Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.10.23, the SlowMist security team received the Plume Network team's security audit application for Plume Network Staking Phase2, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

This is the staking contract component of Plume Network. The audit mainly covers the ReserveStaking contract, while the SBTC.sol and STONE.sol files are primarily used for testing purposes. Users can stake SBTC and STONE tokens in the ReserveStaking contract to earn rewards. This is a pre-staking contract. After the main staking contract is deployed (possibly on another chain), the admin role of the pre-staking contract will migrate users' staked tokens to the main staking contract in a centralized manner.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Risks of excessive privilege	Authority Control Vulnerability Audit	Medium	Acknowledged
N2	Native tokens that are mistakenly transferred to the proxy contract will be locked	Design Logic Audit	Suggestion	Fixed
N3	Actual amount not used in calculation	Design Logic Audit	Suggestion	Fixed
N4	Incompatible with tokens that have transfer hooks	Design Logic Audit	Information	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/plumenetwork/contracts

commit: 70e2c223a57b5ba421b21b8bfee5bfd627626a15

Fixed Version:

https://github.com/plumenetwork/contracts

commit: 65a08362968b2bcf669bfbb8efd8d1eca51e5aa2

Audit Scope:

- staking/src/ReserveStaking.sol
- staking/src/SBTC.sol
- staking/src/STONE.sol
- staking/src/proxy/PlumePreReserveFund.sol
- staking/src/proxy/PlumePreStaking.sol

The main network address of the contract is as follows:

Contract Name	Contract Address	Chain
PlumePreReserveFund (Proxy)	0xBa0Ae7069f94643853Fce3B8Af7f55AcBC11e397	Ethereum
ReserveStaking (Impl)	0x41dfAEe3F2F39bE85eddaB3e0fB43338F2403687	Ethereum
PlumePreStaking (Proxy)	0xdbd03D676e1cf3c3b656972F88eD21784372AcAB	Ethereum
RWAStaking (Impl)	0x2eA61aA006B79606baFc0aE6870D7FFb22411289	Ethereum

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

ReserveStaking			
Function Name	Visibility	Mutability	Modifiers
_getReserveStakingStorage	Private	-	-
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	initializer
_authorizeUpgrade	Internal	Can Modify State	onlyRole
adminWithdraw	External	Can Modify State	onlyRole
stake	External	Can Modify State	-
withdraw	External	Can Modify State	-
getSBTC	External	-	-
getSTONE	External	-	-
getSBTCTotalAmountStaked	External	-	-
getSTONETotalAmountStaked	External	-	-
getUsers	External	-	-
getUserState	External	-	-

ReserveStaking			
getEndTime	External	-	-

PlumePreReserveFund			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC1967Proxy
<Receive Ether>	External	Payable	-

PlumePreStaking			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC1967Proxy
<Receive Ether>	External	Payable	-

SBTC			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20 Ownable
mint	Public	Can Modify State	onlyOwner

STONE			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20 Ownable
mint	Public	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [Medium] Risks of excessive privilege

Category: Authority Control Vulnerability Audit

Content

Similar to the RWA Staking contract, users can stake STONE and SBTC tokens through the ReserveStaking contract to earn rewards. However, after the main staking contract is deployed (possibly on another chain), the admin role will use the adminWithdraw function to withdraw all tokens from the ReserveStaking contract and migrate users' staked tokens to the main staking contract in a centralized manner. This poses the risk of excessive privileges.

Code location: staking/src/ReserveStaking.sol#L164

```
function adminWithdraw() external onlyRole(ADMIN_ROLE) {
    ReserveStakingStorage storage $ = _getReserveStakingStorage();
    if ($.endTime != 0) {
        revert StakingEnded();
    }

    uint256 sbtcAmount = $.sbtc.balanceOf(address(this));
    uint256 stoneAmount = $.stone.balanceOf(address(this));

    $.sbtc.safeTransfer(msg.sender, sbtcAmount);
    $.stone.safeTransfer(msg.sender, stoneAmount);
    $.endTime = block.timestamp;

    emit AdminWithdrawn(msg.sender, sbtcAmount, stoneAmount);
}
```

Solution

It is recommended to use a multi-signature wallet to manage the admin role to avoid single-point-of-failure risk, but this cannot mitigate the risk of excessive privileges. In the long run, management through community governance can effectively alleviate the risk of excessive privileges and increase user trust.

Status

Acknowledged; The project team has transferred the withdrawal permissions of the ReserveStaking contract to the Timelock contract (0x0f5f5515b768C14b55934924ED322c43773308F9), with a multisig wallet (0xa472f6bDf1E676C7B773591d5D820aDC27a2D51c) managing the execution rights of the Timelock contract.

[N2] [Suggestion] Native tokens that are mistakenly transferred to the proxy contract will be locked

Category: Design Logic Audit

Content

The PlumePreStaking and PlumePreReserveFund contracts include a receive function, which allows the contracts to receive native tokens. Unfortunately, there is no interface within the contracts to withdraw native tokens. If users mistakenly transfer native tokens into the contracts, the tokens will be locked. Although the contracts are upgradeable, this will cause inconvenience.

Code location:

staking/src/proxy/PlumePreReserveFund.sol#L19

staking/src/proxy/PlumePreStaking.sol#L19

```
receive() external payable { }
```

Solution

It is recommended that the receive function be removed to prevent users from accidentally transferring native tokens into the contracts.

Status

Fixed

[N3] [Suggestion] Actual amount not used in calculation

Category: Design Logic Audit

Content

In the ReserveStaking contract, users can withdraw their staked tokens using the withdraw function. The contract first settles the rewards for the user's staked tokens and then deducts a certain amount of rewards based on the number of tokens the user withdraws.

