Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.11.11, the SlowMist security team received the Webera team's security audit application for Webera, 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 is a decentralized farming protocol that mainly includes Vault and BeraBend strategies.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Risk of withdrawal failure in the withdraw operation	Others	Medium	Fixed

NO	Title	Category	Level	Status
N2	Issue of unupdated strategy debt in the withdraw function	Others	Critical	Fixed
N3	Recommendation to add reentrancy protection	Reentrancy Vulnerability	Suggestion	Fixed
N4	Missing event record	Malicious Event Log Audit	Suggestion	Fixed
N5	Preemptive Initialization	Race Conditions Vulnerability	Suggestion	Acknowledged
N6	Risk of excessive authority	Authority Control Vulnerability Audit	Low	Acknowledged

4 Code Overview

4.1 Contracts Description

https://github.com/webera-dev/webera-contracts

commit: 4cc9ebd13046db2dad78d8d80a8ff55c65e5b7c5

review commit: 4e2667b8d5d15b66e0fbf00571b67b06d9cc8200

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:

Vault			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
deposit	Public	Can Modify State	-

Vault			
_deposit	Internal	Can Modify State	-
withdraw	Public	Can Modify State	-
mint	Public	Can Modify State	-
redeem	Public	Can Modify State	-
harvest	Public	Can Modify State	onlyOwner
harvestAll	Public	Can Modify State	onlyOwner
_calcuateProfitLoss	Internal	-	-
_syncAssetsAndShares	Internal	Can Modify State	-
_updateUnlockingSchedule	Internal	Can Modify State	-
_updateStrategyParams	Internal	Can Modify State	-
_accountFees	Internal	-	-
addStrategy	External	Can Modify State	onlyOwner
revokeStrategy	External	Can Modify State	onlyOwner
updateStrategyMaxDebt	External	Can Modify State	onlyOwner
updateDebt	External	Can Modify State	onlyOwner
_withdrawFromStrategy	Internal	Can Modify State	-
_assessSharesOfUnrealizedLosses	Internal	-	-
strategyParams	Public	-	-
strategiesLength	External	-	-
withdrawQueueLength	External	-	-
getWithdrawQueue	External	-	-
getStrategies	External	-	-

Vault			
unlockedShares	Public	-	-
_unlockedShares	Internal	-	-
totalAssets	Public	-	-
totalSupply	Public	-	-
withdrawLimit	Public	-	-
setProfitMaxUnlockTime	External	Can Modify State	onlyOwner
setWithdrawQueue	External	Can Modify State	onlyOwner
emergencyWithdrawAll	External	Can Modify State	onlyOwner
emergencyVault	External	Can Modify State	onlyOwner
emergencyStrategy	External	Can Modify State	onlyOwner
protocol_fee_config	External	-	-

HoneyBeraBendStrategy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BaseStrategy
_deployFunds	Internal	Can Modify State	-
_checkAllowance	Internal	Can Modify State	-
_freeFunds	Internal	Can Modify State	-
_harvestAndReport	Internal	Can Modify State	-
_emergencyWithdraw	Internal	Can Modify State	-
getUtilizationRate	Public	-	-

BaseStrategy			
Function Name	Visibility	Mutability	Modifiers
_onlySelf	Internal	-	-
<Constructor>	Public	Can Modify State	-
_deployFunds	Internal	Can Modify State	-
_freeFunds	Internal	Can Modify State	-
_harvestAndReport	Internal	Can Modify State	-
_tend	Internal	Can Modify State	-
_tendTrigger	Internal	-	-
tendTrigger	External	-	-
availableDepositLimit	Public	-	-
availableWithdrawLimit	Public	-	-
_emergencyWithdraw	Internal	Can Modify State	-
deployFunds	External	Can Modify State	onlySelf
freeFunds	External	Can Modify State	onlySelf
harvestAndReport	External	Can Modify State	onlySelf
tendThis	External	Can Modify State	onlySelf
shutdownWithdraw	External	Can Modify State	onlySelf
_delegateCall	Internal	Can Modify State	-
<Fallback>	External	Can Modify State	-

TokenizedStrategy			
Function Name	Visibility	Mutability	Modifiers
requireManagement	Public	-	-

TokenizedStrategy			
requireKeeperOrManagement	Public	-	-
requireEmergencyAuthorized	Public	-	-
_strategyStorage	Internal	-	-
initialize	External	Can Modify State	-
deposit	External	Can Modify State	nonReentrant
mint	External	Can Modify State	nonReentrant
withdraw	External	Can Modify State	-
withdraw	Public	Can Modify State	nonReentrant
redeem	External	Can Modify State	-
redeem	Public	Can Modify State	nonReentrant
totalAssets	External	-	-
totalSupply	External	-	-
convertToShares	External	-	-
convertToAssets	External	-	-
previewDeposit	External	-	-
previewMint	External	-	-
previewWithdraw	External	-	-
previewRedeem	External	-	-
maxDeposit	External	-	-
maxMint	External	-	-

TokenizedStrategy			
maxWithdraw	External	-	-
maxRedeem	External	-	-
_totalAssets	Internal	-	-
_totalSupply	Internal	-	-
_convertToShares	Internal	-	-
_convertToAssets	Internal	-	-
_maxDeposit	Internal	-	-
_maxMint	Internal	-	-
_maxWithdraw	Internal	-	-
_maxRedeem	Internal	-	-
_deposit	Internal	Can Modify State	-
_withdraw	Internal	Can Modify State	-
report	External	Can Modify State	nonReentrant onlyKeepers
unlockedShares	External	-	-
_unlockedShares	Internal	-	-
tend	External	Can Modify State	nonReentrant onlyKeepers
shutdownStrategy	External	Can Modify State	onlyEmergencyAuthorized
emergencyWithdraw	External	Can Modify State	nonReentrant onlyEmergencyAuthorized
asset	External	-	-
apiVersion	External	-	-
management	External	-	-

TokenizedStrategy			
pendingManagement	External	-	-
keeper	External	-	-
emergencyAdmin	External	-	-
performanceFee	External	-	-
performanceFeeRecipient	External	-	-
fullProfitUnlockDate	External	-	-
profitUnlockingRate	External	-	-
profitMaxUnlockTime	External	-	-
lastReport	External	-	-
pricePerShare	External	-	-
isShutdown	External	-	-
setPendingManagement	External	Can Modify State	onlyManagement
acceptManagement	External	Can Modify State	-
setKeeper	External	Can Modify State	onlyManagement
setEmergencyAdmin	External	Can Modify State	onlyManagement
setPerformanceFee	External	Can Modify State	onlyManagement
setPerformanceFeeRecipient	External	Can Modify State	onlyManagement
setProfitMaxUnlockTime	External	Can Modify State	onlyManagement
name	External	-	-
symbol	External	-	-
decimals	External	-	-

TokenizedStrategy			
balanceOf	External	-	-
_balanceOf	Internal	-	-
transfer	External	Can Modify State	-
allowance	External	-	-
_allowance	Internal	-	-
approve	External	Can Modify State	-
transferFrom	External	Can Modify State	-
_transfer	Internal	Can Modify State	-
_mint	Internal	Can Modify State	-
_burn	Internal	Can Modify State	-
_approve	Internal	Can Modify State	-
_spendAllowance	Internal	Can Modify State	-
nonces	External	-	-
permit	External	Can Modify State	-
DOMAIN_SEPARATOR	Public	-	-
_computeDomainSeparator	Internal	-	-
<Constructor>	Public	Can Modify State	-

4.3 Vulnerability Summary

[N1] [Medium] Risk of withdrawal failure in the withdraw operation

Category: Others

Content

In the `_withdrawFromStrategy` function, there is a check using `require(_strategyParams[strategy_].currentDebt >= assetsToWithdraw_, "Vault: Not enough debt in the strategy");.` If the amount of debt being withdrawn from the strategy exceeds its current debt, this will prevent further withdrawals from other strategies.

- contracts/core/Vault.sol

```
function withdraw(uint256 assets, address receiver, address owner) public virtual
override returns (uint256) {

        uint256 maxAssets = maxWithdraw(owner);
        if (assets > maxAssets) {
            revert ERC4626ExceededMaxWithdraw(owner, assets, maxAssets);
        }

        if (assets > totalIdleAssets) {
            uint256 assetsToPull = assets - totalIdleAssets;


            for (uint256 i = 0; i < withdrawQueue.length; i++) {
                if (withdrawQueue[i] == address(0)) {
                    break;
                }

                if (assetsToPull == 0) {
                    break;
                }

                address strategyToPullFrom = withdrawQueue[i];
                uint256 actualAssetsWithdrawn =
_withdrawFromStrategy(strategyToPullFrom, assetsToPull); //@SlowMist

                assetsToPull -= actualAssetsWithdrawn;
            }
        }


        uint256 shares = previewWithdraw(assets);


        require(totalIdleAssets >= assets, "Vault: Not enough assets in the vault");
```

```
        _withdraw(_msgSender(), receiver, owner, assets, shares);


        totalIdleAssets -= assets;


        return shares;
}

function _withdrawFromStrategy(address strategy_, uint256 assetsToWithdraw_)
        internal
        returns (uint256 actualAssetsWithdrawn)
    {
        require(strategy_ != address(0), "Vault: No strategy to pull assets from");
        require(_strategyParams[strategy_].isActivated, "Vault: Strategy not
activated");
        require(_strategyParams[strategy_].currentDebt >= assetsToWithdraw_, "Vault:
Not enough debt in the strategy");//@SlowMist
        uint256 assetsPreWithdraw = IERC20(asset()).balanceOf(address(this));

        IERC4626(strategy_).withdraw(assetsToWithdraw_, address(this),
address(this));

        uint256 assetsPostWithdraw = IERC20(asset()).balanceOf(address(this));

        actualAssetsWithdrawn = Math.min(assetsPostWithdraw - assetsPreWithdraw,
assetsToWithdraw_);

        totalIdleAssets += actualAssetsWithdrawn;
        totalOutstandingDebt -= actualAssetsWithdrawn;
    }
```

Solution

First, calculate the maximum amount of assets available for withdrawal, and then retrieve the remaining amount from the next strategy.

Status

Fixed

[N2] [Critical] Issue of unupdated strategy debt in the withdraw function

Category: Others

Content

In the withdraw function, after asset withdrawal, the strategy's debt is not updated, which may result in inaccurate debt data for the protocol.

- contracts/core/Vault.sol

```
function withdraw(uint256 assets, address receiver, address owner) public virtual
override returns (uint256) {

        uint256 maxAssets = maxWithdraw(owner);
        if (assets > maxAssets) {
            revert ERC4626ExceededMaxWithdraw(owner, assets, maxAssets);
        }

        if (assets > totalIdleAssets) {
            uint256 assetsToPull = assets - totalIdleAssets;


            for (uint256 i = 0; i < withdrawQueue.length; i++) {
                if (withdrawQueue[i] == address(0)) {
                    break;
                }

                if (assetsToPull == 0) {
                    break;
                }

                address strategyToPullFrom = withdrawQueue[i];
                uint256 actualAssetsWithdrawn =
_withdrawFromStrategy(strategyToPullFrom, assetsToPull);

                assetsToPull -= actualAssetsWithdrawn;
            }
        }


        uint256 shares = previewWithdraw(assets);


        require(totalIdleAssets >= assets, "Vault: Not enough assets in the vault");

        _withdraw(_msgSender(), receiver, owner, assets, shares);


        totalIdleAssets -= assets;


        return shares;
```

```
}

function _withdrawFromStrategy(address strategy_, uint256 assetsToWithdraw_)
        internal
        returns (uint256 actualAssetsWithdrawn)
    {
        require(strategy_ != address(0), "Vault: No strategy to pull assets from");
        require(_strategyParams[strategy_].isActivated, "Vault: Strategy not
activated");
        require(_strategyParams[strategy_].currentDebt >= assetsToWithdraw_, "Vault:
Not enough debt in the strategy");
        uint256 assetsPreWithdraw = IERC20(asset()).balanceOf(address(this));

        IERC4626(strategy_).withdraw(assetsToWithdraw_, address(this),
address(this));

        uint256 assetsPostWithdraw = IERC20(asset()).balanceOf(address(this));

        actualAssetsWithdrawn = Math.min(assetsPostWithdraw - assetsPreWithdraw,
assetsToWithdraw_);

        totalIdleAssets += actualAssetsWithdrawn;
        totalOutstandingDebt -= actualAssetsWithdrawn;
    }
```

Solution

In the withdraw function, it is necessary to update the strategy's debt after asset withdrawal.

Status

Fixed

[N3] [Suggestion] Recommendation to add reentrancy protection

Category: Reentrancy Vulnerability

Content

In the withdraw logic, there is a call to transfer funds to the receiver. If the token includes a callback mechanism, this may introduce a reentrancy risk. It is recommended to implement a reentrancy lock on other external functions within the contract for protection.

- contracts/core/Vault.sol

```
function withdraw(uint256 assets, address receiver, address owner) public virtual
override returns (uint256) {
```

```
        uint256 maxAssets = maxWithdraw(owner);
        if (assets > maxAssets) {
            revert ERC4626ExceededMaxWithdraw(owner, assets, maxAssets);
        }

        if (assets > totalIdleAssets) {
            uint256 assetsToPull = assets - totalIdleAssets;


            for (uint256 i = 0; i < withdrawQueue.length; i++) {
                if (withdrawQueue[i] == address(0)) {
                    break;
                }

                if (assetsToPull == 0) {
                    break;
                }

                address strategyToPullFrom = withdrawQueue[i];
                uint256 actualAssetsWithdrawn =
_withdrawFromStrategy(strategyToPullFrom, assetsToPull);

                assetsToPull -= actualAssetsWithdrawn;
            }
        }


        uint256 shares = previewWithdraw(assets);


        require(totalIdleAssets >= assets, "Vault: Not enough assets in the vault");

        _withdraw(_msgSender(), receiver, owner, assets, shares);


        totalIdleAssets -= assets;


        return shares;
}
```

Solution

It is recommended to implement a reentrancy lock on other external functions within the contract for protection.

Status

Fixed

[N4] [Suggestion] Missing event record

Category: Malicious Event Log Audit

Content

The changes to the following key parameters have not been logged with corresponding events.

- contracts/core/Vault.sol

```
function setProfitMaxUnlockTime(uint256 profitMaxUnlockTime_) external onlyOwner {
    profitMaxUnlockTime = profitMaxUnlockTime_;
}
```

Solution

Record the corresponding event.

Status

Fixed

[N5] [Suggestion] Preemptive Initialization

Category: Race Conditions Vulnerability

Content

By calling the initialize function to initialize the contract, there is a potential issue that malicious attackers preemptively call the initialize function to initialize.

