Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.06.01, the SlowMist security team received the Jswap Finance team's security audit application for Jswap Finance, 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: 5e2f546a6a9eb857a978bb7de175d3cfc244b46bce5edf4ed104b51ef27505f0；

Fixed version:

jswap_auth.zip

SHA256: 0ae96f7a3923a5fb4e04df29f2c1c0d6bb52d007d50378c39e2465b67fded87d；

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Risk of excessive authority	Authority Control Vulnerability	Medium	Confirmed
N2	Access control issue	Authority Control Vulnerability	Suggestion	Fixed
N3	Slippage check issue	Design Logic Audit	Suggestion	Ignored
N4	The change of LP pool weights affects users' income	Others	Suggestion	Fixed
N5	Block reward changes lead to revenue change issue	Others	Suggestion	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:

JswapFactory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
allPairsLength	External	-	-

JswapFactory			
pairCodeHash	External	-	-
sortTokens	Public	-	-
pairFor	Public	-	-
createPair	External	Can Modify State	-
getSalt	Public	-	-
setFeeTo	External	Can Modify State	-
setMigrator	External	Can Modify State	-
setFeeToSetter	External	Can Modify State	-

JswapERC20			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_mint	Internal	Can Modify State	-
_burn	Internal	Can Modify State	-
_approve	Private	Can Modify State	-
_transfer	Private	Can Modify State	-
approve	External	Can Modify State	-
transfer	External	Can Modify State	-
transferFrom	External	Can Modify State	-
permit	External	Can Modify State	-

JswapPair			
Function Name	Visibility	Mutability	Modifiers
getReserves	Public	-	-
_safeTransfer	Private	Can Modify State	-
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	-
_update	Private	Can Modify State	-
_mintFee	Private	Can Modify State	-
mint	External	Can Modify State	lock
burn	External	Can Modify State	lock
swap	External	Can Modify State	lock
skim	External	Can Modify State	lock
sync	External	Can Modify State	lock

JswapRouter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setSwapMining	Public	Can Modify State	onlyOwner
<Receive Ether>	External	Payable	-
_addLiquidity	Internal	Can Modify State	-
addLiquidity	External	Can Modify State	ensure

JswapRouter			
addLiquidityETH	External	Payable	ensure
removeLiquidity	Public	Can Modify State	ensure
removeLiquidityETH	Public	Can Modify State	ensure
removeLiquidityWithPermit	External	Can Modify State	-
removeLiquidityETHWithPermit	External	Can Modify State	-
removeLiquidityETHSupportingFeeOnTransferTokens	Public	Can Modify State	ensure
removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	Can Modify State	-
_swap	Internal	Can Modify State	-
swapExactTokensForTokens	External	Can Modify State	ensure
swapTokensForExactTokens	External	Can Modify State	ensure
swapExactETHForTokens	External	Payable	ensure
swapTokensForExactETH	External	Can Modify State	ensure
swapExactTokensForETH	External	Can Modify State	ensure
swapETHForExactTokens	External	Payable	ensure
_swapSupportingFeeOnTransferTokens	Internal	Can Modify State	-
swapExactTokensForTokensSupportingFeeOnTransferTokens	External	Can Modify State	ensure
swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	ensure

JswapRouter			
swapExactTokensForETHSupportingFeeOnTransferTokens	External	Can Modify State	ensure
quote	Public	-	-
getAmountOut	Public	-	-
getAmountIn	Public	-	-
getAmountsOut	Public	-	-
getAmountsIn	Public	-	-

JfBar			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
poolLength	External	-	-
add	Private	Can Modify State	-
setDestroyRate	External	Can Modify State	onlyOwner
setMakerAddr	External	Can Modify State	onlyOwner
pendingJf	External	-	-
claim	External	Can Modify State	-
claimAll	External	Can Modify State	-
_claimJf	Private	Can Modify State	-
updatePool	Private	Can Modify State	-
setStakeReferWeight	External	Can Modify State	onlyOwner

JfBar			
appendReward	External	Can Modify State	-
depositWithCode	External	Can Modify State	-
generateCode	Private	Can Modify State	-
codeCheckCalc	Private	Can Modify State	-
sendPrefix	Private	-	-
deposit	Public	Can Modify State	-
_deposit	Private	Can Modify State	-
withdraw	External	Can Modify State	-
_withdraw	Private	Can Modify State	-
safeJfTransfer	Internal	Can Modify State	-
safeTransferFrom	Internal	Can Modify State	-
safeTransfer	Internal	Can Modify State	-

JfMaker			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
bridgeFor	Public	-	-
setBridge	External	Can Modify State	onlyOwner
convert	External	Can Modify State	onlyEOA
convertMultiple	External	Can Modify State	onlyEOA

JfMaker			
_convert	Internal	Can Modify State	-
_convertStep	Internal	Can Modify State	-
_swap	Internal	Can Modify State	-
_toJF	Internal	Can Modify State	-
transferToBar	Private	Can Modify State	-

JfOracle			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	PriceOracle
setStableToken	Public	Can Modify State	onlyOwner
convert2JF	External	Can Modify State	-
convert2ValuatToken	Private	Can Modify State	-
_consult	Public	Can Modify State	-
addRouter	Public	Can Modify State	onlyOwner
delRouter	Public	Can Modify State	onlyOwner
getRouter	Public	-	-
getRouterlength	Public	-	-

PriceOracle			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-

PriceOracle			
update	Public	Can Modify State	-
computeAmountOut	Private	-	-
consult	Public	Can Modify State	-

MiningPool			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
poolLength	External	-	-
add	Public	Can Modify State	onlyOwner
set	Public	Can Modify State	onlyOwner
setJfPerBlock	External	Can Modify State	onlyOwner
setMigrator	Public	Can Modify State	onlyOwner
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	-
updatePool	Public	Can Modify State	-

MiningPool			
deposit	Public	Can Modify State	-
withdraw	Public	Can Modify State	-
emergencyWithdraw	Public	Can Modify State	-
safeJfTransfer	Internal	Can Modify State	-
dev	Public	Can Modify State	-
safeTransferFrom	Internal	Can Modify State	-
safeTransfer	Internal	Can Modify State	-

SwapMining			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
add	Public	Can Modify State	onlyOwner
setMintRate	External	Can Modify State	onlyOwner
setMintable	External	Can Modify State	onlyOwner
setMintMode	External	Can Modify State	onlyOwner
setRouter	Public	Can Modify State	onlyOwner
withdraw	Public	Can Modify State	-
userPoolPending	Public	-	-
swap	External	Can Modify State	onlyRouter
miningable	Public	-	-

SwapMining			
getPairTVL	Public	-	-
flatAmount	Public	-	-
pending	Public	-	-
poolLength	Public	-	-
poolInfo	Public	-	-
dev	Public	Can Modify State	-

JFToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20
mint	Public	Can Modify State	onlyMinter
addMinter	Public	Can Modify State	onlyOwner
delMinter	Public	Can Modify State	onlyOwner
getMinterLength	Public	-	-
isMinter	Public	-	-
getMinter	Public	-	onlyOwner

4.3 Vulnerability Summary

[N1] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

- In the JswapFactory contract, the feeToSetter role can modify the feeTo role, migrator contract address, and feeToSetter role address through the setFeeTo, setMigrator, and setFeeToSetter functions. This will lead to the risk of excessive feeToSetter role permissions.
- There is a migrate function in the MiningPool contract. The owner can specify the migrator address through the setMigrator function, and transfer the assets staked in the contract to the migrator address by calling the migrate function. This will cause the owner authority to be too large.
- In the JFToken contract, the owner can set any address to the minter role through the addMinter function. The minter role can mint tokens arbitrarily through the mint function. This will lead to the risk of excessive owner permissions.

Code location：

```
function setFeeTo(address _feeTo) external override {
    require(msg.sender == feeToSetter, 'Jswap: FORBIDDEN');
    feeTo = _feeTo;
}

function setMigrator(address _migrator) external override {
    require(msg.sender == feeToSetter, 'Jswap: FORBIDDEN');
    migrator = _migrator;
}

function setFeeToSetter(address _feeToSetter) external override {
    require(msg.sender == feeToSetter, 'Jswap: FORBIDDEN');
    feeToSetter = _feeToSetter;
}
```

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

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 mint(address _to, uint256 _amount) public onlyMinter returns (bool) {
        /*
        if (_amount.add(totalSupply()) > maxSupply) {
            return false;
        }
        */
        _mint(_to, _amount);
        return true;
    }

    function addMinter(address _addMinter) public onlyOwner returns (bool) {
        require(_addMinter != address(0), "MdxToken: _addMinter is the zero
address");
        return EnumerableSet.add(_minters, _addMinter);
    }
```

Solution

- It is recommended to transfer the feeToSetter role of the JswapFactory contract to community governance, or use the timelcok contract.
- Transfer the owner authority of the MiningPool contract to community governance or replace it with the timelock contract.
- Transfer the owner authority of the JFToken contract to community governance or replace it with the timelock contract.

Status

Confirmed; After communicating with the project party, the project party stated that it will transfer the authority to the timelock contract after the mainnet is officially deployed.

[N2] [Suggestion] Access control issue

Category: Authority Control Vulnerability

Content

There is an appendReward function in the JfBar contract to increase the number of JF token rewards in the Bar. The business logic can be added by Maker, but in actual implementation, any user can call this function to add JF token rewards.

Code location：

```
    function appendReward(uint256 amount) external  {
        //70% pool0
        uint256 stakePoolAmount = amount.mul(stakeWeight).div(100);
        updatePool(0, stakePoolAmount);
        //10% pool1
        uint256 referralPoolAmount = amount.mul(referralWeight).div(100);
        updatePool(1, referralPoolAmount);

        //20% destory
        safeTransfer(address(jf), destroyaddr,
amount.sub(stakePoolAmount).sub(referralPoolAmount));
        totalReardJF = totalReardJF.add(amount);
    }
```

Solution

If it is not designed as expected, it is recommended to perform permission control on the call of this function.

Status

Fixed

[N3] [Suggestion] Slippage check issue

Category: Design Logic Audit

Content

There is a `_swap` function in the JfMaker contract, which is used for token exchange. During the redemption process, the amount that should be accepted for redemption (amountOut) is calculated through the getReserves interface of the pair, and used as a slippage check into the swap function of the pair for redemption operations. But

the getReserves interface obtains the real-time number of tokens in the pool, which can be manipulated.

Although it is restricted that only EOA accounts can be called, there is still a risk of being attacked by sandwiches.

Code location：

```
    function _swap(
        address fromToken,
        address toToken,
        uint256 amountIn,
        address to
    ) internal returns (uint256 amountOut) {
        // Checks
        // X1 - X5: OK
        IJswapPair pair = IJswapPair(factory.getPair(fromToken, toToken));
        require(address(pair) != address(0), "JfMaker: Cannot convert");

        if(toToken == jf) {
            to = address(this);
        }
        // Interactions
        // X1 - X5: OK
        (uint256 reserve0, uint256 reserve1, ) = pair.getReserves();
        uint256 amountInWithFee = amountIn.mul(997);
        if (fromToken == pair.token0()) {
            amountOut = amountIn.mul(997).mul(reserve1) /
reserve0.mul(1000).add(amountInWithFee);
            IERC20(fromToken).safeTransfer(address(pair), amountIn);
            pair.swap(0, amountOut, to, new bytes(0));
            // TODO: Add maximum slippage?
        } else {
            amountOut = amountIn.mul(997).mul(reserve0) /
reserve1.mul(1000).add(amountInWithFee);
            IERC20(fromToken).safeTransfer(address(pair), amountIn);
            pair.swap(amountOut, 0, to, new bytes(0));
            // TODO: Add maximum slippage?
        }
        if(toToken == jf) {
            transferToBar(IERC20(jf).balanceOf(address(this)));
        }
    }
```

Solution

It is recommended to use a trusted oracle machine to check slippage.

Status

Ignored; After communicating with the project party, the project party stated that this function is executed frequently and the LP amount accumulated is small, so it is ignored.

[N4] [Suggestion] The change of LP pool weights affects users' income

Category: Others

Content

In the MiningPool contract, when the Owner calls the add function and the set function to add a new pool or reset the weight of the pool, all LP pool weights will change accordingly. The Owner can update all pools before adjusting the weight by passing in the `_withUpdate` parameter with a value of true to ensure that the user's income before the pool weight is changed will not be affected by the adjustment of the pool weight, but if the value of false is passed in the `_withUpdate` parameter, then all pools will not be updated before the pool weight is adjusted, which will cause the user's income to be affected before the pool weight is changed.

Code location：

```
    function add(uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate) public
onlyOwner {
        require( poolIndex[address(_lpToken)] == 0 , "Can not add repeated");

        if (_withUpdate) {
            massUpdatePools();
        }
        uint256 lastRewardBlock = block.number > startBlock ? block.number :
startBlock;
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(PoolInfo({
            lpToken: _lpToken,
            allocPoint: _allocPoint,
            lastRewardBlock: lastRewardBlock,
            accJfPerShare: 0
        }));
```

```
        poolIndex[address(_lpToken)] =  poolInfo.length;
    }

    function set(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public
onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        totalAllocPoint =
totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
        poolInfo[_pid].allocPoint = _allocPoint;
    }
```

Solution

It is recommended to force all pools to be updated before adjusting the weights of LP pools to avoid the impact of user income.

Status

Fixed

[N5] [Suggestion] Block reward changes lead to revenue change issue

Category: Others

Content

In the MiningPool contract, the Owner can change the number of JF token rewards in each block through the setJfPerBlock function, but when the reward is changed, all pools are not updated to settle the user's reward. This will cause the users who are still in staking to be in the block. The rewards obtained after the rewards are changed do not match the expectations.

Code location：

```
    function setJfPerBlock(uint256 _perBlock) external onlyOwner {
        emit PerBlockSetting(jfPerBlock, _perBlock);
        jfPerBlock = _perBlock;
    }
```

Solution

It is recommended to force all pools to be updated before the adjustment of token block rewards to avoid the impact of user income.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002106080001	SlowMist Security Team	2021.06.01 - 2021.06.08	Medium 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 suggestion vulnerabilities. And 1 medium risk vulnerability were confirmed and being fixed; 1 suggestion vulnerability were ignored; All other findings were fixed. The code was not deployed to the mainnet. At present, part of the contract authority has not been transferred to the timelock contract, so there is still a risk of excessive authority.

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