Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.03.07, the SlowMist security team received the StakeStone team's security audit application for StakeStone - Restaking, 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 audit is primarily an iterative audit of the new stakestone modules.

The new module focuses on allowing users to stake and profit from the Eigenlayer protocol through stakestone.

Different pods are managed each time by creating different Account contracts.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Lack of update on the value of withdrawingNodeAmount and activeNodeAmount	Design Logic Audit	High	Fixed
N2	Incorrect function call target	Others	High	Fixed
N3	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged
N4	Authority transfer enhancement	Others	Suggestion	Fixed
N5	Missing event record	Others	Suggestion	Ignored

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/stakestone/stone-vault-v1

commit: ccab0ca8db37c759a289402c8fbb109266c375b7

Fixed Version:

https://github.com/stakestone/stone-vault-v1

commit: 100e73627150e2eaf8298c485240ccdf8d4f5ec4

Audit scope:

- contracts/interfaces/IBatchDeposit.sol
- contracts/interfaces/IEigenPod.sol
- contracts/interfaces/IEigenPodManager.sol
- contracts/strategies/eigen/Account.sol
- contracts/strategies/eigen/EigenNativeRestakingStrategy.sol

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:

Account			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
transferAdmin	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
invoke	Public	Can Modify State	onlyAuth

EigenNativeRestakingStrategy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Strategy
deposit	Public	Payable	onlyController notAtSameBlock
withdraw	Public	Can Modify State	onlyController notAtSameBlock
instantWithdraw	Public	Can Modify State	onlyController notAtSameBlock
_withdraw	Internal	Can Modify State	-
clear	Public	Can Modify State	onlyController
getAllValue	Public	Can Modify State	-
getInvestedValue	Public	Can Modify State	-
getEigenPodsValue	Public	Can Modify State	-
createEigenPod	External	Can Modify State	-

EigenNativeRestakingStrategy			
stakeToNodeOperators	External	Can Modify State	onlyGovernance
finalizePendingNode	External	Can Modify State	onlyGovernance
unstakeFromNodeOperators	External	Can Modify State	onlyGovernance
unstakeFromEigenPod	External	Can Modify State	onlyGovernance
finalizeWithdrawingNode	External	Can Modify State	onlyGovernance
verifyAndProcessWithdrawals	External	Can Modify State	onlyGovernance
verifyWithdrawalCredentials	External	Can Modify State	onlyGovernance
getEigenPods	External	-	-
getEigenPodsLength	External	-	-
setNewEigenPodManager	External	Can Modify State	onlyGovernance
bytesToAddress	Internal	-	-
<Receive Ether>	External	Payable	-

4.3 Vulnerability Summary

[N1] [High] Lack of update on the value of withdrawingNodeAmount and activeNodeAmount

Category: Design Logic Audit

Content

In the EigenNativeRestakingStrategy contract, the unstakeFromEigenPod function is used to submit a request for the cancellation of the stake, however the values of withdrawingNodeAmount and activeNodeAmount are not automatically updated after submitting the request and the _nodeAmount parameter passed in is not used. The normal logic would be that the activeNodeAmount should decrease and the withdrawingNodeAmount should increase after the request is submitted.

Code Location:

contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L156-169

```
    function unstakeFromEigenPod(
        uint256 _nodeAmount,
        address _eigenPod
    ) external onlyGovernance {
        address podOwner = eigenPodOwners[_eigenPod];
        require(podOwner != address(0), "EigenPod not exist");

        Account account = Account(payable(podOwner));
        account.invoke(
            _eigenPod,
            0,
            abi.encodeWithSelector(IEigenPod.withdrawBeforeRestaking.selector)
        );
    }
```

Solution

It is recommended to update the values of withdrawingNodeAmount and activeNodeAmount directly after submitting a request to cancel a stake.

Status

Fixed

[N2] [High] Incorrect function call target

Category: Others

Content

In the EigenNativeRestakingStrategy contract, the finalizeWithdrawingNode function is used to complete the final withdrawal request and transfer the ETH from the podOwner contract back into this contract. In this case, getting the ETH for the cancellation of the stake is done by calling the claimDelayedWithdrawals function to claim the ETH back to podOwner

However, in eigenlayer's protocol, claimDelayedWithdrawals are present in the DelayedWithdrawalRouter contract and not in the eigenPod contract. In the finalizeWithdrawingNode function, the eigenPod contract is called instead, and this incorrect call will cause the final claim operation to fail.

Code Location:

contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L171-190

```
    function finalizeWithdrawingNode(
        uint256 _nodeAmount,
        address _eigenPod
    ) external onlyGovernance {
        address podOwner = eigenPodOwners[_eigenPod];
        require(podOwner != address(0), "EigenPod not exist");

        Account account = Account(payable(podOwner));
        account.invoke(
            _eigenPod,
            0,
            abi.encodeWithSelector(
                IEigenPod.claimDelayedWithdrawals.selector,
                type(uint256).max
            )
        );
        account.invoke(address(this), podOwner.balance, "");

        withdrawingNodeAmount -= _nodeAmount;
    }
```

contracts/interfaces/IEigenPod.sol

```
    function claimDelayedWithdrawals(
        uint256 maxNumberOfDelayedWithdrawalsToClaim
    ) external;
```

Solution

It is recommended that the claimDelayedWithdrawals function should be called specifying that the target of the call is the DelayedWithdrawalRouter contract and not the eigenPod contract.

Status

Fixed

[N3] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

In the EigenNativeRestakingStrategy contract, the Governance role can set the address of the new eigenPodManager by calling the setNewEigenPodManager function and can directly modify the values of the pendingNodeAmount, activeNodeAmount, and withdrawingNodeAmount by calling the finalizePendingNode and unstakeFromNodeOperators functions. If the privilege is lost or misused, this may have an impact on the user's assets.

Code Location:

contracts/strategies/eigen/EigenNativeRestakingStrategy.sol

```
    function finalizePendingNode(uint256 _nodeAmount) external onlyGovernance {
        pendingNodeAmount -= _nodeAmount;
        activeNodeAmount += _nodeAmount;
    }

    function unstakeFromNodeOperators(
        uint256 _nodeAmount
    ) external onlyGovernance {
        activeNodeAmount -= _nodeAmount;
        withdrawingNodeAmount += _nodeAmount;
    }

    ...

    function setNewEigenPodManager(
        address _eigenPodManager
    ) external onlyGovernance {
        emit SetNewEigenPodManager(eigenPodManager, _eigenPodManager);

        eigenPodManager = _eigenPodManager;
    }
```

Solution

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

Status

Acknowledged; Project team response: will use cobo's MPC wallet address for the management of permissions after the project goes live.

[N4] [Suggestion] Authority transfer enhancement

Category: Others

Content

In the Account contract, the admin role does not adopt the pending and access processes. If the admin is incorrectly set, the permission will be lost.

Code Location:

contracts/strategies/eigen/Account.sol#L26-29

```
    function transferAdmin(address _admin) public {
        require(msg.sender == admin, "not admin");
        admin = _admin;
    }
```

Solution

It is recommended to adopt the pending and access processes. Only the new pauserMultisig role can accept the permissions to transfer.

Status

Fixed

[N5] [Suggestion] Missing event record

Category: Others

Content

The following functions in several contracts are for event logging of key parameter settings.

Code Location:

contracts/strategies/eigen/Account.sol#L26-29

```
    function transferAdmin(address _admin) public {
        ...
    }
```

contracts/strategies/eigen/EigenNativeRestakingStrategy.sol

```
    function finalizePendingNode(uint256 _nodeAmount) external onlyGovernance {
        ...
    }

    function unstakeFromNodeOperators(
        uint256 _nodeAmount
    ) external onlyGovernance {
        ...
    }
```

Solution

It is recommended to record events when sensitive parameters are modified for self-inspection or community review.

Status

Ignored

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002403110003	SlowMist Security Team	2024.03.07 - 2024.03.11	Medium Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 2 high risk, 1 medium risk, 2 suggestion vulnerabilities. And 1 suggestion vulnerability was ignored; 1 medium risk vulnerability was Acknowledged; All other findings were fixed. The code was not deployed to the mainnet. Since the permissions of the core roles are not transferred and managed, the risk level of this report is medium risk for the time being. As the project is not yet online and the permission of the core role has not yet been transferred, the level of the report is positioned as medium risk for the time being.

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.