Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.08.25, the SlowMist security team received the Jswap Finance team's security audit application for Jswap Finance Phase 2, 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

Jswap (Jswap.Finance) is a decentralized transaction and wealth management protocol based on OKExChain. It supports swap mining, liquidity mining, DAO dividends, single token liquidity mining and other features.

Audit version：

jswap_auth.zip:

SHA256: 7f61424363897918c4d34345d4851f6c6bc157c6351d87248f4c1b743aaa42b3

Audit scope：

contracts/cherry/JfCheVault.sol

contracts/cherry/SingleSmartChef.sol

contracts/okchain/SingleMiningPoolV2.sol

contracts/strategy/CherrySingleToCheStrategy.sol

Fixed version：

jswap_auth.zip:

SHA256: 7826f7fa1301b92bf8e785282ff18a9735539845b15c6fc6c9ddca69bb8b6cef

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Compatibility issue	Others	Suggestion	Ignored
N2	Balance Check Deficiency	Design Logic Audit	Low	Ignored
N3	Slippage check issue	Design Logic Audit	Medium	Ignored
N4	Strategy setting issue	Design Logic Audit	Suggestion	Confirmed
N5	Redundant code	Others	Suggestion	Ignored
N6	Repeated initialization issue	Design Logic Audit	Low	Fixed
N7	Exit strategy pool issue	Design Logic Audit	Low	Ignored
N8	Emergency exit issue	Design Logic Audit	Low	Ignored

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

Contract Name	Contract Address (OkExChain Mainnet)
JfCheVault	0x6CDa71169cE699a5ba231eb701C9C49C4b988a52
SingleMiningPoolV2 Proxy	0x4e864E36Bb552BD1Bf7bcB71A25d8c96536Af7e3
SingleMiningPoolV2 Implementation	0xF31E8D18c9f7Ed196EFB4f9A260a45dEbcB3b1b6
che2cheStrategy Proxy	0xEeBcFeaF6Ca1B6267dD946E21D6cC45D5fEd49CB
che2cheStrategy Implementation	0xb7FB85fd6350d2d6bc685Fe3bFE01294F02b6b5b
ethk2cheStrategy Proxy	0x13140F54e07de022569046c82a71eEf75D732953
ethk2cheStrategy Implementation	0xA08D8Ec2bb475439A149A4399e2BFD81f6FeBB18
btck2cheStrategy Proxy	0x6Aee9EBBe4C1E54e2ad4e5b12CB73FA296505Ec0
btck2cheStrategy Implementation	0x9f33a910965E1507E9DF1e6a9c5F605763b2cE94
usdt2cheStrategy Proxy	0xC7924ab0308dB27137a4669cec6F1d3BA13fD551
usdt2cheStrategy Implementation	0x23dA1d260Fde1E685Cb4d9a08512C17F7300B73b

4.2 Visibility Description

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

SingleSmartChef			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
stopReward	Public	Can Modify State	onlyOwner
getMultiplier	Public	-	-
updateMultiplier	Public	Can Modify State	onlyOwner

SingleSmartChef			
pendingReward	External	-	-
updatePool	Public	Can Modify State	-
massUpdatePools	Public	Can Modify State	-
deposit	Public	Can Modify State	-
withdraw	Public	Can Modify State	-
emergencyWithdraw	Public	Can Modify State	-
emergencyRewardWithdraw	Public	Can Modify State	onlyOwner

JfCheVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
che2Jf	Public	-	-
price12	Public	-	-
swapCHE2JF	External	Can Modify State	-

SingleMiningPoolV2			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
poolLength	External	-	-
setPoolScale	External	Can Modify State	-
add	Public	Can Modify State	onlyOwner

SingleMiningPoolV2			
set	Public	Can Modify State	onlyOwner
setJfPerBlock	External	Can Modify State	onlyOwner
setMigrator	Public	Can Modify State	onlyOwner
setTimelock	External	Can Modify State	-
emergencyExitStrategy	External	Can Modify State	onlyOwner
setStrategy	Public	Can Modify State	-
migrate	Public	Can Modify State	-
getMultiplier	Public	-	-
pendingJf	External	-	-
claim	External	Can Modify State	-
claimAll	External	Can Modify State	-
_claimJf	Private	Can Modify State	-
massUpdatePools	Public	Can Modify State	-
totalSupply	Public	-	-
updatePool	Public	Can Modify State	-
deposit	Public	Payable	-
withdraw	Public	Can Modify State	-
emergencyWithdraw	Public	Can Modify State	-
safeJfTransfer	Internal	Can Modify State	-
dev	Public	Can Modify State	onlyOwner

SingleMiningPoolV2			
safeTransferFrom	Internal	Can Modify State	-
safeTransferETH	Internal	Can Modify State	-
safeTransfer	Internal	Can Modify State	-
<Receive Ether>	External	Payable	-

CherrySingleToCheStrategy			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
freeze	External	Can Modify State	onlyOwner
unfreeze	External	Can Modify State	onlyOwner
setJfCheVault	External	Can Modify State	onlyOwner
approveStakeAndCompoundPool	Private	Can Modify State	-
initPool	External	Can Modify State	onlyJfPool
exit	External	Can Modify State	onlyJfPool
deposit	External	Can Modify State	onlyJfPool updateInvest
withdraw	External	Can Modify State	onlyJfPool updateInvest
harvest	External	Can Modify State	onlyJfPool updateInvest
compound	Public	Can Modify State	-
reinvest	Private	Can Modify State	-
updatePool	Private	Can Modify State	-

CherrySingleToCheStrategy			
withdrawReward	Private	Can Modify State	-
pendingJf	External	-	-
withdrawCheToUser	Private	Can Modify State	-
min	Private	-	-
withdrawStakeTokenToUser	Private	Can Modify State	-
safeTransferETH	Internal	Can Modify State	-
<Receive Ether>	External	Payable	-

4.3 Vulnerability Summary

[N1] [Suggestion] Compatibility issue

Category: Others

Content

In the SingleSmartChef contract, the user can use the deposit function to perform stake operations. The amount of tokens needed to be staked by the user `_amount` will be transferred into the contract and recorded in user.amount at the same time, but if the stake is deflation tokens, the actual amount of collateral received by the contract is different from the collateral amount `_amount` entered by the user.

Code location：

```
    function deposit(uint256 _amount) public {
        console.log("Syrup.deposit", msg.sender);
        PoolInfo storage pool = poolInfo[0];
        UserInfo storage user = userInfo[msg.sender];
        updatePool(0);
        if (user.amount > 0) {
            uint256 pending =
user.amount.mul(pool.accChePerShare).div(1e12).sub(user.rewardDebt);
```

```
            if(pending > 0) {
                console.log("Syrup.balance", rewardToken.balanceOf(address(this)),
pending );
                console.log("Syrup.rewardToken", address(rewardToken));
                rewardToken.safeTransfer(address(msg.sender), pending);
            }
        }
        if(_amount > 0) {
            pool.lpToken.safeTransferFrom(address(msg.sender), address(this),
_amount);
            user.amount = user.amount.add(_amount);
            totalDeposit=totalDeposit.add(_amount);
        }
        user.rewardDebt = user.amount.mul(pool.accChePerShare).div(1e12);

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

Solution

It is recommended to check the stake token balance of the contract before and after the transfer, and use the before and after difference as the actual stake amount.

Status

Ignored; After communicating with the project party, the project party stated that this contract is an external project party contract and is only used for testing.

[N2] [Low] Balance Check Deficiency

Category: Design Logic Audit

Content

In the SingleSmartChef contract, the owner can withdraw the reward tokens in the contract through the emergencyRewardWithdraw function, and it will check that the amount withdraw is less than the number of tokens in the contract. This will cause the reward tokens in the contract to never be fully withdraw.

Code location：

```
    function emergencyRewardWithdraw(uint256 _amount) public onlyOwner {
        require(_amount < rewardToken.balanceOf(address(this)), 'not enough token');
        rewardToken.safeTransfer(address(msg.sender), _amount);
    }
```

Solution

It is recommended to change the `<` symbol of the balance check to `<=`

Status

Ignored; After communicating with the project party, the project party stated that this is an external project token and is only used for testing.

[N3] [Medium] Slippage check issue

Category: Design Logic Audit

Content

In the JfCheVault contract, the swapCHE2JF function is used to swap CHE tokens into JF tokens, and the swap process will be realized through DEX. However, the slippage was not checked during the swap process. There will be risks such as a sandwich attack.

Code location：

