Smart Contract
Security Audit Report

The SlowMist Security Team received the team's application for smart contract security audit of the Staked STEP.APP on 2022.05.04. The following are the details and results of this smart contract security audit:

Token Name :

Staked STEP.APP

File name and hash (SHA256) :

stepapp-staking-contracts.zip: 1bc35856bb45ccba4cd1cf88b49241d924b4bca257b444940564f25bacfe3fd0

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

NO.	Audit Items	Result
1	Replay Vulnerability	Passed
2	Denial of Service Vulnerability	Passed
3	Race Conditions Vulnerability	Passed
4	Authority Control Vulnerability	Passed
5	Integer Overflow and Underflow Vulnerability	Passed
6	Gas Optimization Audit	Passed
7	Design Logic Audit	Passed
8	Uninitialized Storage Pointers Vulnerability	Passed
9	Arithmetic Accuracy Deviation Vulnerability	Passed
10	"False top-up" Vulnerability	Passed
11	Malicious Event Log Audit	Passed
12	Scoping and Declarations Audit	Passed

NO.	Audit Items	Result
13	Safety Design Audit	Passed
14	Non-privacy/Non-dark Coin Audit	Passed

Audit Result : Passed

Audit Number : 0X002205060001

Audit Date : 2022.05.04 - 2022.05.06

Audit Team : SlowMist Security Team

Summary conclusion : This is a token contract that contains the staking section. The total amount of contract tokens can be changed. Users can mint stFITFI tokens by staking FITFI tokens and burn stFITFI tokens by redeeming FITFI tokens. The contract does not have the Overflow and the Race Conditions issue.

During the audit, we found the following information:

1. The token can not be transferred.
2. The owner role can set a new penalty days value and the maximum value is 90.
3. The owner role can set a new penalty base points value and the maximum value is 50%.
4. The owner role can set a new penalty treasury.
5. The owner role can set a new share bonus base points value for 1M staked tokens.
6. After communication with the project team, the shares are calculated in the contract for off-chain logic and used in future distribution reward contract. In the current contract there are only calculation and saving to the state, and the stakingtoken is the FITFI contract, without rebase logic.

The source code:

```
// SPDX-License-Identifier: MIT
//SlowMist// The contract does not have the Overflow and the Race
pragma solidity =0.8.4;

import "@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol";
```

```
import "@openzeppelin/contracts-
upgradeable/token/ERC20/extensions/ERC20VotesCompUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";

contract StepAppStaking is ERC20VotesCompUpgradeable, OwnableUpgradeable,
ReentrancyGuardUpgradeable {
    using SafeERC20Upgradeable for IERC20Upgradeable;

    /// @dev Max base points
    uint256 public constant MAX_BPS = 1e4;

    /// @dev Max penalty days
    uint256 private constant PENALTY_DAYS_LIMIT = 90;

    /// @dev Max penalty base points
    uint256 private constant PENALTY_BP_LIMIT = 0.5 * 1e4;

    /// @dev Staking token
    IERC20Upgradeable public stakingToken;

    /// @dev Info about each stake by user address
    mapping(address => Stake[]) public stakers;

    /// @dev Penalty days value
    uint16 public penaltyDays;

    /// @dev Penalty base points value
    uint16 public penaltyBP;

    /// @dev The address to which the penalty tokens will be transferred
    address public treasury;

    /// @dev Total shares
    uint192 public totalShares;

    /// @dev Share bonus base points value for 1M staked tokens
    uint32 public shareBonusBPPer1MTokens;

    struct Stake {
        bool unstaked;
        uint128 amount;
        uint48 stakedTimestamp;
        uint16 penaltyDays;
```

```
        uint16 penaltyBP;
        uint192 shares;
    }

    event Staked(
        address indexed staker,
        uint256 indexed stakeIndex,
        uint128 amount,
        uint48 stakedTimestamp,
        uint16 penaltyDays,
        uint16 penaltyBP,
        uint128 totalSupply,
        uint192 shares,
        uint192 totalShares
    );

    event Unstaked(
        address indexed staker,
        uint256 indexed stakeIndex,
        uint128 amount,
        uint128 penaltyAmount,
        uint128 totalSupply,
        uint192 shares,
        uint192 totalShares
    );

    event SetPenaltyDays(uint16 penaltyDays);
    event SetPenaltyBP(uint16 penaltyBP);
    event SetTreasury(address treasury);
    event SetShareBonusBPPer1MTokens(uint32 shareBonusBPPer1MTokens);

    /**
     * @notice Initializer
     * @param _stakingToken Staking token address
     * @param _penaltyDays Penalty days value
     * @param _penaltyBP Penalty base points value
     * @param _treasury The address to which the penalty tokens will be transferred
     * @param _shareBonusBPPer1MTokens Share bonus base points value for 1M staked tokens
     */
    function initialize(
        IERC20Upgradeable _stakingToken,
        uint16 _penaltyDays,
        uint16 _penaltyBP,
        address _treasury,
```

```
        uint32 _shareBonusBPPer1MTokens
    ) external virtual initializer {
        __ERC20_init("Staked STEP.APP", "stFITFI");
        __ERC20Permit_init("Staked STEP.APP");
        __Ownable_init();
        __ReentrancyGuard_init();

        require(address(_stakingToken) != address(0), "StepAppStaking: staking token is the
zero address");
        require(_penaltyDays <= PENALTY_DAYS_LIMIT, "StepAppStaking: penalty days exceeds
limit");
        require(_penaltyBP <= PENALTY_BP_LIMIT, "StepAppStaking: penalty BP exceeds limit");
        require(_treasury != address(0), "StepAppStaking: treasury is the zero address");

        stakingToken = _stakingToken;
        penaltyDays = _penaltyDays;
        penaltyBP = _penaltyBP;
        treasury = _treasury;
        shareBonusBPPer1MTokens = _shareBonusBPPer1MTokens;
    }

    /**
     * @notice Stake staking tokens
     * @param _amount amount to stake
     */
    function stake(uint128 _amount) external nonReentrant {
        stakingToken.safeTransferFrom(msg.sender, address(this), _amount);

        _mint(msg.sender, _amount);

        uint192 shares = calculateShares(_amount);
        totalShares += shares;
        stakers[msg.sender].push(Stake(
            false,
            _amount,
            uint48(block.timestamp),
            penaltyDays,
            penaltyBP,
            shares
        ));

        uint256 stakeIndex = stakers[msg.sender].length - 1;
        emit Staked(msg.sender, stakeIndex, _amount, uint48(block.timestamp), penaltyDays,
penaltyBP, uint128(totalSupply()), shares, totalShares);
```

```
    }

    /**
     * @notice Unstake staking tokens
     * @notice If penalty period is not over grab penalty
     * @param _stakeIndex Stake index in array of user's stakes
     */
    function unstake(uint256 _stakeIndex) external nonReentrant {
        require(_stakeIndex < stakers[msg.sender].length, "StepAppStaking: invalid index");
        Stake storage stakeRef = stakers[msg.sender][_stakeIndex];
        require(!stakeRef.unstaked, "StepAppStaking: unstaked already");

        _burn(msg.sender, stakeRef.amount);
        totalShares -= stakeRef.shares;
        stakeRef.unstaked = true;

        // pays a penalty if unstakes during the penalty period
        uint256 penaltyAmount = 0;
        if (stakeRef.stakedTimestamp + uint48(stakeRef.penaltyDays) * 86400 > block.timestamp)
{
            penaltyAmount = stakeRef.amount * stakeRef.penaltyBP / MAX_BPS;
            stakingToken.safeTransfer(treasury, penaltyAmount);
        }

        stakingToken.safeTransfer(msg.sender, stakeRef.amount - penaltyAmount);

        emit Unstaked(msg.sender, _stakeIndex, stakeRef.amount, uint128(penaltyAmount),
uint128(totalSupply()), stakeRef.shares, totalShares);
    }

    /**
     * @notice Return a length of stake's array by user address
     * @param _stakerAddress User address
     */
    function stakerStakeCount(address _stakerAddress) external view returns (uint256) {
        return stakers[_stakerAddress].length;
    }

    /**
     * @notice Return shares for the staked amount
     * @dev Share bonus percentage doubles every 1M tokens for the entire amount. Value of 18
for token decimals
     * @param _amount Amount to calculate
     * @return shares Calculated shares for this amount
```

```
     */
    function calculateShares(uint256 _amount) public view returns (uint192 shares) {
        uint256 stakingMoreBonus = _amount * _amount * shareBonusBPPer1MTokens / 1e24 /
MAX_BPS;
        shares = uint192(_amount + stakingMoreBonus);
    }

    // ** ONLY OWNER **

    /**
     * @notice Set a new penalty days value
     * @param _penaltyDays New penalty days value
     */
    function setPenaltyDays(uint16 _penaltyDays) external onlyOwner {
        require(_penaltyDays <= PENALTY_DAYS_LIMIT, "StepAppStaking: penalty days exceeds
limit");
        penaltyDays = _penaltyDays;
        emit SetPenaltyDays(_penaltyDays);
    }

    /**
     * @notice Set a new penalty base points value
     * @param _penaltyBP New penalty base points value
     */
    function setPenaltyBP(uint16 _penaltyBP) external onlyOwner {
        require(_penaltyBP <= PENALTY_BP_LIMIT, "StepAppStaking: penalty BP exceeds limit");
        penaltyBP = _penaltyBP;
        emit SetPenaltyBP(_penaltyBP);
    }

    /**
     * @notice Set a new penalty treasury
     * @param _treasury New treasury address
     */
    function setTreasury(address _treasury) external onlyOwner {
        require(_treasury != address(0), "StepAppStaking: treasury is the zero address");
        treasury = _treasury;
        emit SetTreasury(_treasury);
    }

    /**
     * @notice Set a share bonus base points value for 1M staked tokens
     * @param _shareBonusBPPer1MTokens New share bonus base points value for 1M staked tokens
     */
```

```
    function setShareBonusBPPer1MTokens(uint32 _shareBonusBPPer1MTokens) external onlyOwner {
        shareBonusBPPer1MTokens = _shareBonusBPPer1MTokens;
        emit SetShareBonusBPPer1MTokens(_shareBonusBPPer1MTokens);
    }

    // ** INTERNAL **

    /// @dev disable transfers
    function _transfer(address _from, address _to, uint256 _amount) internal override {
        revert("StepAppStaking: NON_TRANSFERABLE");
    }
}
```

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