QuillAudits

Audit Report
October, 2022

For

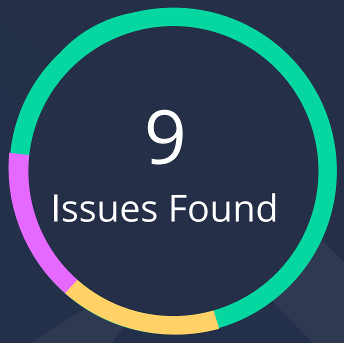
ØSPACE

Table of Content

Table of Content

Executive Summary

Project Name	SpaceFinance SpaceFarm Contract
Overview	SpaceFarm is a yield generating contract for the star token and any other added token. Also it has additional features to stake NFTs to generate extra yield.
Timeline	22nd September, 2022 - 28th October, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze SpaceFinance SpaceFarm codebase for quality, security, and correctness. *https://github.com/SpaceFinance/space-contract/blob/main/StarFarm.sol*
Fixed In	d4e153fa311f668f4bfe7296913438cb0e5fd258



🟥 High 🟨 Medium
🟩 Low 🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	1	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	2	4	1

Types of Severities

High

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 be fixed before moving to a live environment.

Medium

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

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 issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open
Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved
These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged
Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved
Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Using block.timestamp
- ✓ Multiple Sends
- ✓ Using SHA3
- ✓ Using suicide
- ✓ Using throw
- ✓ Using inline assembly

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 analysed 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.

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 the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the 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 the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - SpaceFarm.sol

High Severity Issues

No issues found

Medium Severity Issues

A1. Centralization Risk in SpaceFarm Contract

Description

In the contracts, the role Owner has the authority over the following functions: add() set() setMigrator(), updateMultiplier(), addSetBlokcReward(), setBlockReward(), updatelpPerBlock(), migrate(), withdrawLp()
Any compromise to the Owner account may allow the hacker to take advantage of this

Remediation

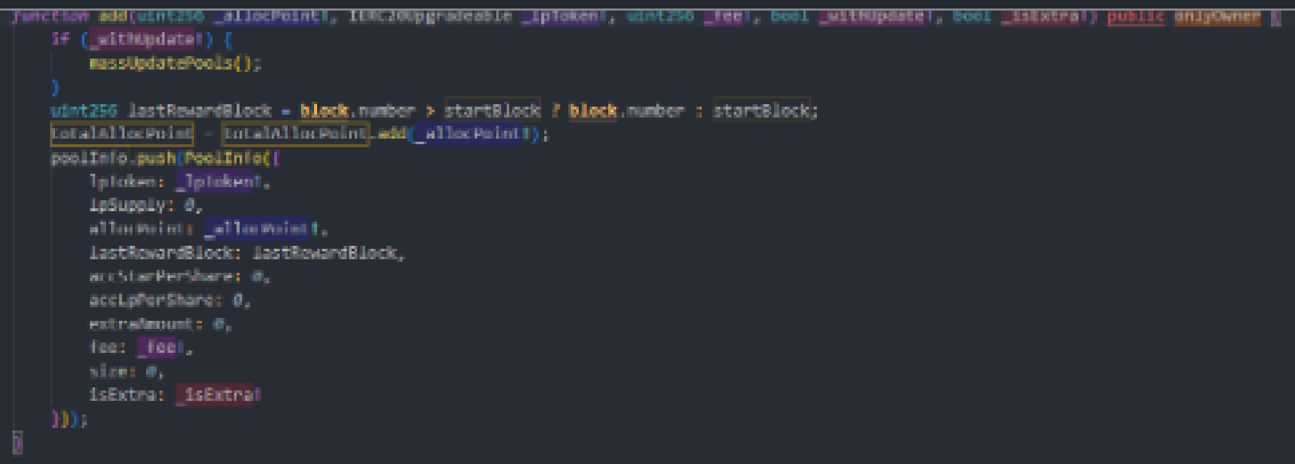
We advise the client to carefully manage the Owner account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets

Status

Resolved

Auditor's Comment: SpaceFi team understands the risk of losing private keys. They will be using multisig to manage the keys.

A2. Possibility of adding same token

Line	modifier - Function: add()
152	

```
function add(uint256 _allocPoint, IERC20Upgradeable _lpToken, uint256 _fee, bool _withUpdate, bool _isExtra) public onlyOwner {
    if (_withUpdate) {
        massUpdatePools();
    }
    uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
    totalAllocPoint = totalAllocPoint.add(_allocPoint);
    poolInfo.push(PoolInfo({
        lpToken: _lpToken,
        lpSupply: 0,
        allocPoint: _allocPoint,
        lastRewardBlock: lastRewardBlock,
        accStarPerShare: 0,
        accLpPerShare: 0,
        extraAmount: 0,
        fee: _fee,
        size: 0,
        isExtra: _isExtra
    }));
}
```

Description

When the same LP token is added into a pool more than once in function add() , the total amount of reward in function updatePool() will be incorrectly calculated. The current implementation is relying on the operation correctness to avoid repeatedly adding the same LP token to the pool, as the function will only be called by the owner.

Remediation

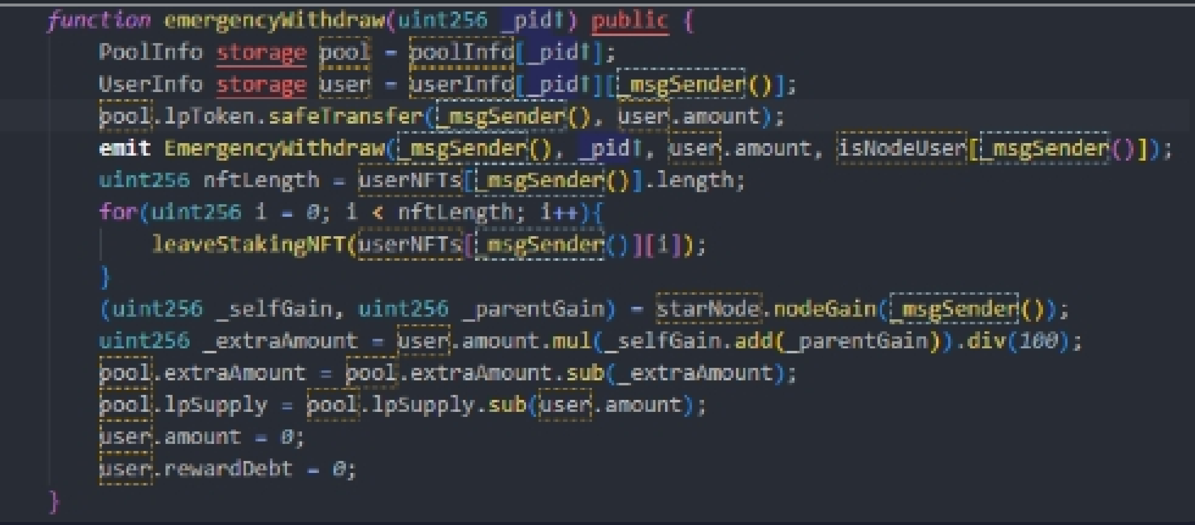
We recommend adding the check for ensuring whether the given pool for addition is a duplicate of an existing pool so that the pool addition is only successful when there is no duplicate. This can be done by using a mapping of addresses -> booleans , which can restrict the same address from being added twice.

Status

Resolved

Low Severity Issues

A3. Missing Check-Interaction-Effect pattern

Line	Function: emergencyWithdraw()
649-664	

```
function emergencyWithdraw(uint256 _pid↑) public {
    PoolInfo storage pool = poolInfo[_pid↑];
    UserInfo storage user = userInfo[_pid↑][_msgSender()];
    pool.lpToken.safeTransfer(_msgSender(), user.amount);
    emit EmergencyWithdraw(_msgSender(), _pid↑, user.amount, isNodeUser[_msgSender()]);
    uint256 nftLength = userNFTs[_msgSender()].length;
    for(uint256 i = 0; i < nftLength; i++){
        leaveStakingNFT(userNFTs[_msgSender()][i]);
    }
    (uint256 _selfGain, uint256 _parentGain) = starNode.nodeGain(_msgSender());
    uint256 _extraAmount = user.amount.mul(_selfGain.add(_parentGain)).div(100);
    pool.extraAmount = pool.extraAmount.sub(_extraAmount);
    pool.lpSupply = pool.lpSupply.sub(user.amount);
    user.amount = 0;
    user.rewardDebt = 0;
}
```

Description

In emergencyWithdraw() the order of external call/transfer function CIE pattern is not followed. It might lead to a re-entrancy bug.

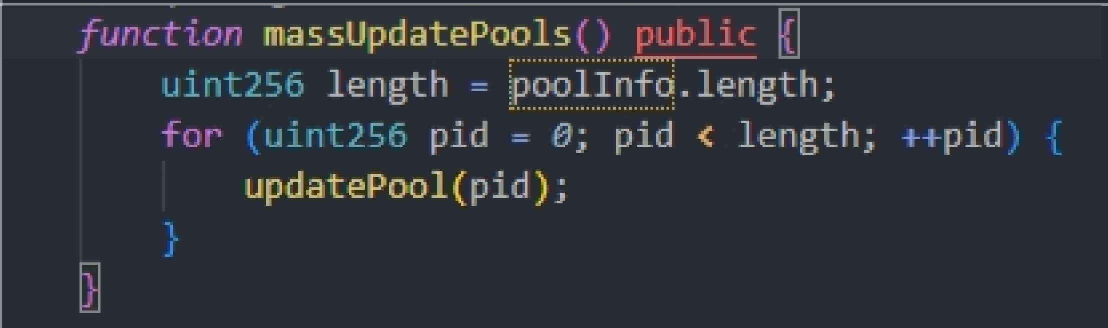
Remediation

It can be remediated using the correct CIE pattern and also use re-entrancygaurd from openzeppelin.

Status

Resolved

A4. Increase in length of pool can cause out of gas error

Line	Function: emergencyWithdraw()
386-392	

```
function massUpdatePools() public {
    uint256 length = poolInfo.length;
    for (uint256 pid = 0; pid < length; ++pid) {
        updatePool(pid);
    }
}
```

Description

In function massUpdatePools() it is possible that function can run out of gas because it does not delete the unused pools.

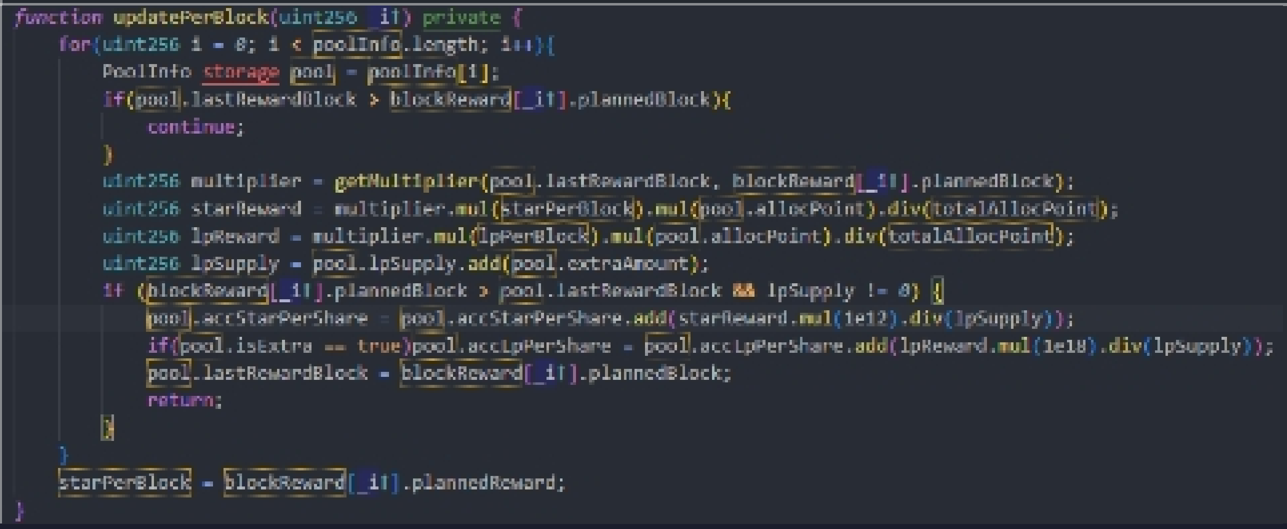
Remediation

Possible solution is to delete unused pools.

Status

Resolved

A5. Large supply of tokens in pool result in less reward

Line	Function: updatePerBlock()
283	

```
function updatePerBlock(uint256 _i1) private {
    for(uint256 i = 0; i < poolInfo.length; i++){
        PoolInfo storage pool = poolInfo[i];
        if(pool.lastRewardBlock > blockReward[_i1].plannedBlock){
            continue;
        }
        uint256 multiplier = getMultiplier(pool.lastRewardBlock, blockReward[_i1].plannedBlock);
        uint256 starReward = multiplier.mul(starPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
        uint256 lpReward = multiplier.mul(lpPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
        uint256 lpSupply = pool.lpSupply.add(pool.extraAmount);
        if (blockReward[_i1].plannedBlock > pool.lastRewardBlock && lpSupply != 0) {
            pool.accStarPerShare = pool.accStarPerShare.add(starReward.mul(1e12).div(lpSupply));
            if(pool.isExtra == true)pool.accLpPerShare = pool.accLpPerShare.add(lpReward.mul(1e18).div(lpSupply));
            pool.lastRewardBlock = blockReward[_i1].plannedBlock;
            return;
        }
    }
    starPerBlock = blockReward[_i1].plannedReward;
}
```

Description

In function updatePerBlock() it is possible that if a token with huge supply added like 4T, 8T, 400B such a memecoins then pool.accStarPerShare can become close to zero resulting in almost no shares in reward.

Remediation

While adding any token make sure to add tokens which have manageable supply and check the precision of the token too.

Status

Acknowledged

Comment: Client dev team understands the issue with large enough token supply can cause issues. They are acknowledging this issue by taking care and being extra careful with which token they are going to add.

A.6: Incompatibility with deflationary tokens

Description

When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged transaction fee. For example, if a user stakes 100 deflationary tokens (with a 10% transaction fee), only 90 tokens actually arrive in the contract. However, the user can still withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction. The contract takes the pool token balance (the lpSupply) into account when calculating the users' reward. An attacker can repeat the process of deposit and withdraw to lower the token balance (lpSupply) in a deflationary token pool and cause the contract to increase the reward amount.

Reference: *https://thoreum-finance.medium.com/what-exploit-happened-today-for-gocerberus-and-garuda-also-for-lokum-ybear-piggy-caramelswap-3943ee23a39f*

Remediation

If there is a need to add deflationary token please have necessary measures and mitigation mechanisms to keep track of balances

Status

Resolved

A.7: Missing comments in the code

Description

The SpaceFarm contract is missing proper code commenting. While it may not seem a necessary step, having comments makes it easier to understand and read the code.

Remediation

Please make sure to add proper comments for the functions or you can use natspec code commenting.

Status

Resolved

Informational Issues

A.8: Unlocked pragma (pragma solidity ^0.8.0)

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.