When calculating the deducted amount, the user's input withdrawal amount (sbtcAmount/stoneAmount) is used for the calculation. However, the actual deduction of the user's staked amount is based on the difference in the contract's balance before and after the transfer (actualSbtcAmount/actualStoneAmount). This means that the parameters used for deduction are not the same as those used for calculating the reward amount.

In fact, STONE and SBTC tokens are not fee-on-transfer type tokens, so these two parameters are actually the same.

Therefore, using the balance difference before and after the transfer is not necessary.

Code location: staking/src/ReserveStaking.sol#L234-L278

```
    function withdraw(uint256 sbtcAmount, uint256 stoneAmount) external {
        ...
        if (sbtcAmount > 0) {
            IERC20 sbtc = $.sbtc;
            userState.sbtcAmountSeconds += userState.sbtcAmountStaked * (timestamp -
userState.sbtcLastUpdate);
            uint256 previousBalance = sbtc.balanceOf(address(this));
            sbtc.safeTransfer(msg.sender, sbtcAmount);
            uint256 newBalance = sbtc.balanceOf(address(this));
            actualSbtcAmount = previousBalance - newBalance;
            userState.sbtcAmountSeconds -= userState.sbtcAmountSeconds * sbtcAmount /
userState.sbtcAmountStaked;
            userState.sbtcAmountStaked -= actualSbtcAmount;
            userState.sbtcLastUpdate = timestamp;
            $.sbtcTotalAmountStaked -= actualSbtcAmount;
        }

        if (stoneAmount > 0) {
            IERC20 stone = $.stone;
            userState.stoneAmountSeconds += userState.stoneAmountStaked * (timestamp -
userState.stoneLastUpdate);
            uint256 previousBalance = stone.balanceOf(address(this));
            stone.safeTransfer(msg.sender, stoneAmount);
            uint256 newBalance = stone.balanceOf(address(this));
            actualStoneAmount = previousBalance - newBalance;
            userState.stoneAmountSeconds -= userState.stoneAmountSeconds * stoneAmount
/ userState.stoneAmountStaked;
            userState.stoneAmountStaked -= actualStoneAmount;
            userState.stoneLastUpdate = timestamp;
            $.stoneTotalAmountStaked -= actualStoneAmount;
        }

        emit Withdrawn(msg.sender, actualSbtcAmount, actualStoneAmount);
    }
```

Solution

It is recommended to use the actualStoneAmount/actualSbtcAmount parameters to calculate the stoneAmountSeconds/sbtcAmountSeconds amount to be deducted. Alternatively, in the withdraw operation, instead

of using the contract's balance difference before and after the transfer for calculations, use the user's input amount for all calculations.

Status

Fixed

[N4] [Information] Incompatible with tokens that have transfer hooks

Category: Design Logic Audit

Content

In the ReserveStaking contract, users can withdraw their staked tokens using the withdraw function. The contract first transfers the tokens to the user via the safeTransfer function before updating the user's userState. This makes it incompatible with ERC223/ERC777-like tokens. When the token transfer has a hook function, it will lead to the withdraw function being reentered. Since the user's LastUpdate state has not been updated at this point, it will allow the user to obtain additional rewards. The currently deployed STONE and SBTC tokens on the mainnet do not have transfer hook functionality, but when deploying the contract to other chains in the future, special attention should be paid to such risks.

Code location: staking/src/ReserveStaking.sol#L251-L275

```
    function withdraw(uint256 sbtcAmount, uint256 stoneAmount) external {
        ...
        if (sbtcAmount > 0) {
            IERC20 sbtc = $.sbtc;
            userState.sbtcAmountSeconds += userState.sbtcAmountStaked * (timestamp -
userState.sbtcLastUpdate);
            uint256 previousBalance = sbtc.balanceOf(address(this));
            sbtc.safeTransfer(msg.sender, sbtcAmount);
            uint256 newBalance = sbtc.balanceOf(address(this));
            actualSbtcAmount = previousBalance - newBalance;
            userState.sbtcAmountSeconds -= userState.sbtcAmountSeconds * sbtcAmount /
userState.sbtcAmountStaked;
            userState.sbtcAmountStaked -= actualSbtcAmount;
            userState.sbtcLastUpdate = timestamp;
            $.sbtcTotalAmountStaked -= actualSbtcAmount;
        }

        if (stoneAmount > 0) {
            IERC20 stone = $.stone;
            userState.stoneAmountSeconds += userState.stoneAmountStaked * (timestamp -
```

```
userState.stoneLastUpdate);
            uint256 previousBalance = stone.balanceOf(address(this));
            stone.safeTransfer(msg.sender, stoneAmount);
            uint256 newBalance = stone.balanceOf(address(this));
            actualStoneAmount = previousBalance - newBalance;
            userState.stoneAmountSeconds -= userState.stoneAmountSeconds * stoneAmount
/ userState.stoneAmountStaked;
            userState.stoneAmountStaked -= actualStoneAmount;
            userState.stoneLastUpdate = timestamp;
            $.stoneTotalAmountStaked -= actualStoneAmount;
        }

        emit Withdrawn(msg.sender, actualSbtcAmount, actualStoneAmount);
    }
```

Solution

If the tokens supported by the contract have transfer hook functionality, the user's userState should be updated before performing the transfer, or a reentrancy lock should be used.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002410240001	SlowMist Security Team	2024.10.23 - 2024.10.24	Low Risk

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 medium risk, 2 suggestions, and 1 information. All the findings were fixed or acknowledged. The code was not deployed to the mainnet. Currently, the withdrawal permissions in both RWAStaking and ReserveStaking contracts are managed by the timelock contract, with multisig contract controlling the execution rights of the timelock. Therefore, the overall contract risk is low.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist