Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.03.17, the SlowMist security team received the Prosper team's security audit application for Prosper Staking Pool 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

The StakingPool project is designed to distribute rewards to users based on their contributions to a staking pool. This pool accepts an immutable token for staking and now allows claiming multiple different reward tokens. Rewards are provided periodically by an address with the REWARD_PROVIDER role. Users can stake tokens to earn rewards over time, and when they unstake, their tokens enter a bonding period before they can be claimed. Key functions within the contract are restricted by a whitelist, which is managed off-chain through a voting process and set by a script.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Risk of Unintentional Staking	Design Logic Audit	High	Fixed
N2	Token compatibility reminder	Others	Suggestion	Fixed
N3	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/hyplabs/animoca-staking-pool

commit: 9e19d43996f222f46d88a76945365706ff4aa8fe

Fixed Version:

https://github.com/hyplabs/animoca-staking-pool

commit: adf5c08dbcd7a8c8c155151ca2a7cdcc9ed50253

Audit Scope:

```
./contract
├── StakingPool.sol
├── interfaces
├── storage
└── types
```

The main network address of the contract is as follows:

Implementation addresses:

https://bscscan.com/address/0x9fb240751680a03352c26169af5c8d5618631305

https://basescan.org/address/0xce10f2dfeacb9bbf71aaa76f094c6eb3fecb318c

Proxy addresses:

https://bscscan.com/address/0x460dDE85b3CA09e82156E37eFFf50cd07bc3F7f9

https://basescan.org/address/0x460dde85b3ca09e82156e37efff50cd07bc3f7f9

4.2 Visibility Description

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

StakingPool			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_authorizeUpgrade	Internal	Can Modify State	onlyRole
__StakingPool_init	External	Can Modify State	initializer
claim	External	Can Modify State	whenNotPaused
claimReleases	External	Can Modify State	whenNotPaused
claimReleases	External	Can Modify State	-
pause	External	Can Modify State	whenNotPaused onlyRole
provideRewards	External	Can Modify State	whenNotPaused onlyRole
setFeeWallet	External	Can Modify State	onlyRole
setProtocolFeeBP	External	Can Modify State	onlyRole
setStakerShareBP	External	Can Modify State	onlyRole
setTreasury	External	Can Modify State	onlyRole
setUnbondingDuration	External	Can Modify State	onlyRole
setClaimWhitelistRoot	External	Can Modify State	onlyRole
setClaimWhitelistStatus	External	Can Modify State	onlyRole
setStakeWhitelistRoot	External	Can Modify State	onlyRole

StakingPool			
setStakeWhitelistStatus	External	Can Modify State	onlyRole
stake	External	Can Modify State	whenNotPaused
unpause	External	Can Modify State	whenPaused onlyRole
unstake	External	Can Modify State	whenNotPaused
getBoundReleaseAmount	External	-	-
getBoundReleases	External	-	-
getClaimableReleaseAmount	External	-	-
getClaimableReleases	External	-	-
getFeeWallet	External	-	-
getLatestRewardIndex	External	-	-
getProtocolFeeBP	External	-	-
getReward	External	-	-
getRewards	External	-	-
getStakerShareBP	External	-	-
getTreasury	External	-	-
getUnbondingDuration	External	-	-
getUpdatedGlobalTS	External	-	-
getUpdatedUserState	External	-	-
getUpdatedUserData	External	-	-
getUpdatedUserRewards	External	-	-
getStoredGlobalTS	External	-	-
getStoredUserState	External	-	-

StakingPool			
getStoredUserData	External	-	-
getStoredUserRewards	External	-	-
getClaimWhitelistRoot	External	-	-
getClaimWhitelistStatus	External	-	-
getStakeWhitelistRoot	External	-	-
getStakeWhitelistStatus	External	-	-
getUserAccruedTS	External	-	-
getUserAccruedTSFromTo	External	-	-
_distributeRewardAllocations	Internal	Can Modify State	-
_accrueRewards	Internal	Can Modify State	-
_migrateLegacyRewards	Internal	Can Modify State	-
_updateUserRewards	Internal	Can Modify State	-
_processRewards	Internal	Can Modify State	-
_updateGlobalTS	Internal	Can Modify State	-
_updateAccountTS	Internal	Can Modify State	-
_update	Internal	Can Modify State	-
_determineRewardToken	Internal	-	-
_allocateRewards	Internal	-	-
_getClaimableRewards	Internal	-	-
_getUserRewardPeriodTS	Internal	-	-
_getUpdatedUserState	Internal	-	-
_getBoundReleases	Internal	-	-

StakingPool			
_getClaimableReleases	Internal	-	-
_getFirstBoundReleaseIndex	Internal	-	-
_addPendingRewards	Internal	-	-
_enforceBasis	Internal	-	-

4.3 Vulnerability Summary

[N1] [High] Risk of Unintentional Staking

Category: Design Logic Audit

Content

In the StakingPool contract, since the staker parameter and the onBehalfOf parameter in the stake function can be inconsistent, and the staked tokens will be transferred from the staker address to this contract, an unexpected situation may occur:

A malicious user will call the stake function to perform the staking operation. He will specify the onBehalfOf parameter as his own wallet address (assuming this address is in the whitelist if there is a whitelist), and the staker parameter as another user who has previously performed the staking operation. If the token allowance of this other user for the StakingPool contract has not been exhausted or the allowance is type(uint256).max, then the tokens of other users will be used to stake for himself, resulting in losses to the assets of other users.

Code Location:

contract/StakingPool.sol#L355-386

```
    function stake(
         address staker,
         address onBehalfOf,
         uint256 amount,
         bytes32[] memory proof
     ) external whenNotPaused {
         ...

         IERC20(STAKE_TOKEN).safeTransferFrom(
             staker, address(this), amount
         );

         _mint(onBehalfOf, amount);

         emit Staked(staker, onBehalfOf, amount);
     }
```

Solution

It is recommended that during the staking operation, tokens should not be transferred in via the externally arbitrarily specifiable staker parameter. Instead, they should be transferred directly by msg.sender. Alternatively, an allowance limit check should be conducted between staker and msg.sender.

Status

Fixed

[N2] [Suggestion] Token compatibility reminder

Category: Others

Content

In the StakingPool contract, the provideRewards function is used by the reward provider role to transfer reward tokens into the contract for distribution. It will call the _distributeRewardAllocations function to transfer different shares of rewards to the contract itself, the feeWallet address, and the treasury address respectively through the safeTransferFrom function. However, if the reward tokens to be provided are deflationary tokens, the actual number of tokens received by the contract will not match the number of tokens recorded by the contract.

Code Location:

contract/StakingPool.sol#L684-699

```
    function _distributeRewardAllocations(
         address rewardToken,
         RewardAllocations memory allocations
     ) internal {
         Storage.Layout storage $ = Storage.layout();

         IERC20(rewardToken).safeTransferFrom(
             _msgSender(), address(this), allocations.stakers
         );
         IERC20(rewardToken).safeTransferFrom(
             _msgSender(), $.feeWallet, allocations.fees
```

```
        );
        IERC20(rewardToken).safeTransferFrom(
            _msgSender(), $.treasury, allocations.treasury
        );
    }
```

Solution

It is recommended to use the difference between the contract balance before and after the transfer to record the user's actual recharge amount.

Status

Fixed

[N3] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.In the StakingPool contract, MANAGER_ROLE can modify the `unbondingDuration` lock period through the setUnbondingDuration function. In addition, the WHITELIST_ROOT_SETTER_ROLE can modify the `claimWhitelistRoot` and `stakeWhitelistRoot` by calling the setClaimWhitelistRoot and setStakeWhitelistRoot functions to affect the verification of MerkleProof.

Code location:

contract/StakingPool.sol#L283-L300

```
    function setUnbondingDuration(uint256 duration)
        external
        onlyRole(MANAGER_ROLE)
    {
        Storage.layout().unbondingDuration = duration;

        emit UnbondingDurationSet(duration);
    }

   function setClaimWhitelistRoot(bytes32 root)
        external
        onlyRole(WHITELIST_ROOT_SETTER_ROLE)
    {
        Storage.layout().claimWhitelistRoot = root;

        emit ClaimWhitelistRootSet(root);
```

```
     }

    function setStakeWhitelistRoot(bytes32 root)
         external
         onlyRole(WHITELIST_ROOT_SETTER_ROLE)
     {
         Storage.layout().stakeWhitelistRoot = root;

         emit StakeWhitelistRootSet(root);
     }
```

2.The UUPSUpgradeable MANAGER_ROLE relevant authority can upgrade the contract, leading to the risk of over-privileged in this role.

Solution

In the short term, transferring the ownership of core roles to multisig contracts is an effective solution to avoid single-point risk. But in the long run, it is a more reasonable solution to implement a privilege separation strategy and set up multiple privileged roles to manage each privileged function separately. The authority involving user funds should be managed by the community, and the authority involving emergency contract suspension can be managed by the EOA address. This ensures both a quick response to threats and the safety of user funds.

Status

Acknowledged; The project team responded: We acknowledge this concern. The DEFAULT_ADMIN will be held only by the multisig and MANAGER_ROLE will be strictly managed.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002503180003	SlowMist Security Team	2025.03.17 - 2025.03.18	Medium 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, and 1 suggestion. All the findings were fixed and acknowledged. The current risk level is temporarily rated as medium simply because the centralized control permissions have not yet been managed.

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.

SLOWMIST

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist