Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.06.24, the SlowMist security team received the StakeStone team's security audit application for StakeStone - NativeLendingETHStrategy&SymbioticDepositWstETHStrategy, 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 mainly focused on two new staking strategies: NativeLendingETHStrategy and SymbioticDepositWstETHStrategy. The owner role can deposit the ETH in the contract into a third-party protocol to obtain profits and mint corresponding deposit certificate tokens, and then redeem the ETH through the deposit certificates.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Potential error in the calculation of the withdrawal amount	Design Logic Audit	Low	Acknowledged
N2	Missing event records	Others	Suggestion	Fixed
N3	Missing zero value check	Others	Suggestion	Fixed
N4	Lack of external interest rate inflation vulnerability check	Design Logic Audit	Low	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

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

commit: 0fe21a5961ea4f86aa0d30c28322584f9b8c815e

Fixed Version:

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

commit: f7fc0bd6f14f6f7047b22093c476e302389bb30f

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:

SymbioticDepositWstETHStrategy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	StrategyV2

SymbioticDepositWstETHStrategy			
deposit	Public	Payable	onlyController notAtSameBlock
withdraw	Public	Can Modify State	onlyController notAtSameBlock
instantWithdraw	Public	Can Modify State	onlyController notAtSameBlock
clear	Public	Can Modify State	onlyController
_withdraw	Internal	Can Modify State	-
getAllValue	Public	Can Modify State	-
getInvestedValue	Public	Can Modify State	-
getWstETHValue	Public	-	-
wrapToWstETH	External	Can Modify State	onlyOwner
unwrapToStETH	External	Can Modify State	onlyOwner
depositIntoSymbiotic	External	Can Modify State	onlyOwner
withdraswFromSymbiotic	External	Can Modify State	onlyOwner
requestToEther	External	Can Modify State	onlyOwner
claimPendingAssets	External	Can Modify State	onlyOwner
claimAllPendingAssets	External	Can Modify State	onlyOwner
checkPendingAssets	Public	Can Modify State	-
<Receive Ether>	External	Payable	-

NativeLendingETHStrategy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	StrategyV2
depositIntoNative	External	Can Modify State	onlyOwner

NativeLendingETHStrategy			
withdrawFromNativeByAmount	External	Can Modify State	onlyOwner
withdrawFromNativeByShare	External	Can Modify State	onlyOwner
getAllValue	Public	Can Modify State	-
getInvestedValue	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
deposit	Public	Payable	onlyController notAtSameBlock
withdraw	Public	Can Modify State	onlyController notAtSameBlock
instantWithdraw	Public	Can Modify State	onlyController notAtSameBlock
clear	Public	Can Modify State	onlyController
_withdraw	Internal	Can Modify State	-

4.3 Vulnerability Summary

[N1] [Low] Potential error in the calculation of the withdrawal amount

Category: Design Logic Audit

Content

In the NativeLendingETHStrategy contract, the owner role can call the withdrawFromNativeByAmount function and the withdrawFromNativeByShare function to redeem the AquaLpToken for ETH. The final calculated withdrawal amount is simply the difference between the current ETH balance of the contract and the ETH balance before the redemption operation. However, the redemption operation first converts the AquaLpToken back to WETH tokens, and then uses the WETH tokens to obtain ETH. If there are surplus WETH tokens in the contract before the redemption (for example, if other users have mistakenly transferred them in), then this excess WETH amount will also be included in the calculated withdrawal amount.

Code Location:

contracts/strategies/NativeLendingETHStrategy.sol#L44-66

```
    function withdrawFromNativeByAmount(
        uint256 _amount
    ) external onlyOwner returns (uint256 withdrawAmount) {
        uint256 beforeBalance = address(this).balance;

        IAquaLpToken(LPTOKEN).redeemUnderlying(_amount);

        WETH.withdraw(WETH.balanceOf(address(this)));

        withdrawAmount = address(this).balance - beforeBalance;
    }

    function withdrawFromNativeByShare(
        uint256 _share
    ) external onlyOwner returns (uint256 withdrawAmount) {
        uint256 beforeBalance = address(this).balance;

        IAquaLpToken(LPTOKEN).redeem(_share);

        WETH.withdraw(WETH.balanceOf(address(this)));

        withdrawAmount = address(this).balance - beforeBalance;
    }
```

Solution

If the intended design is for withdrawAmount to represent only the amount being redeemed, then it is recommended to add the excess WETH token amount in the contract to the beforeBalance calculation before the redemption.

Status

Acknowledged; The project team's response: The expected withdrawalAmount is the amount of ETH that needs to be returned through this function call.

[N2] [Suggestion] Missing event records

Category: Others

Content

In the NativeLendingETHStrategy contract, the owner role can use the ETH in the contract to mint AquaLpToken, and then use the AquaLpToken to redeem for ETH, but there is no event record.

Code Location:

contracts/strategies/NativeLendingETHStrategy.sol#L28-66

```
    function depositIntoNative(
        uint256 _amount
    ) external onlyOwner returns (uint256 mintAmount) {
        ...
    }

    function withdrawFromNativeByAmount(
        uint256 _amount
    ) external onlyOwner returns (uint256 withdrawAmount) {
        ...
    }

    function withdrawFromNativeByShare(
        uint256 _share
    ) external onlyOwner returns (uint256 withdrawAmount) {
        ...
    }
```

Solution

It is recommended to record events when the funds in the contract are being transferred for self-inspection or community review.

Status

Fixed

[N3] [Suggestion] Missing zero value check

Category: Others

Content

In the NativeLendingETHStrategy contract, The owner role does not check whether the input amount for minting and redeeming AquaLpToken tokens is 0 or not. If the input value is 0, the operation can still be executed successfully, but it will consume gas.

Code Location:

contracts/strategies/NativeLendingETHStrategy.sol#L28-66

```
    function depositIntoNative(
        uint256 _amount
    ) external onlyOwner returns (uint256 mintAmount) {
        ...
    }

    function withdrawFromNativeByAmount(
        uint256 _amount
    ) external onlyOwner returns (uint256 withdrawAmount) {
        ...
    }

    function withdrawFromNativeByShare(
        uint256 _share
    ) external onlyOwner returns (uint256 withdrawAmount) {
        ...
    }
```

Solution

It is recommended to add a check for zero values to determine that the value passed in is non-zero.

Status

Fixed

[N4] [Low] Lack of external interest rate inflation vulnerability check

Category: Design Logic Audit

Content

In the NativeLendingETHStrategy and SymbioticDepositWstETHStrategy contracts, the owner role can separately call the depositIntoNative function and the depositIntoSymbiotic function to deposit the funds in the strategy contract into a third-party protocol, and mint the corresponding deposit certificates. However, the functions do not check whether the minted deposit shares are zero in quantity. Since the code of the third-party protocol is not within the scope of this audit, if there is an interest rate inflation vulnerability in the code of the third-party protocol, the funds in the contract may be damaged due to malicious users front-running.

For details on the interest rate inflation vulnerability, please refer to the following link:

https://blog.openzeppelin.com/a-novel-defense-against-erc4626-inflation-attacks

Code Location:

contracts/strategies/NativeLendingETHStrategy.sol#L28-42

```
    function depositIntoNative(
        uint256 _amount
    ) external onlyOwner returns (uint256 mintAmount) {
        ...

        IAquaLpToken(LPTOKEN).mint(_amount);

        mintAmount =
            IAquaLpToken(LPTOKEN).balanceOf(address(this)) -
            beforeLPBalance;
    }
```

contracts/strategies/SymbioticDepositWstETHStrategy.sol#L151-169

```
    function depositIntoSymbiotic(
        uint256 _wstETHAmount
    ) external onlyOwner returns (uint256 shares) {
        ...

        shares = ICollateral(collateralAddr).deposit(
            address(this),
            _wstETHAmount
        );

        emit DepositIntoSymbiotic(
            collateralAddr,
            address(this),
            _wstETHAmount,
            shares
        );
    }
```

Solution

It is recommended to add a check that the minted deposit shares are not zero, to prevent the interest rate inflation vulnerability.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002406240001	SlowMist Security Team	2024.06.24 - 2024.06.24	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 2 low risk and 2 suggestion vulnerabilities. All the findings were Fixed and Acknowledged. 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