Smart Contract

Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.07.19, the SlowMist security team received the StakeStone team's security audit application for StakeStone - MellowDepositWstETHStrategy, 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 a new staking strategies: MellowDepositWstETHStrategy. The owner role can wrap ETH in the contract into wstETH and deposit the wstETH in the contract into the third-party protocol, and then redeem the deposit certificate in the contract into wstETH.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Tokens Obtained from Emergency Withdrawal Partly Locked	Design Logic Audit	Critical	Fixed
N2	The Potential Risk of Fixed Array Lengths	Design Logic Audit	Suggestion	Fixed
N3	Conflict in withdrawal requests	Design Logic Audit	Information	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

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

commit: 83abe03e62ba452940e862019224e4bc630ff100

Fixed Version:

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

commit: 75502741e7695fd9fa46a2f05e21cc5d0eeab273

Audit scope:

- /contracts/strategies/MellowDepositWstETHStrategy.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:

MellowDepositWstETHStrategy			
Function Name	Visibility	Mutability	Modifiers

MellowDepositWstETHStrategy			
<Constructor>	Public	Can Modify State	StrategyV2
deposit	Public	Payable	onlyController
withdraw	Public	Can Modify State	onlyController
instantWithdraw	Public	Can Modify State	onlyController
clear	Public	Can Modify State	onlyController
_withdraw	Internal	Can Modify State	-
getAllValue	Public	Can Modify State	-
getInvestedValue	Public	Can Modify State	-
getWstETHValue	Public	-	-
getDepositedValue	Public	-	-
mintToWstETH	External	Can Modify State	onlyOwner
wrapToWstETH	External	Can Modify State	onlyOwner
unwrapToStETH	External	Can Modify State	onlyOwner
depositIntoMellow	External	Can Modify State	onlyOwner
requestWithdrawFromMellow	External	Can Modify State	onlyOwner
emergencyWithdrawFromMellow	External	Can Modify State	onlyOwner
cancelWithdrawFromMellow	External	Can Modify State	onlyOwner
getPendingValueFromMellow	Public	-	-
requestToEther	External	Can Modify State	onlyOwner
claimPendingAssets	External	Can Modify State	onlyOwner
claimAllPendingAssets	External	Can Modify State	onlyOwner
checkPendingAssets	Public	Can Modify State	-

MellowDepositWstETHStrategy			
getLpRate	Public	-	-
<Receive Ether>	External	Payable	-

4.3 Vulnerability Summary

[N1] [Critical] Tokens Obtained from Emergency Withdrawal Partly Locked

Category: Design Logic Audit

Content

In the MellowDepositWstETHStrategy contract, the owner role can execute an emergency withdrawal operation from the mellowVault contract by calling the emergencyWithdrawFromMellow function.

In the mellowVault contract, during an emergency withdrawal operation, the specified amount of LP tokens from previous withdrawal requests are burned, and two types of tokens, wstETH and DC_wstETH, are transferred to the address indicated in the withdrawal request. The amounts transferred are calculated based on the burned LP tokens and the current balance of these two tokens in the pool.

However, in the MellowDepositWstETHStrategy contract, there is no implementation for redeeming DC_wstETH tokens back into wstETH, which results in these DC_wstETH tokens being locked within the contract.

Furthermore, when calculating the total invested value of the contract using the getInvestedValue function, it fails to account for the value of the held DC_wstETH tokens.

Code Location:

contracts/strategies/MellowDepositWstETHStrategy.sol

```
    function getInvestedValue() public override returns (uint256 value) {
        uint256 etherValue = address(this).balance;
        uint256 stETHValue = IERC20(stETHAddr).balanceOf(address(this));
        (, uint256 claimableValue, uint256 pendingValue) = checkPendingAssets();
        uint256 mellowPending = getPendingValueFromMellow();

        value =
            etherValue +
            stETHValue +
            claimableValue +
            pendingValue +
            getWstETHValue() +
            getDepositedValue() +
            mellowPending;
    }

    ...

    function emergencyWithdrawFromMellow(
        uint256[] memory _minAmounts,
        uint256 _deadline
    ) external onlyOwner returns (uint256 wstETHAmount) {
        IMellowVault(mellowVaultAddr).emergencyWithdraw(_minAmounts, _deadline);
    }
```

Solution

It is recommended to implement a function within the contract that facilitates the redemption of DC_wstETH tokens into wstETH. Additionally, the calculation of the total invested value, as performed by the getInvestedValue function, should be updated to include the Ether (ETH) value corresponding to the held DC_wstETH tokens. This would ensure comprehensive asset management and accurate representation of the contract's overall investment worth.

