SECURITY AUDIT OF

SPACESIP STAKING SMART CONTRACTS

Public Report

Nov 06, 2021

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.
Binance Smart Chain	This dual-chain architecture will empower its users to build their decentralized apps and digital assets on one blockchain and take advantage of the fast trading to exchange on the other: EVM Compatible, Proof of Staked Authority, Cross-Chain Transfer.
BNB	A cryptocurrency whose blockchain is generated by the Binance Smart Chain platform. Matic is used for payment of transactions and computing services in the Binance Smart Chain 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 prepared by Verichains Lab on Nov 06, 2021. We would like to thank the SpaceSIP Team 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 SpaceSIP Staking Smart Contracts. 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 no vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About SpaceSIP Staking Smart Contracts

Space SIP (SIP) is a Play to Earn NFT RPG developed on the Binance Smart Chain platform. The game revolves around the acquisition of a legendary Spaceship and powerful Weapon to wield them. Players can send Spaceship to mine $SIP tokens. Players may participate in combat using their assets to earn $SIP tokens. Assets are player owned NFTs minted in the ERC-721 standard which may be traded on the properietary marketplace.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of SpaceSIP Staking. It was conducted on the source code provided by the SpaceSIP team.

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 was 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 initial review was conducted on Nov 04, 2021 and a total effort of 3 working days was dedicated to identifying and documenting security issues in the code base of the SpaceSIP Staking Smart Contracts.

2.2. Findings

The SpaceSIP Staking Smart Contracts was written in Solidity language, with the required version to be ^0.8.0.

The audit team found no vulnerability in the given version of SpaceSIP Staking Smart Contracts.

2.3. Additional information and recommendations

2.3.1. Transfer error when vault contract does not have enough token in balance INFORMATIVE

In the withdraw function, after calculating the reward for depositing, the contract calls vault contract to transfer totalbonus for the user. If the balance of vault is lower than totalbonus, the transaction will be reverted.

```
function withdraw(uint256 _pId) external {
        UserInfo storage _userInfo = userInfo[_pId][_msgSender()];
        PoolInfo storage _poolInfo = poolInfo[_pId];

        require(_userInfo.totalAmount > 0, "S::W:AE"); // amount is empty

        (, uint256 totalFeeAmount, uint256 totalBonus, uint256 claimableAmount, ) = _getOpenDeposit(_pId, _msgSender());
        feeCharged += totalFeeAmount;
        delete _userInfo.deposits;

        _poolInfo.totalAmount -= _userInfo.totalAmount;
        _userInfo.totalAmount = 0;

        if (totalBonus > 0) {
            vault.transfer(_msgSender(), totalBonus);
        }
        sip.transfer(_msgSender(), claimableAmount);
        emit __withdraw(_msgSender(), _pId, claimableAmount);
    }
```

Snippet 1. Error transferring if not enough balance in vault contract

RECOMMENDATION

We suggest splitting the withdraw function into 2 functions: one for claiming stake and the other for claiming totalbonus.

UPDATES

- *2021-11-05*: This issue has been acknowledged by the SpaceSIP Team.

2.3.2. Redundant variable updates inside the for-loop INFORMATIVE

There are two statements that update variables with unchanged values inside the for-loop at lines 264 and 271.

```
243  function _getOpenDeposit(uint256 _pId, address _user)
244          internal
245          view
246          returns (
247              uint256 totalDepositAmount,
248              uint256 totalFeeAmount,
249              uint256 totalBonus,
250              uint256 claimableAmount,
251              uint256 claimableTime
252          )
253      {
254          UserInfo storage _userInfo = userInfo[_pId][_user];
255          PoolInfo storage _poolInfo = poolInfo[_pId];
256          totalDepositAmount = _userInfo.totalAmount;
257          uint256 _withdrawFeeAmount = 0;
258          uint256 _earlyWithdrawFeeAmount = 0;
259
260          for (uint256 i = 0; i < _userInfo.deposits.length; i++) {
261              Deposit storage _deposit = _userInfo.deposits[i];
262
263              uint256 _endTime = block.timestamp;
264              if (block.timestamp < (_deposit.joinTime + _poolInfo.loc…
     kDuration)) {
265                  _earlyWithdrawFeeAmount += (_deposit.amount * earlyW…
     ithdrawFee) / 10000;
```

```
            } else {
                _withdrawFeeAmount += (_deposit.amount * _deposit.withdrawFee) / 10000;
            }

            if (_endTime > _poolInfo.closeTime) _endTime = _poolInfo.closeTime;

            // calculate bonus
            totalBonus +=
                (_deposit.amount *
                    (((_endTime - _deposit.joinTime) * _poolInfo.apr) / (12 * kDefaultOneMonthInSeconds))) /
                10000;
        }
        totalFeeAmount = _withdrawFeeAmount + _earlyWithdrawFeeAmount;
        claimableAmount = _userInfo.totalAmount - totalFeeAmount;
        claimableTime = (_userInfo.deposits.length > 0)
            ? _userInfo.deposits[_userInfo.deposits.length - 1].joinTime + _poolInfo.lockDuration
            : 0;
    }
```

Snippet 2. Unless update variables inside the for-loop

RECOMMENDATION

We suggest moving those statements above for-loop for readability and gas saving like the code below.

```
function _getOpenDeposit(uint256 _pId, address _user)
        internal
        view
        returns (
            uint256 totalDepositAmount,
            uint256 totalFeeAmount,
            uint256 totalBonus,
            uint256 claimableAmount,
            uint256 claimableTime
        )
    {
        UserInfo storage _userInfo = userInfo[_pId][_user];
        PoolInfo storage _poolInfo = poolInfo[_pId];
        totalDepositAmount = _userInfo.totalAmount;

        uint256 _withdrawFeeAmount = 0;
        uint256 _earlyWithdrawFeeAmount = 0;

        uint256 _endTime = block.timestamp;
        if (_endTime > _poolInfo.closeTime) _endTime = _poolInfo.closeTime;

        for (uint256 i = 0; i < _userInfo.deposits.length; i++) {
            Deposit storage _deposit = _userInfo.deposits[i];

            if (block.timestamp < (_deposit.joinTime + _poolInfo.lockDuration)) {
                _earlyWithdrawFeeAmount += (_deposit.amount * earlyWithdrawFee) / 10000;
            } else {
                _withdrawFeeAmount += (_deposit.amount * _deposit.withdrawFee) / 10000;
            }

            // calculate bonus
            totalBonus +=
                (_deposit.amount *
                    (((_endTime - _deposit.joinTime) * _poolInfo.apr) / (12 * kDefaultOneMonthInSeconds))) /
                10000;
        }

        totalFeeAmount = _withdrawFeeAmount + _earlyWithdrawFeeAmount;
        claimableAmount = _userInfo.totalAmount - totalFeeAmount;
        claimableTime = (_userInfo.deposits.length > 0)
            ? _userInfo.deposits[_userInfo.deposits.length - 1].joinTime + _poolInfo.lockDuration
            : 0;
    }
```

UPDATES

- *2021-11-05*: SpaceSIP Team has fixed the issue at commit 4e5d0b7c8003039cfaaec629aad554936a3e5012.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*2021-11-06*	Public Report	Verichains Lab

Table 2. Report versions history