Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.09.17, the SlowMist security team received the Binomial team's security audit application for Binomial SimpleStaking, 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 a real attack.

The test method information:

Test method	Description
Black box testing	Conduct an external security test from an attacker's perspective.
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 code is scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the code for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were 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 an audit of the SimpleStaking contract of the Binomial protocol. The SimpleStaking contract allows users to deposit whitelisted tokens for staking, and users can withdraw these tokens at any time. The contract also implements an address banning feature, where the owner role can ban any user to ensure they cannot perform stake/unstake operations.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Fee-on-transfer token compatibility issues	Design Logic Audit	Medium	Fixed
N2	Risks of Fake Top-ups	"False top-up" Vulnerability	Critical	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/Binomial-fi/binomialfi-contracts

commit: 899517cad7c459edf2e1663385bd07413798d33a

Fixed Version:

https://github.com/Binomial-fi/binomialfi-contracts

commit: baac4b770257846fe42c2e743e405da7bda7aa24

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

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

SimpleStaking			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Ownable
whitelistToken	External	Can Modify State	onlyOwner
removeWhitelistedToken	External	Can Modify State	onlyOwner
setBannedAddress	External	Can Modify State	onlyOwner

SimpleStaking			
stake	External	Can Modify State	nonReentrant checkBannedAddress
unstake	External	Can Modify State	nonReentrant checkBannedAddress
stakeNative	External	Payable	nonReentrant checkBannedAddress
unstakeNative	External	Can Modify State	nonReentrant checkBannedAddress

4.3 Vulnerability Summary

[N1] [Medium] Fee-on-transfer token compatibility issues

Category: Design Logic Audit

Content

In the SimpleStaking contract, users can deposit a specified amount of whitelisted tokens through the stake function. The contract uses the transferFrom interface to transfer the user's deposited amount into the contract and record this amount. However, it should be noted that if the whitelisted token is a fee-on-transfer token, the actual amount of tokens received by the contract will be less than the user's deposit amount. This will result in the contract's recorded amount being greater than the user's actual deposit amount.

Code location: src/contracts/simple-staking/SimpleStaking.sol#L67-L70

```
    function stake(address token, uint256 amount) external override nonReentrant
checkBannedAddress {
        if (!whitelistedTokens[token]) revert TokenNotWhitelisted();

        stakes[msg.sender][token] += amount;
        totalStaked[token] += amount;

        IERC20(token).transferFrom(msg.sender, address(this), amount);

        emit Stake(msg.sender, token, amount, block.timestamp);
    }
```

Solution

It is recommended to use the difference in token balance before and after the transfer as the user's actual deposit

amount for recording purposes.

Status

Fixed; The protocol will reject all fee-on-transfer tokens.

[N2] [Critical] Risks of Fake Top-ups

Category: "False top-up" Vulnerability

Content

In the stake function of the SimpleStaking contract, the contract uses the transferFrom interface to transfer the user's deposited amount into the contract, but does not check its return value. However, it should be noted that some weird ERC20 tokens will not revert when the transfer fails, but instead return false (e.g., ZRX). Failure to check the transfer return value will result in the risk of user deposit failure while the contract still successfully records the amount.

Code location: src/contracts/simple-staking/SimpleStaking.sol#L70,L86

```
    function stake(address token, uint256 amount) external override nonReentrant
checkBannedAddress {
        ...
        IERC20(token).transferFrom(msg.sender, address(this), amount);
        ...
    }

    function unstake(address token, uint256 amount) external override nonReentrant
checkBannedAddress {
        ...
        IERC20(token).transfer(msg.sender, amount);
        ...
    }
```

Solution

It is recommended to use OpenZeppelin's SafeERC20 library for token transfer/transferFrom operations.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002409170001	SlowMist Security Team	2024.09.17 - 2024.09.17	Passed

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 critical risk, and 1 medium risk vulnerability. All the findings were fixed. The code was not deployed to the mainnet.

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