Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.03.25, the SlowMist security team received the Ruby Protocol team's security audit application for Ruby Protocol, 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

Project:

Ruby Protocol

Module:

contracts/amm

contracts/ruby_router

contracts/nfts

- RubyMasterChef.sol
- RubyMaker.sol
- RubyNFTAdmin.sol
- RubyStaker.sol

Project Git:

https://github.com/RubyExchange/contracts

commit:

957f00297cd1f50d4f0f3c27f89ace795ebbbd3d

Fixed version:

ce1ae8ea80d50f35064f7f2d98cf8e644cb0e998

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	_convert execution may keep failing	Design Logic Audit	Suggestion	Fixed

NO	Title	Category	Level	Status
N2	Unexpected swap fees	Others	Low	Fixed
N3	DoS issue	Denial of Service Vulnerability	Suggestion	Fixed
N4	unused variable	Others	Suggestion	Fixed
N5	Computational precision problem	Others	Suggestion	Ignored
N6	Missing event record	Malicious Event Log Audit	Suggestion	Fixed
N7	Risk of excessive authority	Authority Control Vulnerability	Low	Ignored
N8	Deflationary tokens are not compatible	Design Logic Audit	Low	Ignored

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:

UniswapV2ERC20			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-

UniswapV2ERC20			
_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	-

UniswapV2Factory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
allPairsLength	External	-	-
pairCodeHash	External	-	-
createPair	External	Can Modify State	-
setFeeTo	External	Can Modify State	-
setMigrator	External	Can Modify State	-
setPairCreator	External	Can Modify State	-
setAdmin	External	Can Modify State	-

UniswapV2Pair

UniswapV2Pair			
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
_updateSwap	Private	Can Modify State	-
skim	External	Can Modify State	lock
sync	External	Can Modify State	lock

UniswapV2Router02			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
_addLiquidity	Internal	Can Modify State	-
addLiquidity	External	Can Modify State	ensure

UniswapV2Router02			
removeLiquidity	Public	Can Modify State	ensure
removeLiquidityWithPermit	External	Can Modify State	-
_swap	Internal	Can Modify State	-
swapExactTokensForTokens	External	Can Modify State	ensure
swapTokensForExactTokens	External	Can Modify State	ensure
_swapSupportingFeeOnTransferTokens	Internal	Can Modify State	-
swapExactTokensForTokensSupportingFeeOnTransferTokens	External	Can Modify State	ensure
quote	Public	-	-
getAmountOut	Public	-	-
getAmountIn	Public	-	-
getAmountsOut	Public	-	-
getAmountsIn	Public	-	-
setFactory	External	Can Modify State	onlyOwner
setNftAdmin	External	Can Modify State	onlyOwner

RubyNFT			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer

RubyNFT			
mint	External	Can Modify State	onlyMinter
setMinter	External	Can Modify State	onlyOwner
setDescription	External	Can Modify State	onlyOwner
setVisualAppearance	External	Can Modify State	onlyOwner

RubyRouter			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
swap	Public	Can Modify State	-
_handleInputToken	Private	Can Modify State	-
_handleOutputToken	Private	Can Modify State	-
_swapAmm	Private	Can Modify State	-
_swapStablePool	Private	Can Modify State	-
_increaseTokenAllowance	Private	Can Modify State	-
enableStablePool	Public	Can Modify State	onlyOwner
disableStablePool	Public	Can Modify State	onlyOwner
setAmmRouter	Public	Can Modify State	onlyOwner
setNftAdmin	Public	Can Modify State	onlyOwner
setMaxHops	Public	Can Modify State	onlyOwner

RubyMaker

RubyMaker			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setBurnPercent	External	Can Modify State	onlyOwner
bridgeFor	Public	-	-
setBridge	External	Can Modify State	onlyOwner
convert	External	Can Modify State	onlyEOA
convertMultiple	External	Can Modify State	onlyEOA
_convert	Internal	Can Modify State	-
_convertStep	Internal	Can Modify State	-
_swap	Internal	Can Modify State	-
_toRUBY	Internal	Can Modify State	-

RubyMasterChef			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
poolLength	External	-	-
add	Public	Can Modify State	onlyOwner
set	Public	Can Modify State	onlyOwner
pendingTokens	External	-	-
rewarderBonusTokenInfo	Public	-	-

RubyMasterChef			
massUpdatePools	Public	Can Modify State	-
updatePool	Public	Can Modify State	-
deposit	External	Can Modify State	nonReentrant
withdraw	External	Can Modify State	nonReentrant
claim	External	Can Modify State	-
_mintRubyRewards	Internal	Can Modify State	-
emergencyWithdraw	External	Can Modify State	nonReentrant
setTreasuryAddr	Public	Can Modify State	-
setTreasuryPercent	Public	Can Modify State	onlyOwner
setRubyStaker	Public	Can Modify State	onlyOwner
updateEmissionRate	Public	Can Modify State	onlyOwner
emergencyWithdrawRubyTokens	External	Can Modify State	onlyOwner

RubyNFTAdmin			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
calculateAmmSwapFeeDeduction	External	-	-
mintProfileNFT	External	Can Modify State	onylMinter
setProfileNFT	External	Can Modify State	onlyOwner
setFreeSwapNFT	External	Can Modify State	onlyOwner

RubyNFTAdmin			
setMinter	External	Can Modify State	onlyOwner

RubyStaker			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setRewardMinter	External	Can Modify State	onlyOwner
addReward	Public	Can Modify State	onlyOwner
approveRewardDistributor	External	Can Modify State	onlyOwner
_rewardPerToken	Internal	-	-
_earned	Internal	-	-
lastTimeRewardApplicable	Public	-	-
rewardPerToken	External	-	-
getRewardForDuration	External	-	-
claimableRewards	External	-	-
totalBalance	External	-	-
unlockedBalance	External	-	-
earnedBalances	External	-	-
lockedBalances	External	-	-
withdrawableBalance	Public	-	-

RubyStaker			
stake	External	Can Modify State	nonReentrant updateReward
mint	External	Can Modify State	onlyRewardMinter updateReward
withdraw	Public	Can Modify State	nonReentrant updateReward
getReward	Public	Can Modify State	nonReentrant updateReward
exit	External	Can Modify State	updateReward
withdrawExpiredLocks	External	Can Modify State	-
_notifyReward	Internal	Can Modify State	-
notifyRewardAmount	External	Can Modify State	onlyRewardDistributor updateReward
recoverERC20	External	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [Suggestion] _convert execution may keep failing

Category: Design Logic Audit

Content

- contracts/RubyMaker.sol

  If `rubyRewards==0`, the transaction will fail.

```
function _convert(address token0, address token1) internal {
        // Interactions
        IUniswapV2Pair pair = IUniswapV2Pair(factory.getPair(token0, token1));
        require(address(pair) != address(0), "RubyMaker: Invalid pair");
```

```
        IERC20(address(pair)).safeTransfer(address(pair),
pair.balanceOf(address(this)));

        (uint256 amount0, uint256 amount1) = pair.burn(address(this));
        if (token0 != pair.token0()) {
            (amount0, amount1) = (amount1, amount0);
        }
        uint256 totalConvertedRuby = _convertStep(token0, token1, amount0, amount1);
        uint256 rubyToBurn = (totalConvertedRuby.mul(burnPercent)).div(100);

        uint256 rubyRewards = totalConvertedRuby - rubyToBurn;

        // Burn ruby
        RubyToken(ruby).burn(rubyToBurn);

        rubyStaker.notifyRewardAmount(1, rubyRewards); //SlowMist//If rubyRewards==0,
the transaction will fail

        emit LogConvert(msg.sender, token0, token1, amount0, amount1, rubyRewards,
rubyToBurn);
    }
```

Solution

In the case of `rubyRewards=0`, `notifyRewardAmount` can not be executed

Status

Fixed

[N2] [Low] Unexpected swap fees

Category: Others

Content

- contracts/amm/UniswapV2Pair.sol

Users can call swap by themselves and `feeMultiplier` can be set to 1000 to achieve 0 transaction fee.

```
function swap(
        uint256 amount0Out,
```

```
        uint256 amount1Out,
        address to,
        uint256 feeMultiplier,
        bytes calldata data
    ) external lock {
        require(amount0Out > 0 || amount1Out > 0, "UniswapV2:
INSUFFICIENT_OUTPUT_AMOUNT");
        (uint112 _reserve0, uint112 _reserve1, ) = getReserves(); // gas savings
        require(amount0Out < _reserve0 && amount1Out < _reserve1, "UniswapV2:
INSUFFICIENT_LIQUIDITY");
        require(feeMultiplier >= 997 && feeMultiplier <= 1000, "UniswapV2:
FEE_MULTIPLIER");

        uint256 balance0;
        uint256 balance1;
        {
            // scope for _token{0,1}, avoids stack too deep errors
            address _token0 = token0;
            address _token1 = token1;
            require(to != _token0 && to != _token1, "UniswapV2: INVALID_TO");
            if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); //
optimistically transfer tokens
            if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); //
optimistically transfer tokens
            if (data.length > 0) IUniswapV2Callee(to).uniswapV2Call(msg.sender,
amount0Out, amount1Out, data);
            balance0 = IERC20Uniswap(_token0).balanceOf(address(this));
            balance1 = IERC20Uniswap(_token1).balanceOf(address(this));
        }
        // function split to avoid stack too deep errors
        _updateSwap(balance0, balance1, amount0Out, amount1Out, to, feeMultiplier);
    }
```

Solution

If you want the fee to be expected by the project party, you can add a permission management to the swap call.

Status

Fixed

[N3] [Suggestion] DoS issue

Category: Denial of Service Vulnerability

Content

`numRewards` only increases and does not decrease. If the length of `numRewards` is too long, the `updateReward` function will fail to execute and other functions cannot be executed.

```
modifier updateReward(address account) {
    uint256 balance;
    uint256 supply = lockedSupply;
    rewardData[0].rewardPerTokenStored = _rewardPerToken(0, supply);
    rewardData[0].lastUpdateTime = lastTimeRewardApplicable(0);
    if (account != address(0)) {
        // Special case, use the locked balances and supply for stakingReward rewards
        rewards[account][0] = _earned(account, 0, balances[account].locked, supply);
        userRewardPerTokenPaid[account][0] = rewardData[0].rewardPerTokenStored;
        balance = balances[account].total;
    }

    supply = totalSupply;
    for (uint256 i = 1; i <= numRewards; i++) {
        rewardData[i].rewardPerTokenStored = _rewardPerToken(i, supply);
        rewardData[i].lastUpdateTime = lastTimeRewardApplicable(i);
        if (account != address(0)) {
            rewards[account][i] = _earned(account, i, balance, supply);
            userRewardPerTokenPaid[account][i] = rewardData[i].rewardPerTokenStored;
        }
    }
    _;
}
```

Solution

The length of `numRewards` can be limited.

Status

Fixed

[N4] [Suggestion] unused variable

Category: Others

Content

migrator not used.

```
    function setMigrator(address newMigrator) external override {
        require(msg.sender == admin, "UniswapV2: FORBIDDEN");
        migrator = newMigrator;
        emit MigratorSet(newMigrator);
    }
```

Solution

delete unused variables.

Status

Fixed

[N5] [Suggestion] Computational precision problem

Category: Others

Content

- contracts/RubyStaker.sol

  `unlockTime` Dividing first and multiplying will result in precision error. The calculated

  block.timestamp.div(rewardsDuration).mul(rewardsDuration) will be smaller than block.timestamp.

```
function stake(uint256 amount, bool lock) external nonReentrant
updateReward(msg.sender) {
        require(amount > 0, "RubyStaker: Invalid staking amount");
        totalSupply = totalSupply.add(amount);
        Balances storage bal = balances[msg.sender];
        bal.total = bal.total.add(amount);
        if (lock) {
            lockedSupply = lockedSupply.add(amount);
            bal.locked = bal.locked.add(amount);
            uint256 unlockTime =
```

```
block.timestamp.div(rewardsDuration).mul(rewardsDuration).add(lockDuration);//SlowMis
t//Computational precision problem
            uint256 idx = userLocks[msg.sender].length;
            if (idx == 0 || userLocks[msg.sender][idx - 1].unlockTime < unlockTime) {
                userLocks[msg.sender].push(LockedBalance({ amount: amount,
unlockTime: unlockTime }));
            } else {
                userLocks[msg.sender][idx - 1].amount = userLocks[msg.sender][idx -
1].amount.add(amount);
            }
        } else {
            bal.unlocked = bal.unlocked.add(amount);
        }
        rubyToken.safeTransferFrom(msg.sender, address(this), amount);
        emit Staked(msg.sender, amount);
    }
```

Solution

Calculate multiplication first before calculating division.

Status

Ignored; In line with project design decisions - the precision loss leads to withdrawal time grouping.

[N6] [Suggestion] Missing event record

Category: Malicious Event Log Audit

Content

- contracts/ruby_router/RubyRouter.sol

  Modifying important variables in the contract requires corresponding event records.

```
function setAmmRouter(IUniswapV2Router02 newAmmRouter) public onlyOwner {
    require(address(newAmmRouter) != address(0), "RubyRouter: Invalid AMM router
address.");
    ammRouter = newAmmRouter;
    }

    function setNftAdmin(IRubyNFTAdmin newNftAdmin) public onlyOwner {
        require(address(newNftAdmin) != address(0), "RubyRouter: Invalid NFT admin
```

```
address.");
        nftAdmin = newNftAdmin;
    }

    function setMaxHops(uint256 maxSwapHops) public onlyOwner {
        require(maxSwapHops > 0, "RubyRouter: Invalid max swap hops;");
        _maxSwapHops = maxSwapHops;
    }
```

Solution

Record key events.

Status

Fixed

[N7] [Low] Risk of excessive authority

Category: Authority Control Vulnerability

Content

The authority of the owner role is too large, and the modification of the owner can take effect immediately. Once the private key is lost,attacker can cause losses to the project party by modifying the owner permissions.

