QuillAudits

Audit Report
August, 2021

Contents

Scope of Audit

The scope of this audit was to analyze and document the RuufPay Token smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.
SmartCheck.

Static Analysis
Static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step a series of automated tools are used to test security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of automated analysis were manually verified.

Gas Consumption
In this step we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

Every issue in this report has been assigned with a severity level. There are four levels of severity and each of them has been explained below.

High severity issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality and we recommend these issues to be fixed before moving to a live environment.

Medium level severity issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems and they should still be fixed.

Low level severity issues

Low level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity four issues which indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	0	0
Closed	1	2	2	2

Introduction

During the period of **August 11, 2021 to August 19, 2021** - QuillAudits Team performed a security audit for RuufPay smart contracts.

The code for the audit was taken from the following official link: https://github.com/RuufPay/smartcontracts/blob/main/contracts/StakeFarm.sol

Note	Date	Commit Hash
Version 1	August	af1263f9569a4030fd724cdeaa08475d407ec57c
Version 2	August	2bf0728d389b607ca32d7abaa680c5a48dffa962

Issues Found – Code Review / Manual Testing

High severity issues

1. Unlimited rewards can be withdrawn

Line	Code

41-45	else {
	balances[_user].amount += _amount;
	balances[_user].rewards += _calculateRewards(_user);
	balances[_user].stakeDate = block.timestamp;
	}

Description

At every new stake deposit, rewards up to that time are calculated and updated for that user. The stakeDate is also reset to block.timestamp. This means, for the next stake deposit, new rewards will be calculated from the last stakeDate.

As we can see, _calculateRewards() function is used which calculates a multiplier that is multiplied with the user's amount in the balance mapping. But as the amount is updated before calling _calculateRewards() function, the reward calculation uses the new amount.

Example scenario:
This means a user can deposit 1 token for 30 days, and then make a new stake again on the 30th day with 100 tokens, then the updated rewards in the balance for that user will be very high as it will take into consideration the 100 tokens also for calculating the reward for the last 30 days. This can be repeated a number of times to withdraw all the tokens of the contract as rewards.

Remediation

Update the amount in the balance mapping after updating the rewards.

Status: Closed

In the latest version, 'amount' attribute is updated after rewards.

Medium severity issues

2. Centralization Risks

Description
The role owner has the authority to
- withdraw all the tokens from the contract at any point in time.

Remediation
We advise the client to handle the governance account carefully to avoid any potential hack. We also recommend the client to consider the following solutions:

- Timelock with reasonable latency for community awareness on privileged operations;
- Multisig with community-voted 3rd-party independent co-signers;
- DAO or Governance module increasing transparency and community involvement;

Status: Closed
In the latest version, the function to withdraw all the tokens was removed from the contract.

3. Function input parameters lack of check

Line	Code

64-66	function withdraw(address _user) external {
	require(msg.sender == owner, "BadOwner");
	require(balances[msg.sender].stakeDate > 0, "NoStaked");

Description
There is no condition which checks if the user has any amount of tokens staked or not. It only checks for the owner's stakeDate > 0.

Remediation
Instead of checking for the msg.sender's stakeDate > 0, check for _user

Status: Closed
In the latest version, _user's balance was checked.

Low level severity issues

4. Missing zero address validation

Line	Code

86-89	function changeOwner(address _owner) external {
	require(msg.sender == owner, "BadOwner");
	owner = _owner;
	}
	}

Description

When updating the owner address, it should be checked for zero address. Otherwise, the owner will never get back the ownership of the contract and will not be able to perform the onlyOwner actions.

Remediation

Use a require statement to check for zero address when updating the owner address.

Status: Closed

Comments by the Auditee: It is intentional, so that the owner can burn the admin keys if they want.

5. Unchecked Transfer

Line	Code
47	IERC20(homeToken).transferFrom(msg.sender, address(this), _amount);
59	IERC20(homeToken).transfer(msg.sender, homes + rewards);
72	IERC20(homeToken).transfer(_user, homes + rewards);
81	IERC20(homeToken).transfer(owner, balance);

Description

The return value of an external transfer/transferFrom call is not checked.

Remediation
Use SafeERC20 library by OpenZeppelin, or ensure that the transfer/transferFrom return value is checked.

Status: Closed
In the latest version, SafeERC20 by OpenZeppelin was implemented.

Informational

6. Missing Events for Significant Transactions

Description
The missing event makes it difficult to track off-chain liquidity fee changes. An event should be emitted for significant transactions calling the function:
- changeOwner

Remediation
We recommend emitting an event for the change of ownership.

Status: Closed
In the latest version, 'OwnershipTransferred' event is emitted.

7. Floating Pragma

pragma solidity ^0.8.3;

Description
Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation
Lock the pragma version and also consider known bugs for the compiler version that is chosen.

Status: Closed
In the latest version, pragma was fixed to solidity 0.8.7

Functional test

Function Names	Testing results
stake()	Passed
withdraw()	Passed
emergencyWithdraw()	Passed
changeOwner()	Passed
getUserData()	Passed

Automated Testing

Slither

```
INFO:Detectors:
StakeFarm.stake(address,uint256) (StakeFarm.sol#31-50) ignores return value by IERC20(homeToken).transferFrom(msg.sender,address(this),
_amount) (StakeFarm.sol#47)
StakeFarm.withdraw() (StakeFarm.sol#52-62) ignores return value by IERC20(homeToken).transfer(msg.sender,homes + rewards) (StakeFarm.so
l#59)
StakeFarm.withdraw(address) (StakeFarm.sol#64-75) ignores return value by IERC20(homeToken).transfer(_user,homes + rewards) (StakeFarm.
sol#72)
StakeFarm.emergencyWithdraw() (StakeFarm.sol#77-84) ignores return value by IERC20(homeToken).transfer(owner,balance) (StakeFarm.sol#81
)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
```

```
StakeFarm._calculateMultiplier(uint256) (StakeFarm.sol#105-123) performs a multiplication on the result of a division:
        -multiplier = eight.div(100).mul(secondsToCalculate.pow(one.div(3))) (StakeFarm.sol#110)
StakeFarm._calculateMultiplier(uint256) (StakeFarm.sol#105-123) performs a multiplication on the result of a division:
        -multiplier = secondsToCalculate_scope_0.mul(maxOneMonth).div(_SECONDS_IN_ONE_MONTH) (StakeFarm.sol#115)
        -delta__calculateMultiplier_asm_0 = multiplier * 20 (StakeFarm.sol#119)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
StakeFarm.stake(address,uint256) (StakeFarm.sol#31-50) uses a dangerous strict equality:
        - balances[_user].amount == 0 (StakeFarm.sol#35)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
StakeFarm.constructor(address)._homeToken (StakeFarm.sol#26) lacks a zero-check on :
                - homeToken = _homeToken (StakeFarm.sol#27)
StakeFarm.changeOwner(address)._owner (StakeFarm.sol#86) lacks a zero-check on :
                - owner = _owner (StakeFarm.sol#88)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in StakeFarm.emergencyWithdraw() (StakeFarm.sol#77-84):
        External calls:
        - IERC20(homeToken).transfer(owner,balance) (StakeFarm.sol#81)
        Event emitted after the call(s):
        - EmegercyWithdraw(owner,balance) (StakeFarm.sol#83)
Reentrancy in StakeFarm.stake(address,uint256) (StakeFarm.sol#31-50):
        External calls:
        - IERC20(homeToken).transferFrom(msg.sender,address(this),_amount) (StakeFarm.sol#47)
        Event emitted after the call(s):
        - HomeTokenStaked(_user,_amount) (StakeFarm.sol#49)
```

```
Reentrancy in StakeFarm.withdraw() (StakeFarm.sol#52-62):
        External calls:
        - IERC20(homeToken).transfer(msg.sender,homes + rewards) (StakeFarm.sol#59)
        Event emitted after the call(s):
        - Withdraw(msg.sender,homes,rewards) (StakeFarm.sol#61)
Reentrancy in StakeFarm.withdraw(address) (StakeFarm.sol#64-75):
        External calls:
        - IERC20(homeToken).transfer(_user,homes + rewards) (StakeFarm.sol#72)
        Event emitted after the call(s):
        - Withdraw(_user,homes,rewards) (StakeFarm.sol#74)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
StakeFarm.stake(address,uint256) (StakeFarm.sol#31-50) uses timestamp for comparisons
        Dangerous comparisons:
        - balances[_user].amount == 0 (StakeFarm.sol#35)
StakeFarm.withdraw() (StakeFarm.sol#52-62) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(balances[msg.sender].stakeDate > 0,NoStaked) (StakeFarm.sol#53)
StakeFarm.withdraw(address) (StakeFarm.sol#64-75) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(balances[msg.sender].stakeDate > 0,NoStaked) (StakeFarm.sol#66)
StakeFarm._calculateMultiplier(uint256) (StakeFarm.sol#105-123) uses timestamp for comparisons
        Dangerous comparisons:
        - (block.timestamp - _stakeDate) / _SECONDS_IN_ONE_MONTH > 0 (StakeFarm.sol#108)
        - multiplier > 250000000000000000 (StakeFarm.sol#111)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

Results

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the RuufPay platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the RuufPay Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.

No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract.

Many issues were discovered during the initial audit; all of them are fixed in the latest version and checked for correctness.

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com