```
    function swapCHE2JF(uint256 _cheAmount, address _to ) external returns (uint256)
{
        address[] memory CHE_USDT_PATH = new address[](2);
        CHE_USDT_PATH[0] = cheToken;
        CHE_USDT_PATH[1] = bridgeToken;
        IERC20(cheToken).safeApprove(address(cheRouter), _cheAmount);
        cheRouter.swapExactTokensForTokens(
                _cheAmount,
                0,
                CHE_USDT_PATH,
                address(this),
                block.timestamp + 100
        );
        address[] memory USDT_JF_PATH = new address[](2);
        USDT_JF_PATH[0] = bridgeToken;
        USDT_JF_PATH[1] = jfToken;
        uint256 bridgeAmount = IERC20(bridgeToken).balanceOf(address(this));
        IERC20(bridgeToken).safeApprove(address(jfRouter), bridgeAmount);
        uint256[] memory amountOut = jfRouter.swapExactTokensForTokens(
                bridgeAmount,
                0,
                USDT_JF_PATH,
                _to,
                block.timestamp + 100
        );
        return amountOut[1];
    }
```

Solution

It is recommended to check slippage during the swap process.

Status

Ignored; After communicating with the project party, the project party stated that this function will be used frequently and will not lead to the scene of large capital swap, so the slippage issue is ignored.

[N4] [Suggestion] Strategy setting issue

Category: Design Logic Audit

Content

In the SingleMiningPoolV2 contract, the timelock role can set the strategy pool address through the setStrategy function, but the expected design is that the strategy pool cannot be set for deflationary tokens, but the setStrategy function does not check whether `pool.lpToken` is a deflationary token.

Code location：

```
    function setStrategy(uint256 _pid, address _strategy) public {
        require(msg.sender == timeLock, "SetStrategy only timeLock allowed");

        PoolInfo storage pool = poolInfo[_pid];
        IERC20 lpToken = pool.lpToken;

        //First init migrate Asset to new Stategy
        if(poolStrategy[_pid] == address(0)) {
            poolStrategy[_pid] = _strategy;
            if(address(pool.lpToken) == WETH) {
                //wrapped by WETH
                IWETH(WETH).deposit{value: address(this).balance}();
            }
            lpToken.safeTransfer(_strategy, lpToken.balanceOf(address(this)));
            IBaseStrategy(_strategy).initPool();
        } else {
            address _assetReceive = _strategy;
            if(_strategy == address(0)) {
                require(address(pool.lpToken) != WETH, "Strategy for WETH cannot
migrate to address(0)");
                _assetReceive = address(this);
            }
            address preStrategy = poolStrategy[_pid];
            // withdraw StakeAssets & EarnAssets
            IBaseStrategy(preStrategy).exit(_assetReceive);
            IBaseStrategy(_strategy).initPool();
            poolStrategy[_pid] = _strategy;
        }
    }
```

Solution

It is recommended to check whether the staking tokens are deflationary tokens when setting up the strategy pool.

Status

Confirmed

[N5] [Suggestion] Redundant code

Category: Others

Content

In the SingleMiningPoolV2 contract, there is no specific implementation of the migrate function.

Code location:

```
    function migrate(uint256 _pid) public {
        // require(address(migrator) != address(0), "migrate: no migrator");
        // PoolInfo storage pool = poolInfo[_pid];
        // IERC20 lpToken = pool.lpToken;
        // uint256 bal = lpToken.balanceOf(address(this));
        // lpToken.safeApprove(address(migrator), bal);
        // IERC20 newLpToken = migrator.migrate(lpToken);
        // require(bal == newLpToken.balanceOf(address(this)), "migrate: bad");
        // pool.lpToken = newLpToken;
    }

    function setMigrator(IMigratorChef _migrator) public onlyOwner {
        migrator = _migrator;
    }
```

Solution

It is recommended to remove redundant codes.

Status

Ignored

[N6] [Low] Repeated initialization issue

Category: Design Logic Audit

Content

In the CherrySingleToCheStrategy contract, JfPool can call the initPool function to initialize the pool when setting the strategy pool, but it does not limit repeated initialization. After the strategy pool is abandoned, the data in the contract still exists. If it is re-enabled and initialized in the future, this will cause data conflicts.

Code location：

```
    function initPool() external onlyJfPool  {
        uint256 initAmount = stakeToken.balanceOf(address(this));
        if(initAmount > 0) {
            singleStakePool.deposit(initAmount);
            totalDeposit = initAmount;
        }
    }
```

Solution

It is recommended to not call the initPool function repeatedly.

Status

Fixed

[N7] [Low] Exit strategy pool issue

Category: Design Logic Audit

Content

In the CherrySingleToCheStrategy contract, the JfPool contract can exit the strategy pool through the exit function, but the totalDeposit parameter is not set to 0 when exiting.

Code location：

```
    function exit(address _assetsTo) external onlyJfPool {
        //withdraw all
        //1. withdraw All rewards
        withdrawReward();
        //2. emergencyWithdraw all Stake Token
        singleCompoundPool.emergencyWithdraw();
        singleStakePool.emergencyWithdraw();
        //3. transfer to _assetsTo
        uint256 harvestAmount = harvestToken.balanceOf(address(this));

        if(harvestAmount > 0 ) {
            harvestToken.transfer(_assetsTo, harvestAmount);
        }
        uint256 stakeAmount = stakeToken.balanceOf(address(this));
        if(stakeAmount > 0) {
            stakeToken.transfer(_assetsTo, stakeAmount);
        }
    }
```

Solution

It is recommended to set totalDeposit to 0 when exiting.

Status

Ignored; After communicating with the project party, the project party stated that after it withdraws from the strategy contract, the contract will be abandoned and no longer used.

[N8] [Low] Emergency exit issue

Category: Design Logic Audit

Content

In the SingleMiningPoolV2 contract, the owner can call the emergencyExitStrategy function to exit the stake urgently from the strategy pool. When exiting, the strategy pool will send the staking tokens and harvest tokens to the SingleMiningPoolV2 contract. However, in this contract, the harvested tokens cannot be exchanged and withdraw, which will cause the harvested tokens to be locked in the SingleMiningPoolV2 contract.

Code location:

```
    function emergencyExitStrategy(uint256 _pid) external onlyOwner {
        address preStrategy = poolStrategy[_pid];
        require(preStrategy != address(0), "No Strategy");

        PoolInfo storage pool = poolInfo[_pid];
        require(address(pool.lpToken) != WETH, "WETH strategy cannot be address(0),
pause in Strategy");

        IBaseStrategy(preStrategy).exit(address(this));
        poolStrategy[_pid] = address(0);
    }

    function exit(address _assetsTo) external onlyJfPool {
        //withdraw all
        //1. withdraw All rewards
        withdrawReward();
        //2. emergencyWithdraw all Stake Token
        singleCompoundPool.emergencyWithdraw();
        singleStakePool.emergencyWithdraw();
        //3. transfer to _assetsTo
        uint256 harvestAmount = harvestToken.balanceOf(address(this));

        if(harvestAmount > 0 ) {
            harvestToken.transfer(_assetsTo, harvestAmount);
        }
        uint256 stakeAmount = stakeToken.balanceOf(address(this));
        if(stakeAmount > 0) {
            stakeToken.transfer(_assetsTo, stakeAmount);
        }
    }
```

Solution

If it is not the expected design, it is recommended to deal with the emergency withdrawal of the harvested tokens.

Status

Ignored; After communicating with the project party, the project party stated that this function is only used in emergency situations, and will give priority to guaranteeing the security of staking funds in emergency situations, so the reward will not be processed.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002109080003	SlowMist Security Team	2021.08.25 - 2021.08.27	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 medium risk, 4 low risk, 3 suggestion vulnerabilities. And 1 suggestion vulnerabilities were confirmed and being fixed; 1 medium risk, 3 low risk, 2 suggestion vulnerabilities were ignored; 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.

SLOWMIST

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist