Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.05.21, the SlowMist security team received the PancakeSwap team's security audit application for CakeVault, 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 party 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.

Level	Description
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:

- Reentrancy Vulnerability
- Replay Vulnerability
- Reordering Vulnerability
- Short Address Vulnerability
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Uninitialized Storage Pointers Vulnerability
- Arithmetic Accuracy Deviation Vulnerability
- tx.origin Authentication Vulnerability

- "False top-up" Vulnerability
- Variable Coverage Vulnerability
- Gas Optimization Audit
- Malicious Event Log Audit
- Redundant Fallback Function Audit
- Unsafe External Call Audit
- Explicit Visibility of Functions State Variables Aduit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

Just stake some tokens to earn. High APR, low risk.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	The deflationary token docking issue	Others	Suggestion	Confirmed
N2	CakeAtLastUserAction parameter record error issue	Design Logic Audit	Suggestion	Confirmed
N3	Missing event records	Others	Suggestion	Confirmed

NO	Title	Category	Level	Status
N4	Emergency withdrawal issue	Design Logic Audit	Low	Fixed

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

Address: 0xa80240Eb5d7E05d3F250cF000eEc0891d00b51CC

The CakeVault Contract Link Address:

https://bscscan.com/address/0xa80240Eb5d7E05d3F250cF000eEc0891d00b51CC

The VaultOwner Contract Link Address:

https://www.bscscan.com/address/0xb6958D19b60e5fC85908C67c37a5b954E9D60d99

4.2 Visibility Description

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

CakeVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
deposit	External	Can Modify State	whenNotPaused notContract
withdrawAll	External	Can Modify State	notContract

CakeVault			
harvest	External	Can Modify State	notContract whenNotPaused
setAdmin	External	Can Modify State	onlyOwner
setTreasury	External	Can Modify State	onlyOwner
setPerformanceFee	External	Can Modify State	onlyAdmin
setCallFee	External	Can Modify State	onlyAdmin
setWithdrawFee	External	Can Modify State	onlyAdmin
setWithdrawFeePeriod	External	Can Modify State	onlyAdmin
emergencyWithdraw	External	Can Modify State	onlyAdmin
inCaseTokensGetStuck	External	Can Modify State	onlyAdmin
pause	External	Can Modify State	onlyAdmin whenNotPaused
unpause	External	Can Modify State	onlyAdmin whenPaused
calculateHarvestCakeRewards	External	-	-
calculateTotalPendingCakeRewards	External	-	-
getPricePerFullShare	External	-	-
withdraw	Public	Can Modify State	notContract
available	Public	-	-
balanceOf	Public	-	-

CakeVault			
_earn	Internal	Can Modify State	-
_isContract	Internal	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] The deflationary token docking issue

Category: Others

Content

Users can transfer the cake token into the vault contract through the deposit function. Under normal circumstances, the number of staking tokens transferred by the user is the same as the `_amount` parameter passed in. But if the staking token is a deflationary token, the number of tokens transferred by the user may be different from the number of tokens actually received in the contract.

Code location:

```
function deposit(uint256 _amount) external whenNotPaused notContract {
    require(_amount > 0, "Nothing to deposit");

    uint256 pool = balanceOf();
    token.safeTransferFrom(msg.sender, address(this), _amount);
    uint256 currentShares = 0;
    if (totalShares != 0) {
        currentShares = (_amount.mul(totalShares)).div(pool);
    } else {
        currentShares = _amount;
    }
    UserInfo storage user = userInfo[msg.sender];

    user.shares = user.shares.add(currentShares);
    user.lastDepositedTime = block.timestamp;

    totalShares = totalShares.add(currentShares);
```

```
        user.cakeAtLastUserAction = user.shares.mul(balanceOf()).div(totalShares);
        user.lastUserActionTime = block.timestamp;

        _earn();

        emit Deposit(msg.sender, _amount, currentShares, block.timestamp);
    }
```

Solution

If the staking token of this Vault contract is a deflationary token, it is recommended to check the token balance of the contract before and after the user transfer.

Status

Confirmed

[N2] [Suggestion] CakeAtLastUserAction parameter record error issue

Category: Design Logic Audit

Content

In the Vault contract, the user can withdraw the funds staked by the user through the withdraw function. If the user does not withdraw all funds (user.shares> 0), this function will recalculate the user's cakeAtLastUserAction value. In the calculation process, the number of cake tokens obtained by the `balanceOf` function is used to participate in the calculation. But at the end of this function, a certain amount of cake tokens will be transferred to the user through the safeTransfer function, so the number of cake tokens obtained by the `balanceOf` function used in the calculation of cakeAtLastUserAction is relatively large.

Code location:

```
 if (user.shares > 0) {
            user.cakeAtLastUserAction =
user.shares.mul(balanceOf()).div(totalShares);
        } else {
            user.cakeAtLastUserAction = 0;
        }
```

```
        user.lastUserActionTime = block.timestamp;

        token.safeTransfer(msg.sender, currentAmount);
```

Solution

It is recommended to perform the safeTransfer operation first and then calculate the user's cakeAtLastUserAction value.

Status

Confirmed

[N3] [Suggestion] Missing event records

Category: Others

Content

In the contract, the owner role can set the addresses of the admin role and the treasure role through the setAdmin function and the setTreasury function, respectively, but no event recording is performed.

In the contract, the admin role can change the sensitive parameters of the contract through the setPerformanceFee, setCallFee, setWithdrawFee, and setWithdrawFeePeriod functions, but no event recording is performed.

Code location:

```
    function setAdmin(address _admin) external onlyOwner {
        require(_admin != address(0), "Cannot be zero address");
        admin = _admin;
    }

    function setTreasury(address _treasury) external onlyOwner {
        require(_treasury != address(0), "Cannot be zero address");
        treasury = _treasury;
    }

    function setPerformanceFee(uint256 _performanceFee) external onlyAdmin {
        require(_performanceFee <= MAX_PERFORMANCE_FEE, "performanceFee cannot be
more than MAX_PERFORMANCE_FEE");
        performanceFee = _performanceFee;
    }
```

```
    function setCallFee(uint256 _callFee) external onlyAdmin {
        require(_callFee <= MAX_CALL_FEE, "callFee cannot be more than
MAX_CALL_FEE");
        callFee = _callFee;
    }

    function setWithdrawFee(uint256 _withdrawFee) external onlyAdmin {
        require(_withdrawFee <= MAX_WITHDRAW_FEE, "withdrawFee cannot be more than
MAX_WITHDRAW_FEE");
        withdrawFee = _withdrawFee;
    }

    function setWithdrawFeePeriod(uint256 _withdrawFeePeriod) external onlyAdmin {
        require(
            _withdrawFeePeriod <= MAX_WITHDRAW_FEE_PERIOD,
            "withdrawFeePeriod cannot be more than MAX_WITHDRAW_FEE_PERIOD"
        );
        withdrawFeePeriod = _withdrawFeePeriod;
    }
```

Solution

It is recommended that event logging be performed when modifying sensitive parameters of the contract for follow-up self-examination or community review.

Status

Confirmed

[N4] [Low] Emergency withdrawal issue

Category: Design Logic Audit

Content

In the Vault contract, the admin role can make emergency withdrawals of cake tokens from the MasterChef contract to the Vault contract via the emergencyWithdraw function. However, it should be noted that any user can obtain 0.25% of the cake token reward in the Vault contract through the harvest function, and re-stake the remaining cake tokens into the MasterChef contract. So if the emergencyWithdraw operation is performed while the contract is not suspended it may cause unintended results.

Code location:

```
    function emergencyWithdraw() external onlyAdmin {
        IMasterChef(masterchef).emergencyWithdraw(0);
    }

    function harvest() external notContract whenNotPaused {
        IMasterChef(masterchef).leaveStaking(0);

        uint256 bal = available();
        uint256 currentPerformanceFee = bal.mul(performanceFee).div(10000);
        token.safeTransfer(treasury, currentPerformanceFee);

        uint256 currentCallFee = bal.mul(callFee).div(10000);
        token.safeTransfer(msg.sender, currentCallFee);

        _earn();

        lastHarvestedTime = block.timestamp;

        emit Harvest(msg.sender, currentPerformanceFee, currentCallFee);
    }
```

Solution

It is recommended to suspend the Vault contract before the emergencyWithdraw operation, or set the `_paused` parameter to true in the emergencyWithdraw function.

Status

Fixed; After communicating with the project party, the project party decided to deploy a wrapper contract that can call all admin only functions except emergencyWithdraw. The mainnet address of this contract is:

0xb6958D19b60e5fC85908C67c37a5b954E9D60d99

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0x002105260004	SlowMist Security Team	2021.05.21 - 2021.05.26	Passed

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 low risk, 3 suggestion vulnerabilities. And 3 suggestion vulnerabilities were confirmed and being fixed; All other findings were fixed.

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