**
File Name	contracts/Staking.sol
Status	Acknowledged

Description

Constants and immutables like minStakingAmount do not follow the recommended naming convention (e.g., uppercase with underscores).

Impact

Non-standard naming reduces readability and consistency.

Code Affected

```
Line: ERC20Staking.sol
 uint256 public immutable minStakingAmount;
```

Proposed Solution

Rename to MIN_STAKING_AMOUNT.

Developer's Response

Acknowledged.

Severity: Info	
PNSA09	**Lack of event emission for some critical functions**
File Name	contracts/Staking.sol
Status	Acknowledged

Description

Functions like setUnlockParams do not emit events, reducing traceability.

Impact

Lack of event emission makes it difficult for off-chain services to monitor and verify contract state changes, impacting transparency and trust.

Code Affected

```
Line: ERC20Staking.sol
function setUnlockParams(uint256 _unstakingStartBlock, uint256
_lockingPeriodBlocks) external onlyOwner
```

Proposed Solution

Emit an event such as UnlockParamsUpdated with the new values whenever setUnlockParams is called.

Developer's Response

Acknowledged.

Concluding Remarks

During the audit of Plume Network, our auditors identified issues within the system categorized as medium and low severity along with a few recommendations for code readability. The development team swiftly responded by acknowledging the security team's concerns and the implications of the reported findings. The ImmuneBytes' security team also acknowledged the client's business needs for all the issues identified and reported for the protocol under the audit scope.

Disclaimer

ImmuneBytes' security audit services are designed to help identify and mitigate potential security risks in smart contracts and related systems. However, it is essential to recognize that no security audit can guarantee absolute protection against all possible security threats. The audit findings and recommendations are based on the information provided during the assessment.

Furthermore, the security landscape is dynamic and ever-evolving, with new vulnerabilities and threats emerging regularly. As a result, it is strongly advised to conduct multiple audits and establish robust bug bounty programs as part of a comprehensive and continuous security strategy. A single audit alone cannot protect against all potential risks, making ongoing vigilance and proactive risk management essential.

To ensure maximum security, the project must implement the recommendations provided in the audit report and regularly monitor its smart contracts for vulnerabilities. It is imperative to adopt appropriate risk management strategies and remain proactive in addressing potential threats as they arise.

Please note that auditors cannot be held liable for any security breaches, losses, or damages that may occur after the audit has been completed despite using audit services. The responsibility for implementing the recommendations and maintaining the ongoing security of the systems rests solely with the project.