Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.06.28, the SlowMist security team received the team's security audit application for Neopin Part2, 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.

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:

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

Serial Number	Audit Class	Audit Subclass
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
		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

Audit version:

https://github.com/Neopin/neopin-defi-contracts

commit: 6698f361c67abe6111ab2114a3571538f6d56161

-packages/dex-contract/contracts/farm

-packages/dex-contract/contracts/easyDex

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	Gas Optimization	Gas Optimization Audit	Suggestion	Confirmed
N3	Missing zero address validation	Others	Suggestion	Confirmed
N4	Compatibility issue	Others	Suggestion	Confirmed

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:

EasyRouter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
_approve	Internal	Can Modify State	-
_getPairAddress	Internal	-	-
addLiquidity	External	Can Modify State	whenNotPaused nonReentrant
addLiquidityETH	External	Payable	whenNotPaused nonReentrant
removeLiquidity	Public	Can Modify State	whenNotPaused nonReentrant
removeLiquidityETH	Public	Can Modify State	whenNotPaused nonReentrant
removeLiquidityETHSupportingFeeOnTransferTokens	Public	Can Modify State	whenNotPaused nonReentrant

BonusChef			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	onlyOwner
poolLength	External	-	-
updateRewardProvider	Public	Can Modify State	onlyOwner

BonusChef			
addPool	Public	Can Modify State	onlyOwner
getMultiplier	Public	-	-
pendingReward	External	-	-
updatePool	Public	Can Modify State	-
claimByDeposit	External	Can Modify State	nonReentrant
claimByWithdraw	External	Can Modify State	nonReentrant
claimReward	External	Can Modify State	nonReentrant
attachMasterChef	External	Can Modify State	-
detachMasterChef	External	Can Modify State	-
emergencyRedeemReward	Public	Can Modify State	onlyOwner
safeRewardTransfer	Internal	Can Modify State	-

MasterChef			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	onlyOwner
poolLength	External	-	-
updateRewardBar	Public	Can Modify State	onlyOwner
addPool	Public	Can Modify State	onlyOwner
updateBonusChef	Public	Can Modify State	onlyOwner

MasterChef			
updateNextRewardPerBlock	Public	Can Modify State	onlyOwner
getStakeToken	Public	-	-
getUserAmount	Public	-	-
getMultiplier	Public	-	-
pendingReward	External	-	-
updatePool	Public	Can Modify State	-
deposit	Public	Can Modify State	nonReentrant whenNotPaused
depositByEasyRouter	Public	Can Modify State	nonReentrant whenNotPaused
withdraw	Public	Can Modify State	nonReentrant whenNotPaused
withdrawByEasyRouter	Public	Can Modify State	nonReentrant whenNotPaused
emergencyWithdraw	Public	Can Modify State	nonReentrant whenNotPaused
emergencyRedeemReward	Public	Can Modify State	onlyOwner
safeRewardTransfer	Internal	Can Modify State	-

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

4.3 Vulnerability Summary

[N1] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

1.In the BonusChef and StakeChef contract, the owner role can withdraw any amount of the rewardToken through the emergencyRedeemReward function.

Code location:

dex-contract/contracts/farm/MasterChef.sol#304-308

```
    function emergencyRedeemReward(uint256 _pid, uint256 _amount) public onlyOwner {
        PoolInfo storage pool = poolInfo[_pid];
        safeRewardTransfer(pool.rewardToken, msg.sender, _amount);
        emit EmergencyRedeemReward(msg.sender, _pid, _amount);
    }
```

2.The owner role can add a pool arbitrarily through the add function, and there is a risk that the Owner can add apool to mine by itself to obtain rewards. When calling the add function to add a pool, the lastRewardBlock and totalAllocPoint will be updated, and related information about the pool will be stored.

Code location:

dex-contract/contracts/farm/BonusChef.sol#69-91

```
    function addPool(IERC20 _rewardToken, uint256 _rewardPerBlock, uint256
_startBlock, uint256 _endBlock) public onlyOwner {
        require(address(masterChef) != address(0), "not initialized masterChef");
        require(address(rewardProvider) != address(0), "not initialized
rewardProvider");
        require(_endBlock > _startBlock, "invalid endBlock");

        uint256 totalRewardAmount = _rewardPerBlock * (_endBlock - _startBlock);
        _rewardToken.safeTransferFrom(rewardProvider, address(this),
totalRewardAmount);

        uint256 lastRewardBlock = block.number > _startBlock ? block.number :
_startBlock;
        poolInfo.push(PoolInfo({
```

```
            rewardToken: _rewardToken,
            rewardPerBlock: _rewardPerBlock,
            lastRewardBlock: lastRewardBlock,
            accRewardPerShare: 0,
            startBlock: _startBlock,
            endBlock: _endBlock,
            spid: 0,
            stakeSupply: 0,
            isAttached: false
        }));
        uint256 _pid = poolInfo.length - 1;
        emit AddPool(_pid, _rewardToken, _rewardPerBlock, _startBlock, _endBlock);
    }
```

dex-contract/contracts/farm/MasterChef.sol#73-92

```
    function addPool(IERC20 _stakeToken, IERC20 _rewardToken, uint256 _startBlock,
uint256 _rewardPerBlock, IBonusChef _bonusChef, uint256 _bpid) public onlyOwner {
        require(address(rewardBar) != address(0), 'invalid RewardBar');
        uint256 lastRewardBlock = block.number > _startBlock ? block.number :
_startBlock;
        poolInfo.push(PoolInfo({
            stakeToken: _stakeToken,
            rewardToken: _rewardToken,
            rewardPerBlock: _rewardPerBlock,
            lastRewardBlock: lastRewardBlock,
            accRewardPerShare: 0,
            nextRewardPerBlock: 0,
            nextBlockNumber: 0,
            bonusChef: _bonusChef,
            bpid: _bpid
        }));
        uint256 _pid = poolInfo.length - 1;
        if (address(_bonusChef) != address(0)) {
            _bonusChef.attachMasterChef(_pid, _bpid);
        }
        emit AddPool(address(_stakeToken), address(_rewardToken), _startBlock,
_rewardPerBlock, address(_bonusChef), _bpid);
    }
```

Solution

It is recommended to use a time lock mechanism or community governance to restrict.

Status

Confirmed

[N2] [Suggestion] Gas Optimization

Category: Gas Optimization Audit

Content

In the EasyRouter contract, using assert will consume the remaining gas when the transaction fails to execute.

Code location:

dex-contract/contracts/easyDex/EasyRouter.sol#34

```
  receive() external payable {
    assert(msg.sender == WETH); // only accept ETH via fallback from the WETH
contract
  }
```

Solution

It is recommended to use require instead of assert to optimize gas.

Status

Confirmed

[N3] [Suggestion] Missing zero address validation

Category: Others

Content

In the MasterChef and the BonusChef contract, the owner role can set and change the rewardProvider and rewardBar, but there are missing zero address validation.

Code location:

dex-contract/contracts/farm/BonusChef.sol#63-67

```
    function updateRewardProvider(address _rewardProvider) public onlyOwner {
        rewardProvider = _rewardProvider;

        emit UpdateRewardProvider(_rewardProvider);
    }
```

dex-contract/contracts/farm/MasterChef.sol#68-71

```
    function updateRewardBar(IRewardBar _rewardBar) public onlyOwner {
        rewardBar = _rewardBar;
        emit UpdateRewardBar(address(_rewardBar));
    }
```

Solution

It is recommended to add zero address validation.

Status

Confirmed

[N4] [Suggestion] Compatibility issue

Category: Others

Content

In the MasterChef contract, users can stake/withdraw their tokens through the deposit function and withdraw function. It will directly record the amount parameter passed by the user into user.amount, and transfer the tokens to the contract through the safeTransferFrom function. If the contract receives deflationary tokens, the actual number of tokens received by the contract will not match the number of tokens recorded in the contract.

Code location:

dex-contract/contracts/farm/MasterChef.sol#177-201, 234-257

```
        function deposit(uint256 _pid, uint256 _amount) public nonReentrant
whenNotPaused {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];

        updatePool(_pid);

        if (user.amount > 0) {
            uint256 pending = user.amount * pool.accRewardPerShare / 1e12 -
user.rewardDebt;
            if(pending > 0) {
                safeRewardTransfer(pool.rewardToken, msg.sender, pending);
                emit ClaimReward(msg.sender, _pid, pending);
            }
            user.claimedReward += pending;
        }
        if (address(pool.bonusChef) != address(0)) {
            pool.bonusChef.claimByDeposit(_pid, pool.bpid, msg.sender, _amount);
        }
        if (_amount > 0) {
            pool.stakeToken.safeTransferFrom(address(msg.sender), address(this),
_amount);
            user.amount = user.amount + _amount;
        }
        user.rewardDebt = user.amount * pool.accRewardPerShare / 1e12;

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

       function withdraw(uint256 _pid, uint256 _amount) public nonReentrant
whenNotPaused {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];

        require(user.amount >= _amount, "withdraw not good");
        updatePool(_pid);

        uint256 pending = user.amount * pool.accRewardPerShare / 1e12 -
user.rewardDebt;
        if(pending > 0) {
            safeRewardTransfer(pool.rewardToken, msg.sender, pending);
            user.claimedReward += pending;
            emit ClaimReward(msg.sender, _pid, pending);
```

```
        }
        if (address(pool.bonusChef) != address(0)) {
            pool.bonusChef.claimByWithdraw(_pid, pool.bpid, msg.sender, _amount);
        }
        if(_amount > 0) {
            user.amount = user.amount - _amount;
            pool.stakeToken.safeTransfer(address(msg.sender), _amount);
        }
        user.rewardDebt = user.amount * pool.accRewardPerShare / 1e12;

        emit Withdraw(msg.sender, _pid, _amount);
    }
```

Solution

It is recommended to use the difference between the contract balance before and after the transfer to record the user's actual recharge amount.

Status

Confirmed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002206290001	SlowMist Security Team	2022.06.28 - 2022.06.29	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, 3 suggestion vulnerabilities. And 1 medium risk, 3 suggestion vulnerabilities were confirmed. 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