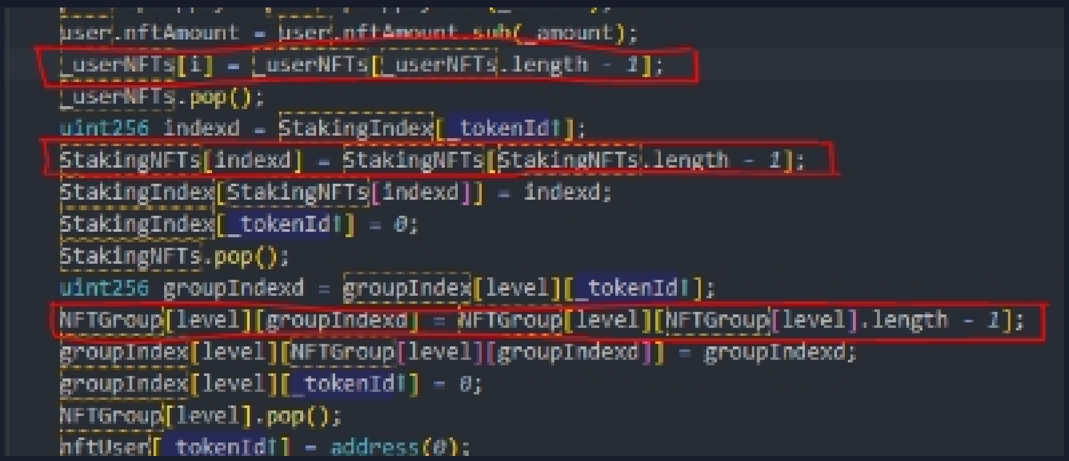
Remediation

Here all the in-scope contracts have an unlocked pragma, it is recommended to lock the same

Status

Resolved

A9. General Recommendations



```
user.nftAmount = user.nftAmount.sub(_amount);
_userNFTs[i] = _userNFTs[_userNFTs.length - 1];
_userNFTs.pop();
uint256 indexd = StakingIndex[_tokenId];
StakingNFTs[indexd] = StakingNFTs[StakingNFTs.length - 1];
StakingIndex[StakingNFTs[indexd]] = indexd;
StakingIndex[_tokenId] = 0;
StakingNFTs.pop();
uint256 groupIndexd = groupIndex[level][_tokenId];
NFTGroup[level][groupIndexd] = NFTGroup[level][NFTGroup[level].length - 1];
groupIndex[level][NFTGroup[level][groupIndexd]] = groupIndexd;
groupIndex[level][_tokenId] = 0;
NFTGroup[level].pop();
nftUser[_tokenId] = address(0);
```

1) In some places in code if you know it won't overflow/underflow then you can use unchecked blocks to save gas

2) Function such as updateMultiplier(), add(), set(), getNFTGroupAmount(), getUserStakingAmount(), gertStakingNFTAmount(), emergencyWithdraw(), setToken(), withdrawLp(), addBlockReward(), setBlockReward() can be set as external rather than public to save gas

Status

Resolved

Functional Testing

- ✓ mint emit test (99ms)
- ✓ setLockRatio (63ms)
- ✓ lockWithdraw
- ✓ allWithdrawal (151ms)
- ✓ addTotalAmount (65ms)
- ✓ nodeGain
- ✓ depositNode
- ✓ add (181ms)
- ✓ set (179ms)
- ✓ addBlockReward (90ms)
- ✓ setBlockReward (145ms)
- ✓ addStarPerBlock (179ms)
- ✓ delBlockReward (95ms)
- ✓ reckon setBlockReward (87ms)
- ✓ reckon (178ms)
- ✓ deposit (1337ms)
- ✓ withdraw (813ms)
- ✓ mint (6197ms)

Automated Tests

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

Closing Summary

In this report, we have considered the security of the SpaceFinance Farm. We performed our audit according to the procedure described above.

Some Issues of Medium,low and informational were Found During the Course of Audit.In the End, SpaceFi Team Resolved Most of the Issues.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the SpaceFinance Farm 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 SpaceFinance Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

600+
Audits Completed

$15B
Secured

600K
Lines of Code Audited

Follow Our Journey

Audit Report
October, 2022

For

ØSPACE

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com