Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.06.11, the SlowMist security team received the Alpaca Finance team's security audit application for Alpaca Finance Phase 3, 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

Leveraged yield farming on Binance Smart Chain.

Audit Files:

CakeMaxiWorker.sol:

https://github.com/alpaca-finance/bsc-alpaca-contract/blob/main/contracts/6/protocol/workers/CakeMaxiWorker.sol

commit: 20c4c545a0323b71d2969fd79db8316e60bc7d76

SingleAssetWorkerConfig.sol:

https://github.com/alpaca-finance/bsc-alpaca-contract/blob/main/contracts/6/protocol/workers/SingleAssetWorkerConfig.sol

commit: 20c4c545a0323b71d2969fd79db8316e60bc7d76

All strategies under this folder:

https://github.com/alpaca-finance/bsc-alpaca-contract/tree/main/contracts/6/protocol/strategies/pancakeswapV2-restricted-single-asset

commit: 20c4c545a0323b71d2969fd79db8316e60bc7d76

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Risk of manipulation of health parameters	Design Logic Audit	High	Discussed
N2	Slippage check missing issues	Design Logic Audit	Suggestion	Confirmed
N3	Price update interval issue	Design Logic Audit	Low	Fixed

4 Code Overview

4.1 Contracts Description

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:

CakeMaxiWorker

CakeMaxiWorker			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
shareToBalance	Public	-	-
balanceToShare	Public	-	-
reinvest	External	Can Modify State	onlyEOA onlyReinvestor nonReentrant
_rewardToBeneficialVault	Internal	Can Modify State	-
work	External	Can Modify State	onlyOperator nonReentrant
getMktSellAmount	Public	-	-
health	External	-	-
liquidate	External	Can Modify State	onlyOperator nonReentrant
actualFarmingTokenBalance	Internal	-	-
_addShare	Internal	Can Modify State	-
_removeShare	Internal	Can Modify State	-
getPath	External	-	-
getReversedPath	Public	-	-
getRewardPath	External	-	-
setReinvestBountyBps	External	Can Modify State	onlyOwner
setBeneficialVaultBountyBps	External	Can Modify State	onlyOwner

CakeMaxiWorker			
setMaxReinvestBountyBps	External	Can Modify State	onlyOwner
setStrategyOk	External	Can Modify State	onlyOwner
setReinvestorOk	External	Can Modify State	onlyOwner
setPath	External	Can Modify State	onlyOwner
setRewardPath	External	Can Modify State	onlyOwner
setCriticalStrategies	External	Can Modify State	onlyOwner

SingleAssetWorkerConfig			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
setOracle	External	Can Modify State	onlyOwner
setConfigs	External	Can Modify State	onlyOwner
isStable	Public	-	-
acceptDebt	External	-	-
workFactor	External	-	-
killFactor	External	-	-
setGovernor	External	Can Modify State	onlyOwner
emergencySetAcceptDebt	External	Can Modify State	onlyGovernor

PancakeswapV2RestrictedSingleAssetStrategyAddBaseTokenOnly

PancakeswapV2RestrictedSingleAssetStrategyAddBaseTokenOnly			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
execute	External	Can Modify State	onlyWhitelistedWorkers nonReentrant
setWorkersOk	External	Can Modify State	onlyOwner

PancakeswapV2RestrictedSingleAssetStrategyAddBaseWithFarm			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
execute	External	Can Modify State	onlyWhitelistedWorkers nonReentrant
setWorkersOk	External	Can Modify State	onlyOwner

PancakeswapV2RestrictedSingleAssetStrategyLiquidate			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
execute	External	Can Modify State	onlyWhitelistedWorkers nonReentrant
setWorkersOk	External	Can Modify State	onlyOwner

PancakeswapV2RestrictedSingleAssetStrategyWithdrawMinimizeTrading			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
execute	External	Can Modify State	onlyWhitelistedWorkers nonReentrant

PancakeswapV2RestrictedSingleAssetStrategyWithdrawMinimizeTrading			
setWorkersOk	External	Can Modify State	onlyOwner
<Receive Ether>	External	Payable	-

4.3 Vulnerability Summary

[N1] [High] Risk of manipulation of health parameters

Category: Design Logic Audit

Content

In the CakeMaxiWorker contract, the health function will be used to check the health of the position of the specified id. But it is calculated by obtaining the number of tokens in the pool through the getReserves interface of the pair contract. The getReserves interface obtains the real-time number of tokens in the pool, which means it can be manipulated.

Code location:

```
  function health(uint256 id) external override view returns (uint256) {
    IPancakePair currentLP;
    uint256[] memory amount;
    address[] memory reversedPath = getReversedPath();
    amount = new uint256[](reversedPath.length);
    amount[0] = shareToBalance(shares[id]);
    for(uint256 i = 1; i < reversedPath.length; i++) {
        /// 1. Get the current LP based on the specified paths.
        currentLP = IPancakePair(factory.getPair(reversedPath[i-1],
reversedPath[i]));
        /// 2. Get the pool's total supply of the token of path i-1 and the token of
path i.
        (uint256 r0, uint256 r1,) = currentLP.getReserves();
        (uint256 rOut, uint256 rIn) = currentLP.token0() == reversedPath[i] ? (r0,
r1) : (r1, r0);
        /// 3. Convert all amount on the token of path i-1 to the token of path i.
        amount[i] = getMktSellAmount(
            amount[i-1], rIn, rOut
```

```
        );
    }
    /// @notice return the last amount, since the last amount is the amount that we
shall get in baseToken if we sell the farmingToken at the market price
    return amount[amount.length - 1];
  }
```

Solution

It is recommended to use an oracle with delayed price feed for calculation.

Status

Discussed; In the vault contract, it will check the price through the isStable of the SingleAssetWorkerConfig contract, where the price is fed by the oracle.

[N2] [Suggestion] Slippage check missing issues

Category: Design Logic Audit

Content

- In the CakeMaxiWorker contract, the `_rewardToBeneficialVault` function is used to convert the farmingToken into a beneficialVaultToken, but the swapExactTokensForTokens function of the router contract is called for the Swap operation without slippage check, which will lead to a possible sandwich attack during the swap process.
- In the CakeMaxiWorker contract, the Operator role can perform a liquidating operation on the specified id via the liquidate function and exchange farmingToken to baseToken. but the slippage is not checked when performing the exchange operation, which would be at risk of a sandwich attack.

Code location:

```
  function _rewardToBeneficialVault(uint256 _beneficialVaultBounty, address
_rewardToken) internal {
    /// 1. approve router to do the trading
    _rewardToken.safeApprove(address(router), uint256(-1));
    /// 2. read base token from beneficialVault
    address beneficialVaultToken = beneficialVault.token();
```

```
    /// 3. swap reward token to beneficialVaultToken
    uint256[] memory amounts =
router.swapExactTokensForTokens(_beneficialVaultBounty, 0, rewardPath, address(this),
now);
    beneficialVaultToken.safeTransfer(address(beneficialVault),
beneficialVaultToken.myBalance());
    _rewardToken.safeApprove(address(router), 0);

    emit BeneficialVaultTokenBuyback(_msgSender(), beneficialVault,
amounts[amounts.length - 1]);
  }
```

```
  function liquidate(uint256 id) external override onlyOperator nonReentrant {
    // 1. Remove shares on this position back to farming tokens
    _removeShare(id);
    farmingToken.safeTransfer(address(liqStrat), actualFarmingTokenBalance());
    liqStrat.execute(address(0), 0, abi.encode(0));
    // 2. Return all available base token back to the operator.
    uint256 wad = baseToken.myBalance();
    baseToken.safeTransfer(_msgSender(), wad);
    emit Liquidate(id, wad);
  }
```

Solution

It is recommended to check slippage when performing swap operations.

Status

Confirmed; After communicating with the project party, the project party stated that the amount of funds in each transaction is not large when the token swap operation is performed through the `_rewardToBeneficialVault` function, so this risk is within an acceptable range. For liquidate operations, liquidation is required regardless of the price of the tokens, so this is an expected design.

[N3] [Low] Price update interval issue

Category: Design Logic Audit

Content

In the SingleAssetWorkerConfig contract, the isStable function will get the price from oracle and check whether the

price is stable. But its allowed price update interval is 1 day. There will be a situation, if the price fluctuates sharply in a short period of time (rapidly rising or falling), and oracle does not update the price in time due to some circumstances, then the check on lpPrice and price will fail. But at this time, the user can change the value of lpPrice by manipulating the number of tokens in the pool to pass the check of lpPrice and price.

Note: After passing the complete Swap operation, the pair will update the reserves and balances of the pool at the same time through the update function, so the following checks can still be passed after the number of tokens in the pool is controlled by the complete Swap, because `tbal` and `r` always increase and decrease at the same time of.

```
        require(t0bal.mul(100) <= r0.mul(101), "SingleAssetWorkerConfig::isStable::
bad t0 balance");
        require(t1bal.mul(100) <= r1.mul(101), "SingleAssetWorkerConfig::isStable::
bad t1 balance");
```

Code location:

```
  function isStable(address _worker) public view returns (bool) {
    IWorker02 worker = IWorker02(_worker);
    address[] memory path = worker.getPath();
    // @notice loop over the path for validating the price of each pair
    IPancakePair currentLP;
    uint256 maxPriceDiff = workers[_worker].maxPriceDiff;
    for(uint256 i = 1; i < path.length; i++) {
        // 1. Get the position's LP balance and LP total supply.
        currentLP = IPancakePair(factory.getPair(path[i-1], path[i]));
        address token0 = currentLP.token0();
        address token1 = currentLP.token1();
        // 2. Check that reserves and balances are consistent (within 1%)
        (uint256 r0, uint256 r1,) = currentLP.getReserves();
        uint256 t0bal = token0.balanceOf(address(currentLP));
        uint256 t1bal = token1.balanceOf(address(currentLP));
        require(t0bal.mul(100) <= r0.mul(101), "SingleAssetWorkerConfig::isStable::
bad t0 balance");
        require(t1bal.mul(100) <= r1.mul(101), "SingleAssetWorkerConfig::isStable::
bad t1 balance");
        // 3. Check that price is in the acceptable range
        (uint256 price, uint256 lastUpdate) = oracle.getPrice(token0, token1);
        require(lastUpdate >= now - 1 days, "SingleAssetWorkerConfig::isStable::
```

```
price too stale");
        uint256 spotPrice = r1.mul(1e18).div(r0);
        require(spotPrice.mul(10000) <= price.mul(maxPriceDiff),
"SingleAssetWorkerConfig::isStable:: price too high");
        require(spotPrice.mul(maxPriceDiff) >= price.mul(10000),
"SingleAssetWorkerConfig::isStable:: price too low");
    }
    return true;
  }
```

Solution

It is recommended to shorten the price update interval.

Status

Fixed; After communicating with the project party, the project party stated that the price feed interval will be reduced when the market fluctuates sharply in a short period of time.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002106110005	SlowMist Security Team	2021.06.11 - 2021.06.11	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 high risk, 1 low risk, 1 suggestion vulnerabilities. And 1 high risk, 1 suggestion vulnerabilities were Discussed; All other findings were fixed. 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