- contracts/core/Vault.sol

```
    function initialize(
        address _asset,
        string memory _name,
        address _management,
        address _performanceFeeRecipient,
        address _keeper
    ) external {
        // Cache storage pointer.
        StrategyData storage S = _strategyStorage();
```

```
        // Make sure we aren't initialized.
        require(address(S.asset) == address(0), "initialized");

        // Set the strategy's underlying asset.
        S.asset = ERC20(_asset);
        // Set the Strategy Tokens name.
        S.name = _name;
        // Set decimals based off the `asset`.
        S.decimals = ERC20(_asset).decimals();
        // Set initial chain id for permit replay protection.
        require(block.chainid < type(uint88).max, "invalid chain id");
        S.INITIAL_CHAIN_ID = uint88(block.chainid);
        // Set the initial domain separator for permit functions.
        S.INITIAL_DOMAIN_SEPARATOR = _computeDomainSeparator(S);

        // Default to a 10 day profit unlock period.
        S.profitMaxUnlockTime = 10 days;
        // Set address to receive performance fees.
        // Can't be address(0) or we will be burning fees.
        require(_performanceFeeRecipient != address(0), "ZERO ADDRESS");
        // Can't mint shares to its self because of profit locking.
        require(_performanceFeeRecipient != address(this), "self");
        S.performanceFeeRecipient = _performanceFeeRecipient;
        // Default to a 10% performance fee.
        S.performanceFee = 1_000;
        // Set last report to this block.
        S.lastReport = uint96(block.timestamp);

        // Set the default management address. Can't be 0.
        require(_management != address(0), "ZERO ADDRESS");
        S.management = _management;
        // Set the keeper address
        S.keeper = _keeper;

        // Emit event to signal a new strategy has been initialized.
        emit NewTokenizedStrategy(address(this), _asset, API_VERSION);
    }
```

Solution

It is suggested that the initialization operation can be called in the same transaction immediately after the contract is created to avoid being maliciously called by the attacker.

Status

Acknowledged

[N6] [Low] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

In the protocol, the `owner` has the authority to configure critical parameters and the right to perform emergency withdrawals. if the owner's private key is compromised, it could pose a significant risk to the protocol's asset security.

- contracts/core/Vault.sol

```
Owner can setProfitMaxUnlockTime
Owner can setWithdrawQueue
Owner can emergencyWithdrawAll
Owner can emergencyVault
Owner can emergencyStrategy
```

In the protocol, `management` has the ability to configure critical permissions and control the settings of other roles. If the management's private key is compromised, it could lead to severe consequences for the protocol's operations.

- contracts/tokenized-strategy/TokenizedStrategy.sol

```
management can setKeeper
management can setPendingManagement
management can setEmergencyAdmin
management can setPerformanceFee
management can setPerformanceFeeRecipient
management can setProfitMaxUnlockTime
management can report
management can tend
management can shutdownStrategy
management can emergencyWithdraw
Keeper can report
Keeper can tend
emergencyAdmin can emergencyWithdraw
emergencyAdmin can shutdownStrategy
```

Solution

In the short term, during the early stages of the project, the protocol may need to frequently set various parameters to ensure the stable operation of the protocol. Therefore, transferring the ownership of core roles to a multisig

management can effectively solve the single-point risk, but it cannot mitigate the excessive privilege risk. In the long run, after the protocol stabilizes, transferring the owner ownership to community governance and executing through a timelock can effectively mitigate the excessive privilege risk and increase the community users' trust in the protocol.

Status

Acknowledged; The project team will use multisig to manage the owner address and introduce governance with a timelock upon the mainnet launch to mitigate risks.

Currently, the contract of the project's vault has not been open-sourced, but it can be seen through tools that the owner has used a multi-signature address for control. The contract address is 0xf04c74768445c6b444e38c1ed44E3C6548c62fF8.

ERC-20: webera-vault-wbera (weWBERA)

owner:0xf04c74768445c6b444e38c1ed44E3C6548c62fF8

https://berascan.com/address/0x55a050f76541c2554e9dfa3a0b4e665914bf92ea

ERC-20: webera-vault-honey (weHONEY)

owner:0xf04c74768445c6b444e38c1ed44E3C6548c62fF8

https://berascan.com/address/0x4ead3867554e597c7b0d511dc68cead59286870d

ERC-20: webera-vault-nect (weNECT)

owner:0xf04c74768445c6b444e38c1ed44E3C6548c62fF8

https://berascan.com/address/0xb96723631a16723e9b0f22d4d91117160e7f8aba

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002411190001	SlowMist Security Team	2024.11.11 - 2024.11.19	Low 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 1 critical risk, 2 medium risk, 3 suggestion vulnerabilities.

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