verichains

SECURITY AUDIT OF

SPACEPI SMART CONTRACT

Public Report

Apr 17, 2023

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Apr 17, 2023. We would like to thank the SpacePi for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the SpacePi Smart Contract. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified 3 vulnerable issues in the contract code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY 6

1.1. About SpacePi Smart Contract 6

1.2. Audit scope 6

1.3. Audit methodology 6

1.4. Disclaimer 7

2. AUDIT RESULT 8

2.1. Overview 8

2.1.1. ERC721Distributor.sol 8

2.1.2. NFTMiner.sol 8

2.1.3. SpacePiETHDistributor.sol 8

2.1.4. SpacePiToken.sol 8

2.1.5. StakeSpacePi.sol 9

2.2. Findings 9

2.2.1. StakeSpacePi.sol - The value of user.rewardDebt is incremented twice in `deposit()` function CRITICAL 9

2.2.2. StakeSpacePi.sol - Users may lose tokens if they withdraw before claiming CRITICAL 10

2.2.3. SpacePiETHDistributor.sol - Access control vulnerability in the `setLimit()` function CRITICAL 12

2.3. Additional notes and recommendations 12

2.3.1. StakeSpacePi.sol, ERC721Distributor.sol - Missing file and function INFORMATIVE 12

2.3.2. StakeSpacePi.sol - pre-multiply and divide later INFORMATIVE 13

2.3.3. StakeSpacePi.sol, ERC721Distributor.sol, SpacePiETHDistributor.sol, NFTMiner.sol - Redundancy of the SafeMath library INFORMATIVE 13

2.3.4. StakeSpacePi.sol - Redundancy of `&&!user.claimed` in the `claim()` function INFORMATIVE 14

2.3.5. StakeSpacePi.sol - `user.settledBlock` should only be less than or equal to `pool.lockBlocks` INFORMATIVE 14

2.3.6. ERC721Distributor.sol - Unused `tokens` variable INFORMATIVE 15

2.3.7. NFTMiner.sol - The approval for a wallet with the maximum amount of tokens INFORMATIVE 15

2.3.8. NFTMiner.sol - Idempotence in the `setPause()` function INFORMATIVE 15

2.3.9. NFTMiner.sol - The `setMultLP()` function should only be called once INFORMATIVE 16

2.3.10. NFTMiner.sol - The `replaceMultLP()` function should set the approval for the old `multLpChef` address to 0 INFORMATIVE 16

3. VERSION HISTORY 17

1. MANAGEMENT SUMMARY

1.1. About SpacePi Smart Contract

As a blockchain game supported by the concept of NFT + Metaverse, SpacePi includes several core sections in terms of platform ecological construction. NFT game system Metaverse game ecology SpacePi Exchange

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the SpacePi Smart Contract.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
c376c18f3e23d5fb6de017708804adf23200621bd56e84b6531f487dda0a0191	StakeSpacePi.sol
c63338267b9f912d58951082ba6c93697da2c88cd3b6e9cfb6ac0c00b1b170f7	SpacePiETHDistributor.sol
c593f1d13d1a5e79d4b4b6c322bf3c85d0daeaba0ec79ab9cb26a758bf538493	ERC721Distributor.sol
7a06964e17bfb7bc278cc19734f99bbb939f351f4dc594a0060fd06bae16972e	NFTMiner.sol
df83b8b06cb3aa101a23944232448d4232aa60f8dc44bd0c4e0f113b55c4ec81	SpacePiToken.sol

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- 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 were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence

- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The SpacePi Smart Contract was written in `Solidity` language, with the required version to be `^0.8.0`. The source code was written based on OpenZeppelin's library.

2.1.1. ERC721Distributor.sol

The `ERC721Distributor` contract extends `Ownable`, `ReentrancyGuard` and `Relationship` contracts. With `Ownable`, by default, the contract Owner is the contract deployer, but he can transfer ownership to another address at any time. With `Relationship`, users can specify who invited them.