Status

Fixed

[N2] [Suggestion] The Potential Risk of Fixed Array Lengths

Category: Design Logic Audit

Content

In the MellowDepositWstETHStrategy contract, the owner role can call the depositIntoMellow function to deposit wstETH tokens from the contract into the MellowVault, where the length of the passed amounts array is fixed at 1. Within the deposit function of the MellowVault contract, a check is performed to ensure that the lengths of the contract's _underlyingTokens array and the passed amounts array are equal.

Currently, as the _underlyingTokens array in the MellowVault contract also contains only 1 element, this check passes successfully. However, the MellowVault contract features a function (addToken) that allows for adding new token data to the _underlyingTokens array. If, in the future, the _underlyingTokens array expands due to the addition of new tokens, the depositIntoMellow function may fail this length check and consequently be unable to execute properly. The same issues also apply when making a withdrawal request.

Code Location:

contracts/strategies/MellowDepositWstETHStrategy.sol#L177&L200

```
function depositIntoMellow(
    uint256 _wstETHAmount,
    uint256 _minLpAmount
) external onlyOwner returns (uint256 lpAmount) {
    require(_wstETHAmount != 0, "zero");

    TransferHelper.safeApprove(wstETHAddr, mellowVaultAddr, _wstETHAmount);

    uint256[] memory amounts = new uint256[](1);
    amounts[0] = _wstETHAmount;
    (, lpAmount) = IMellowVault(mellowVaultAddr).deposit(
        address(this),
        amounts,
        _minLpAmount,
        block.timestamp
    );

    emit DepositIntoMellow(
        mellowVaultAddr,
        address(this),
        _wstETHAmount,
        lpAmount
    );
}

function requestWithdrawFromMellow(
    uint256 _share,
    uint256 _minAmount
) external onlyOwner {
    require(_share != 0, "zero");

    uint256[] memory amounts = new uint256[](1);
    amounts[0] = _minAmount;
    IMellowVault(mellowVaultAddr).registerWithdrawal(
        address(this),
        _share,
        amounts,
        block.timestamp,
        type(uint256).max,
        true
    );

    emit WithdrawFromMellow(mellowVaultAddr, address(this), _share);
}
```

Solution

It is recommended that the length of the amounts array be set dynamically based on the current length of the _underlyingTokens array in the MellowVault contract, rather than being hardcoded to 1.

Status

Fixed

[N3] [Information] Conflict in withdrawal requests

Category: Design Logic Audit

Content

In the MellowDepositWstETHStrategy contract, The owner role can initiate a withdrawal request for wstETH by calling the requestWithdrawFromMellow function, with the closePrevious parameter set to true by default. This implies that if a previous withdrawal request has been submitted and is still pending, it will first be canceled before replacing it with the newly submitted withdrawal request.

Code Location:

contracts/strategies/MellowDepositWstETHStrategy.sol#L208

```
    function requestWithdrawFromMellow(
        uint256 _share,
        uint256 _minAmount
    ) external onlyOwner {
        require(_share != 0, "zero");

        uint256[] memory amounts = new uint256[](1);
        amounts[0] = _minAmount;
        IMellowVault(mellowVaultAddr).registerWithdrawal(
            address(this),
            _share,
            amounts,
            block.timestamp,
            type(uint256).max,
            true
        );
        emit WithdrawFromMellow(mellowVaultAddr, address(this), _share);
    }
```

Solution

If the project team's expectation aligns with this behavior, no modification is needed; however, if the project team wishes to process withdrawal requests sequentially (meaning the next withdrawal can only be initiated after the previous one is completed), it is advised to adjust the parameter from true to false.

Status

Acknowledged; Project team response: It conforms to the intended design where each new request to withdraw overrides any pending, unprocessed withdrawal request. If the parameter is set to false, attempting to submit a new withdrawal request before the previous one is processed will result in an error, preventing further requests until the initial one is completed.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002407190002	SlowMist Security Team	2024.07.19 - 2024.07.19	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, 1 suggestion vulnerabilities and 1 information. 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.

SLOWMIST

Official Website

www.slowmist.com

✉

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist