SLOWMIST

Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.04.22, the SlowMist security team received the Stake Mask team's security audit application for Stake Mask, 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

The project in this audit focus on providing users with the ability to lock in deposits and bonuses. In the StakeManager contract, the user can lock the current pool with a deposit and withdraw funds once the pool status is unlocked.

In the Reward contract, users can acquire tokens in the corresponding reward pool through Merkel Proofs.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Lack of previous pool status check	Design Logic Audit	High	Fixed
N2	Difference check when changing pools	Design Logic Audit	Suggestion	Fixed
N3	Missing return value check	Others	Suggestion	Fixed
N4	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/DimensionDev/stake_mask

commit: 5ef2937b0701b90d0a0e1f5d92795832ec5f74a6

Fixed Version:

https://github.com/DimensionDev/stake_mask

commit: 0ddaacb0a95434a500000da15b75851c494cc150

The main network address of the contract is as follows:

StakeManager: https://etherscan.io/address/0x089f9e409e2ae5837def520ce6bfb2fa03ce5128

Reward: https://etherscan.io/address/0xb55f6363e8033641ada71afadabd667d071bc9b1

4.2 Visibility Description

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

Reward			
Function Name	Visibility	Mutability	Modifiers

Reward			
<Constructor>	Public	Can Modify State	Ownable
claim	Public	Can Modify State	nonReentrant
createRewardPool	Public	Can Modify State	onlyOwner
updateRewardPool	Public	Can Modify State	onlyOwner
emergencyWithdraw	Public	Can Modify State	onlyOwner

TestToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20

StakeManager			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Ownable
depositAndLock	Public	Can Modify State	nonReentrant
withdraw	Public	Can Modify State	nonReentrant
changePool	Public	Can Modify State	nonReentrant
createPool	Public	Can Modify State	onlyOwner
updatePool	Public	Can Modify State	onlyOwner
updateCurrentPoolId	Public	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [High] Lack of previous pool status check

Category: Design Logic Audit

Content

In the StakeManager contract, the user can deposit funds into the contract by calling the depositAndLock function and withdraw funds by calling the withdraw function. However, when a new pool is added and currentPoolId is updated to the id of the new pool, the user calls depositAndLock function again to make a deposit without checking the unlocked state of the pool that the user deposited in before. So the poolId in the user's information will be directly overwritten with the new currentPoolId, even if the pool state at the time of the previous deposit was a locked state.

If the state of the new pool is unlocked, the withdraw function can be called directly to withdraw all of the user's deposits, even if the pool where the first deposit was made was in a locked state.

Code Location:

src/StakeManager.sol

```
    function depositAndLock(uint256 _amount) public nonReentrant {
        Pool storage pool = pools[currentPoolId];

        require(pool.stakingEnabled, "Staking is disabled for this pool");

        require(maskToken.transferFrom(msg.sender, address(this), _amount), "Transfer 
failed");
        userInfos[msg.sender].stakedAmount += _amount;
        // depositAndLock will always stake to currentPoolId
        // it will init userInfos[msg.sender].poolId for the first time
        // it will change userInfos[msg.sender].poolId to currntPoolId(which means new 
pool) when
        //   user deposit after prev pool unlocked
        userInfos[msg.sender].poolId = currentPoolId;

        emit Staked(msg.sender, currentPoolId, _amount);
    }

    function withdraw(uint256 _amount) public nonReentrant {
        Pool storage pool = pools[userInfos[msg.sender].poolId];

        require(pool.unlocked, "Pool is locked");
        require(userInfos[msg.sender].stakedAmount >= _amount, "Insufficient 
balance");

        userInfos[msg.sender].stakedAmount -= _amount;
        require(maskToken.transfer(msg.sender, _amount), "Transfer failed");
        emit unstaked(msg.sender, userInfos[msg.sender].poolId, _amount);
    }
```

Solution

It is recommended to map the funds at each deposit to the currentPoolId at the time of deposit, and to deduct only the funds corresponding to the poolId portion at the time of withdrawal.

Status

Fixed

[N2] [Suggestion] Difference check when changing pools

Category: Design Logic Audit

Content

In the StakeManager contract, the user can update the poolId in the user information to the latest currentPoolId by calling the changePool function. However, there is no check to see if the currentPoolId matches the poolId in the user information.

Code Location:

src/StakeManager.sol

```
    function changePool() public nonReentrant {
        uint8 fromPoolId = userInfos[msg.sender].poolId;
        Pool storage fromPool = pools[userInfos[msg.sender].poolId];
        Pool storage toPool = pools[currentPoolId];

        require(toPool.stakingEnabled, "Staking is disabled for this pool");
        require(fromPool.unlocked, "From pool is locked");
        require(userInfos[msg.sender].stakedAmount > 0, "No staked amount");

        userInfos[msg.sender].poolId = currentPoolId;

        emit StakeChanged(msg.sender, fromPoolId, currentPoolId);
    }
```

Solution

It is recommended to check if the current currentPoolId is not equal to the poolId in the user information.

Status

Fixed

[N3] [Suggestion] Missing return value check

Category: Others

Content

In the Reward contract, the owner can call the emergencyWithdraw function to transfer any tokens in the contract. But it does not check the return value. If external tokens do not adopt the EIP20 standard, it may lead to “false top-up” issues.

Code location:

src/Reward.sol#L57-59

```
    function emergencyWithdraw(address _token, uint256 _amount) public onlyOwner {
        IERC20(_token).transfer(msg.sender, _amount);
    }
```

Solution

It is recommended to add a check return value or use SafeERC20.

Status

Fixed

[N4] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

In the Reward contract, the owner can call the emergencyWithdraw function to transfer any tokens in the contract. If the privilege is lost or misused, there may be an impact on the user's funds.

Code location:

src/Reward.sol#L57-59

```
    function emergencyWithdraw(address _token, uint256 _amount) public onlyOwner {
        IERC20(_token).transfer(msg.sender, _amount);
    }
```

Solution

It is recommended that in the early stages of the project, the core role like the owner should use multi-signatures and the time-lock contract to avoid single-point risks. After the project is running stably, the authority of the core role should be handed over to community governance for management.

Status

Fixed; The owner's permission has been transferred to the multi-signature contract. The transaction is: https://etherscan.io/tx/0xc7785ee7e9a06bf75373359db9a8d15e49aced806d13304c92ae76306476a8d1

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002404220002	SlowMist Security Team	2024.04.22 - 2024.04.22	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 high risk, 1 medium risk, 2 suggestion vulnerabilities. All findings were fixed. The code has been 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.