Only invited users are allowed to claim NFTs until the maximum limit is reached. Additionally, if the user passes `isBlack` as `true` in the `claim()` function, 90% of the tokens will be transferred to the wallet with the address 'dead', otherwise, they will be transferred to the address designated by the owner, and the remaining 10% will be transferred to the inviter.

2.1.2. NFTMiner.sol

The `NFTMiner` contract extends `Ownable`, `ERC20` contracts. With `Ownable`, by default, the contract Owner is the contract deployer, but he can transfer ownership to another address at any time. The contract has basic functions including:

- deposit: to receive interest in the form of ERC20 tokens, and it may also have the option to receive `multLpToken`.
- withdraw: users can withdraw any amount they want.
- emergencyWithdraw: to withdraw all tokens back.

Users can only perform these functions when the contract is not paused and can burn a certain amount of tokens in order to mint an NFT.

2.1.3. SpacePiETHDistributor.sol

The `SpacePiETHDistributor` contract extends `Ownable` contract. With `Ownable`, by default, the contract Owner is the contract deployer, but he can transfer ownership to another address at any time. Only users permitted by the owner are allowed to claim.

2.1.4. SpacePiToken.sol

The `SpacePiToken` contract extends `ERC20` contracts. Upon deployment, a specified amount of tokens will be minted to the address designated by the deployer. Users can burn their own tokens.

2.1.5. StakeSpacePi.sol

The `StakeSpacePi` contract extends `ReentrancyGuard` and `Relationship` contracts. During deployment, there are 3 different pools with different APRs and lock-up periods added. There are 3 main functions:

- deposit: Users can invest and earn interest. Users can deposit additional funds at any time as long as the lock-up period has not expired (at this point, the contract will record its debt to the user and start over). If the lock-up period has expired, users should withdraw.
- withdraw: Users wait until the lock-up period has expired and then withdraw their entire investment along with the profits.
- claim: Users can claim their interest, as long as they are still within the lock-up period, they can claim multiple times. If the lock-up period has expired, users should withdraw their funds, as they will not be able to claim again until they withdraw.

2.2. Findings

During the audit process, the audit team found 3 vulnerabilities in the given version of SpacePi Smart Contract.

2.2.1. StakeSpacePi.sol - The value of user.rewardDebt is incremented twice in `deposit()` function CRITICAL

When a user makes their second deposit, the value of `user.rewardDebt` is first incremented in the `pending()` function, and then incremented again in the `deposit()` function.

```
function deposit(uint256 pid, uint256 amount) external nonReentrant onlyInvited(msg.sender)
inDuration {
    Pool storage pool = pools[pid];
    UserInfo storage user = userInfo[msg.sender][pid];
    token.transferFrom(msg.sender, address(this), amount);
    uint256 reward = pending(pid, msg.sender);
    uint256 currentBlock = block.number;
    if (user.enterBlock == 0) {
        user.enterBlock = block.number;
    }
    if (currentBlock > user.enterBlock.add(pool.lockBlocks)) {
        if (reward > 0) revert("deposit: reward claim first");
        user.enterBlock = block.number;
    }

    if (user.amount > 0) {
        if (reward > 0) {
            user.rewardDebt = user.rewardDebt.add(reward);
            user.settledBlock = block.number.sub(user.enterBlock);
        }
```

```
    }
    pool.amount = pool.amount.add(amount);
    user.amount = user.amount.add(amount);
    accDeposit = accDeposit.add(amount);
    emit Deposit(msg.sender, pid, amount);
}

function pending(uint256 pid, address play) public view returns (uint256){
    uint256 time = block.number;
    Pool memory pool = pools[pid];
    UserInfo memory user = userInfo[play][pid];
    if (user.amount == 0) return 0;
    uint256 perBlock = user.amount.mul(pool.apr).div(365 days).div(100).mul(perBlockTime);
    if (time >= pool.lockBlocks.add(user.enterBlock)) {
        if (user.settledBlock >= pool.lockBlocks) return 0;
        return perBlock.mul(pool.lockBlocks.sub(user.settledBlock)).add(user.rewardDebt);
    }
    return
perBlock.mul(time.sub(user.enterBlock).sub(user.settledBlock)).add(user.rewardDebt);
}
```

RECOMMENDATION

Do not add additional `rewardDebt` in the `deposit()` function.

```
function deposit(uint256 pid, uint256 amount) external nonReentrant onlyInvited(msg.sender)
inDuration {
    ...
    if (user.amount > 0) {
        if (reward > 0) {
            user.rewardDebt = reward;
            user.settledBlock = block.number.sub(user.enterBlock);
        }
    }
    ...
}
```

UPDATES

- *Apr 12, 2023*: This issue has been acknowledged and fixed by the SpacePi team.

2.2.2. StakeSpacePi.sol - Users may lose tokens if they withdraw before claiming CRITICAL

If a user withdraws before claiming, the `user.amount` will be 0, resulting in 0 rewards calculated at the time of claiming.

```
function withdraw(uint256 pid) external nonReentrant onlyUnLock(pid, msg.sender) {
    UserInfo storage user = userInfo[msg.sender][pid];
    Pool storage pool = pools[pid];
```

```
    uint256 amount = user.amount;
    require(user.amount >= 0, "withdraw: Principal is zero");
    user.amount = 0;
    user.enterBlock = 0;
    user.settledBlock = 0;
    user.claimed = false;
    accDeposit = accDeposit.sub(amount);
    pool.amount = pool.amount.sub(amount);
    token.transfer(msg.sender, amount);
    emit Withdraw(msg.sender, pid, amount);
}

// @dev claim interest, not locking withdraw
// @dev inviter will get setting percent of the interest
function claim(uint256 pid) external nonReentrant{
    UserInfo storage user = userInfo[msg.sender][pid];
    Pool memory pool = pools[pid];
    uint256 reward = pending(pid, msg.sender);
    require(reward > 0, "claim: interest is zero");
    require(!user.claimed, "claim: time ends claimed");
    if (token.balanceOf(address(this)).sub(accDeposit) >= reward&&!user.claimed) {
        address inviter = getParent(msg.sender);

        uint256 userInviteReward = reward.mul(inviteRewardRate).div(100);
        uint256 userReward = reward.sub(userInviteReward);
        token.transfer(inviter, userInviteReward);
        token.transfer(msg.sender, userReward);
        if (user.enterBlock.add(pool.lockBlocks) < block.number) user.claimed = true;
        user.accReward = user.accReward.add(userReward);
        user.settledBlock = block.number.sub(user.enterBlock);
        user.rewardDebt = 0;
        accReward = accReward.add(reward);
        inviteReward[inviter] = inviteReward[inviter].add(userInviteReward);
        emit InviterReward(inviter, pid, userInviteReward);
        emit Reward(msg.sender, pid, userReward);
    }
}
```

RECOMMENDATION

The team should review the logic of the code.

UPDATES

- *Apr 12, 2023*: This issue has been acknowledged and fixed by the SpacePi team.

2.2.3. SpacePiETHDistributor.sol - Access control vulnerability in the `setLimit()` function CRITICAL

Although the contract extends the `Ownable` contract, it does not use the `onlyOwner()` modifier in the `setLimit()` function, which allows users to arbitrarily change the amount of tokens they can claim.

```
function setLimit(uint256 newLimit) external {
    perUserClaimLimit = newLimit;
}
function claim(uint256 index,bytes memory signature)
public
virtual
{
    address account = msg.sender;
    if (isClaimed(index)) revert AlreadyClaimed();

    // Verify the merkle proof.
    bytes memory node = abi.encodePacked(index, account);
    address signer = ECDSA.recover(ECDSA.toEthSignedMessageHash(node), signature);
    if (signer != owner()) revert InvalidSignature();

    // Mark it claimed and send the token.
    _setClaimed(index);
    IERC20(token).safeTransfer(account, perUserClaimLimit);

    emit Claimed(index, account, perUserClaimLimit);
}
```

RECOMMENDATION

Use the `onlyOwner()` modifier in the `setLimit()` function.

UPDATES

- *Apr 17, 2023*: This issue has been acknowledged and fixed by the SpacePi team.

2.3. Additional notes and recommendations

2.3.1. StakeSpacePi.sol, ERC721Distributor.sol - Missing file and function INFORMATIVE

The file `./utils/Relationship.sol` and the modifier `inDuration` are imported and used, but no code logic is provided.

UPDATES

- *Apr 12, 2023*: This issue has been acknowledged and fixed by the SpacePi team.

2.3.2. StakeSpacePi.sol - pre-multiply and divide later INFORMATIVE

The "pre-multiplication and post-division" technique in Solidity is used to avoid loss of accuracy due to the behavior of integer division in arithmetic operations on unsigned integers. When dividing two unsigned integers, the result is rounded down (floor division), which can result in loss of accuracy in the result. To avoid this, one can perform multiplication before and after the division operation, thereby ensuring the accuracy of the result in Solidity.

```
function pending(uint256 pid, address play) public view returns (uint256){
    uint256 time = block.number;
    Pool memory pool = pools[pid];
    UserInfo memory user = userInfo[play][pid];
    if (user.amount == 0) return 0;
    uint256 perBlock = user.amount.mul(pool.apr).div(365 days).div(100).mul(perBlockTime);
    if (time >= pool.lockBlocks.add(user.enterBlock)) {
        if (user.settledBlock >= pool.lockBlocks) return 0;
        return perBlock.mul(pool.lockBlocks.sub(user.settledBlock)).add(user.rewardDebt);
    }
    return
perBlock.mul(time.sub(user.enterBlock).sub(user.settledBlock)).add(user.rewardDebt);
}
```

UPDATES

- *Apr 12, 2023*: This issue has been acknowledged and fixed by the SpacePi team.

2.3.3. StakeSpacePi.sol, ERC721Distributor.sol, SpacePiETHDistributor.sol, NFTMiner.sol - Redundancy of the SafeMath library INFORMATIVE

Starting from Solidity version 8.0 and above, the SafeMath library is no longer necessary in Solidity source code, as Solidity has integrated built-in safety checks for arithmetic operations on unsigned integers and floating-point numbers. Previously, the use of the SafeMath library was recommended in Solidity to prevent integer overflow or underflow errors in arithmetic operations.

RECOMMENDATION

Remove the SafeMath library.

UPDATES

- *Apr 12, 2023*: This issue has been acknowledged by the SpacePi team.

2.3.4. StakeSpacePi.sol - Redundancy of `&&!user.claimed` in the `claim()` function INFORMATIVE

Since `user.claimed` has been checked earlier at line 147, it is not necessary to include it in the if condition at line 148. Instead, a condition should be added: `require(token.balanceOf(address(this)).sub(accDeposit) >= reward, "insufficient tokens for rewards")` to notify users when the contract does not have enough funds to reward them.

```
function claim(uint256 pid) external nonReentrant{
    UserInfo storage user = userInfo[msg.sender][pid];
    Pool memory pool = pools[pid];
    uint256 reward = pending(pid, msg.sender);
    require(reward > 0, "claim: interest is zero");
    require(!user.claimed, "claim: time ends claimed");
    if (token.balanceOf(address(this)).sub(accDeposit) >= reward&&!user.claimed) {
        address inviter = getParent(msg.sender);

        uint256 userInviteReward = reward.mul(inviteRewardRate).div(100);
        uint256 userReward = reward.sub(userInviteReward);
        token.transfer(inviter, userInviteReward);
        token.transfer(msg.sender, userReward);
        if (user.enterBlock.add(pool.lockBlocks) < block.number) user.claimed = true;
        user.accReward = user.accReward.add(userReward);
        user.settledBlock = block.number.sub(user.enterBlock);
        user.rewardDebt = 0;
        accReward = accReward.add(reward);
        inviteReward[inviter] = inviteReward[inviter].add(userInviteReward);
        emit InviterReward(inviter, pid, userInviteReward);
        emit Reward(msg.sender, pid, userReward);
    }
}
```

UPDATES

- *Apr 12, 2023*: The redundancy has been removed by the SpacePi team.

2.3.5. StakeSpacePi.sol - `user.settledBlock` should only be less than or equal to `pool.lockBlocks` INFORMATIVE

Since the meaning of `settledBlock` is the number of blocks already staked, it must always be less than `pool.lockBlocks`.

```
function claim(uint256 pid) external nonReentrant{
    UserInfo storage user = userInfo[msg.sender][pid];
    Pool memory pool = pools[pid];
    uint256 reward = pending(pid, msg.sender);
    require(reward > 0, "claim: interest is zero");
    require(!user.claimed, "claim: time ends claimed");
    if (token.balanceOf(address(this)).sub(accDeposit) >= reward&&!user.claimed) {
```

```
        address inviter = getParent(msg.sender);

        uint256 userInviteReward = reward.mul(inviteRewardRate).div(100);
        uint256 userReward = reward.sub(userInviteReward);
        token.transfer(inviter, userInviteReward);
        token.transfer(msg.sender, userReward);
        if (user.enterBlock.add(pool.lockBlocks) < block.number) user.claimed = true;
        user.accReward = user.accReward.add(userReward);
        user.settledBlock = block.number.sub(user.enterBlock);
        user.rewardDebt = 0;
        accReward = accReward.add(reward);
        inviteReward[inviter] = inviteReward[inviter].add(userInviteReward);
        emit InviterReward(inviter, pid, userInviteReward);
        emit Reward(msg.sender, pid, userReward);
    }
}
```

RECOMMENDATION

`settledBlock` must be calculated as `min(block.number.sub(user.enterBlock), pool.lockBlocks)`.

UPDATES

- *Apr 12, 2023*: This issue has been acknowledged and fixed by the SpacePi team.

2.3.6. ERC721Distributor.sol - Unused `tokens` variable INFORMATIVE

The variable `tokens` is declared at line 38 but not used.

RECOMMENDATION

Remove if not in use.

UPDATES

- *Apr 12, 2023*: This issue has been acknowledged and fixed by the SpacePi team.

2.3.7. NFTMiner.sol - The approval for a wallet with the maximum amount of tokens INFORMATIVE

The contract is currently approving the address `multLpChef` with the maximum amount of tokens, allowing `multLpChef` to freely use this approved token amount.

2.3.8. NFTMiner.sol - Idempotence in the `setPause()` function INFORMATIVE

If there is confusion, and the `setPause()` function is called twice in a row, the result of this function will not change.

RECOMMENDATION

Add a boolean parameter `_isPause` to the `setPause()` function.

```
function setPause(bool _isPause) public onlyOwner {
        paused = _isPause;
    }
```

2.3.9. NFTMiner.sol - The `setMultLP()` function should only be called once INFORMATIVE

Because the logic in the `setMultLP()` function is already included in the `replaceMultLP()` function, calling `setMultLP()` for the second time will not approve the existing MultLP tokens for the new wallet. On the other hand, with the `replaceMultLP()` function, MultLP tokens can be removed and added back. Therefore, `setMultLP()` should only be called once.

RECOMMENDATION

Add a require statement for `multLpToken == 0x0 && multLpChef == 0x0` to `setMultLP()` function.

2.3.10. NFTMiner.sol - The `replaceMultLP()` function should set the approval for the old `multLpChef` address to 0 INFORMATIVE

After replacing the `multLpChef` address in the `replaceMultLP()` function, it is still approving a large amount of tokens to the old `multLpChef` address. Therefore, it is recommended to set the approval for the old `multLpChef` address to 0.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Apr 17,2023*	Public Report	Verichains Lab

Table 2. Report versions history