- contracts/RubyMaker.sol

```
function withdrawLP(address pair) external onlyOwner {
        require(pair != address(0), "RubyMaker: Invalid pair address.");
        require(isContract(pair), "RubyMaker: pair is not a contract address.");
        IERC20 _pair = IERC20(pair);
        uint256 pairBalance = _pair.balanceOf(address(this));
        _pair.safeTransfer(owner(), pairBalance);
        emit PairWithdrawn(pair, pairBalance);
    }
```

- contracts/RubyStaker.sol

```
function setRewardMinter(address _rewardMinter) external onlyOwner {
        require(_rewardMinter != address(0), "RubyStaker: Invalid new reward
```

```
minter.");
        rewardMinter = _rewardMinter;
        emit RewardMinterSet(rewardMinter);
    }

function mint(address user, uint256 amount) external override onlyRewardMinter
updateReward(user) {
        totalSupply = totalSupply.add(amount);
        Balances storage bal = balances[user];
        bal.total = bal.total.add(amount);
        bal.earned = bal.earned.add(amount);
        uint256 unlockTime =
block.timestamp.div(rewardsDuration).mul(rewardsDuration).add(lockDuration);
        LockedBalance[] storage earnings = userEarnings[user];
        uint256 idx = earnings.length;

        if (idx == 0 || earnings[idx - 1].unlockTime < unlockTime) {
            earnings.push(LockedBalance({ amount: amount, unlockTime: unlockTime }));
        } else {
            earnings[idx - 1].amount = earnings[idx - 1].amount.add(amount);
        }
        emit Staked(user, amount);
    }
```

- contracts/RubyMasterChef.sol

```
function emergencyWithdrawRubyTokens(address _receiver, uint256 _amount) external
onlyOwner {
        require(_receiver != address(0), "RubyMasterChef: Invalid withdrawal
address.");
        require(_amount != 0, "RubyMasterChef: Invalid withdrawal amount.");
        require(RUBY.balanceOf(address(this)) >= _amount, "RubyMasterChef: Not enough
balance to withdraw.");
        RUBY.safeTransfer(_receiver, _amount);
        emit RubyTokenEmergencyWithdrawal(_receiver, _amount);
    }
```

- contracts/RubyNFTAdmin.sol

```
function setProfileNFT(address newProfileNFT) external override onlyOwner {
        require(newProfileNFT != address(0), "RubyNFTAdmin: Invalid profile NFT");
        profileNFT = newProfileNFT;
        emit RubyProfileNFTset(profileNFT);
    }

    function setFreeSwapNFT(address newFreeSwapNFT) external override onlyOwner {
        require(newFreeSwapNFT != address(0), "RubyNFTAdmin: Invalid free swap NFT");
        freeSwapNFT = newFreeSwapNFT;
        emit FreeSwapNFTSet(freeSwapNFT);
    }

    function setMinter(address minter, bool allowance) external override onlyOwner {
        require(minter != address(0), "RubyNFTAdmin: Invalid minter address");
        minters[minter] = allowance;
        emit MinterSet(minter, allowance);
    }
```

Solution

It is recommended to transfer the permissions of roles with excessive permissions to governance contracts or timelock contracts. At least multisig should be used.

Status

Ignored; The contract will be managed using a multi-signature account after release.

[N8] [Low] Deflationary tokens are not compatible

Category: Design Logic Audit

Content

- contracts/RubyMasterChef.sol

  If the number of deflationary token records is smaller than the actual number of receipts, if malicious users continue to `deposit` and `withdraw`, the pool of deflationary tokens will be exhausted,malicious users can obtain excess revenue in the corresponding pid.

```
    function deposit(uint256 _pid, uint256 _amount) external nonReentrant {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        updatePool(_pid);
        if (user.amount > 0) {
            // Harvest accRubyPerShare
            uint256 pending =
user.amount.mul(pool.accRubyPerShare).div(ACC_TOKEN_PRECISION).sub(user.rewardDebt);
            _mintRubyRewards(msg.sender, pending);
            emit Harvest(msg.sender, _pid, pending);
        }
        user.amount = user.amount.add(_amount);
        user.rewardDebt =
user.amount.mul(pool.accRubyPerShare).div(ACC_TOKEN_PRECISION);

        IRubyMasterChefRewarder rewarder = poolInfo[_pid].rewarder;
        if (address(rewarder) != address(0)) {
            rewarder.onRubyReward(msg.sender, user.amount);
        }

        pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
        emit Deposit(msg.sender, _pid, _amount);
    }
```

Solution

Check the token balance before and after the recharge as the real recharge amount.

Status

Ignored; The project will not use deflationary tokens.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002204130002	SlowMist Security Team	2022.03.25 - 2022.04.13	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 3 low risk, 5 suggestion vulnerabilities.

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