Smart Contract Security Audit Report

The SlowMist Security Team received the CTSI team's application for smart contract security audit of the Staking on Nov. 04, 2020. The following are the details and results of this smart contract security audit:

Porject name :

CTSI

File name and HASH(SHA256) :

RewardManager.sol:

55193a0e46abb87818807e08352a27d2795651a65011b1114c8191995fa8ecd7

Staking.sol:

76bf7f3563fbb895d176cebca7e75af6523a6f4877fd1b263fc6d8c29eac45e8

StakingImpl.sol:

d025f90ef62c96c110ad9d265325cbf07048327e2ae42cfa37e91e6cbf4eb208

The audit items and results :

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

No.	Audit Items	Audit Subclass	Audit Subclass Result
1	Overflow Audit	-	Passed
2	Race Conditions Audit	-	Passed
3	Authority Control Audit	Permission vulnerability audit	Passed
		Excessive auditing authority	Passed
4	Safety Design Audit	Zeppelin module safe use	Passed
		Compiler version security	Passed
		Hard-coded address security	Passed
		Fallback function safe use	Passed
		Show coding security	Passed
		Function return value security	Passed
		Call function security	Passed
5	Denial of Service Audit	-	Passed

6	Gas Optimization Audit	-	Passed
7	Design Logic Audit	-	Passed
8	“False Deposit” vulnerability Audit	-	Passed
9	Malicious Event Log Audit	-	Passed
10	Scoping and Declarations Audit	-	Passed
11	Replay Attack Audit	ECDSA's Signature Replay Audit	Passed
12	Uninitialized Storage Pointers Audit	-	Passed
13	Arithmetic Accuracy Deviation Audit	-	Passed

Audit Result : Passed

Audit Number : 0X002011240001

Audit Date : Nov. 24, 2020

Audit Team : SlowMist Security Team

Summary: This is the reward distribution contract and stake contract part of the CTSI project. OpenZeppelin's SafeMath security module is used, which is a recommend approach. The contract does not have the Overflow and the Race Conditions issue.

The source code:

RewardManager.sol:

```
// Copyright (C) 2020 Cartesi Pte. Ltd.

// SPDX-License-Identifier: GPL-3.0-only
// This program is free software: you can redistribute it and/or modify it under
// the terms of the GNU General Public License as published by the Free Software
// Foundation, either version 3 of the License, or (at your option) any later
```

```
// version.

// This program is distributed in the hope that it will be useful, but WITHOUT ANY
// WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
// PARTICULAR PURPOSE. See the GNU General Public License for more details.

// You should have received a copy of the GNU General Public License
// along with this program.  If not, see <https://www.gnu.org/licenses/>.

// Note: This component currently has dependencies that are licensed under the GNU
// GPL, version 3, and so you should treat this component as a whole as being under
// the GPL version 3. But all Cartesi-written code in this component is licensed
// under the Apache License, version 2, or a compatible permissive license, and can
// be used independently under the Apache v2 license. After this component is
// rewritten, the entire component will be released under the Apache v2 license.

/// @title RewardManager
/// @author Felipe Argento
```

//SlowMist// The contract does not have the Overflow and the Race Conditions issue

```
pragma solidity ^0.7.0;

import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract RewardManager {
    using SafeMath for uint256;

    uint256 minReward;
    uint256 maxReward;
    uint256 distNumerator;
    uint256 distDenominator;
    address operator;
    IERC20 ctsi;

    /// @notice Creates contract
    /// @param _operator address of the operator
    /// @param _ctsiAddress address of token instance being used
    /// @param _maxReward maximum reward that this contract pays
    /// @param _minReward minimum reward that this contract pays
    /// @param _distNumerator multiplier factor to define reward amount
```

```
    /// @param _distDenominator dividing factor to define reward amount
    constructor(
        address _operator,
        address _ctsiAddress,
        uint256 _maxReward,
        uint256 _minReward,
        uint256 _distNumerator,
        uint256 _distDenominator
    ) {

        operator = _operator;
        ctsi = IERC20(_ctsiAddress);

        minReward = _minReward;
        maxReward = _maxReward;
        distNumerator = _distNumerator;
        distDenominator = _distDenominator;
    }

    /// @notice Rewards address
    /// @param _address address be rewarded
    /// @param _amount reward
    /// @dev only the pos contract can call this
```

//SlowMist// This function will be called by the poc contract, which is not within the scope of this audit

```
    function reward(address _address, uint256 _amount) public {
        require(msg.sender == operator, "Only the operator contract can call this function");

        ctsi.transfer(_address, _amount);
    }

    /// @notice Get RewardManager's balance
    function getBalance() public view returns (uint256) {
        return ctsi.balanceOf(address(this));
    }

    /// @notice Get current reward amount
    function getCurrentReward() public view returns (uint256) {
        uint256 cReward = (getBalance().mul(distNumerator)).div(distDenominator);
        cReward = cReward > minReward? cReward : minReward;
```

```
            cReward = cReward > maxReward? maxReward : cReward;

            return cReward > getBalance()? getBalance() : cReward;
        }
}
```

Staking.sol:

```
// Copyright (C) 2020 Cartesi Pte. Ltd.

// SPDX-License-Identifier: GPL-3.0-only
// This program is free software: you can redistribute it and/or modify it under
// the terms of the GNU General Public License as published by the Free Software
// Foundation, either version 3 of the License, or (at your option) any later
// version.

// This program is distributed in the hope that it will be useful, but WITHOUT ANY
// WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
// PARTICULAR PURPOSE. See the GNU General Public License for more details.

// You should have received a copy of the GNU General Public License
// along with this program.  If not, see <https://www.gnu.org/licenses/>.

// Note: This component currently has dependencies that are licensed under the GNU
// GPL, version 3, and so you should treat this component as a whole as being under
// the GPL version 3. But all Cartesi-written code in this component is licensed
// under the Apache License, version 2, or a compatible permissive license, and can
// be used independently under the Apache v2 license. After this component is
// rewritten, the entire component will be released under the Apache v2 license.

//SlowMist// The contract does not have the Overflow and the Race Conditions issue
/// @title Interface staking contract
pragma solidity ^0.7.0;

interface Staking {

    /// @notice Returns total amount of tokens counted as stake
    /// @param _userAddress user to retrieve staked balance from
    /// @return finalized staked of _userAddress
    function getStakedBalance(
        address _userAddress) external view returns (uint256);
```

```
    /// @notice Returns the timestamp when next deposit can be finalized
    /// @return timestamp of when finalizeStakes() is callable
    function getMaturingTimestamp(address _userAddress) external view returns (uint256);

    /// @notice Returns the timestamp when next withdraw can be finalized
    /// @return timestamp of when finalizeWithdraw() is callable
    function getReleasingTimestamp(address _userAddress) external view returns (uint256);


    /// @notice Returns the balance waiting/ready to be matured
    /// @return amount that will get staked after finalization
    function getMaturingBalance(address _userAddress) external view   returns (uint256);

    /// @notice Returns the balance waiting/ready to be released
    /// @return amount that will get withdrew after finalization
    function getReleasingBalance(address _userAddress) external view   returns (uint256);


    /// @notice Deposit CTSI to be staked. The money will turn into staked
    ///             balance after timeToStake days
    /// @param _amount The amount of tokens that are gonna be deposited.
    function stake(uint256 _amount) external;

    /// @notice Remove tokens from staked balance. The money can
    ///             be released after timeToRelease seconds, if the
    ///             function withdraw is called.
    /// @param _amount The amount of tokens that are gonna be unstaked.
    function unstake(uint256 _amount) external;

    /// @notice Transfer tokens to user's wallet.
    /// @param _amount The amount of tokens that are gonna be transferred.
    function withdraw(uint256 _amount) external;

    // events
    /// @notice CTSI tokens were deposited, they count as stake after _maturationDate
    /// @param _amount amount deposited for staking
    /// @param _address address of msg.sender
    /// @param _maturationDate date when the stake can be finalized
    event Stake(
        uint256 indexed _amount,
        address indexed _address,
        uint256 indexed _maturationDate
```

```
    );

    /// @notice Unstake tokens, moving them to releasing structure
    /// @param _amount amount of tokens to be released
    /// @param _address address of msg.sender
    /// @param _maturationDate date when the tokens can be withdrew
    event Unstake(
        uint256 indexed _amount,
        address indexed _address,
        uint256 indexed _maturationDate
    );

    /// @notice Withdraw process was finalized
    /// @param _amount amount of tokens withdrawn
    /// @param _address address of msg.sender
    event Withdraw(
        uint256 indexed _amount,
        address indexed _address
    );
}
```

StakingImpl.sol:

```
// Copyright (C) 2020 Cartesi Pte. Ltd.

// SPDX-License-Identifier: GPL-3.0-only
// This program is free software: you can redistribute it and/or modify it under
// the terms of the GNU General Public License as published by the Free Software
// Foundation, either version 3 of the License, or (at your option) any later
// version.

// This program is distributed in the hope that it will be useful, but WITHOUT ANY
// WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
// PARTICULAR PURPOSE. See the GNU General Public License for more details.

// You should have received a copy of the GNU General Public License
// along with this program.  If not, see <https://www.gnu.org/licenses/>.

// Note: This component currently has dependencies that are licensed under the GNU
// GPL, version 3, and so you should treat this component as a whole as being under
// the GPL version 3. But all Cartesi-written code in this component is licensed
// under the Apache License, version 2, or a compatible permissive license, and can
```

```
// be used independently under the Apache v2 license. After this component is
// rewritten, the entire component will be released under the Apache v2 license.
```

//SlowMist// The contract does not have the Overflow and the Race Conditions issue

```
/// @title Cartesi Staking
/// @author Felipe Argento
pragma solidity ^0.7.0;

import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

import "./Staking.sol";

contract StakingImpl is Staking {
    using SafeMath for uint256;
    IERC20 private ctsi;

    uint256 timeToStake; // time it takes for deposited tokens to become staked.
    uint256 timeToRelease; // time it takes from witdraw signal to tokens to be unlocked.

    mapping(address => uint256) staked; // amount of money being staked.
    mapping(address => MaturationStruct) maturing; // deposits waiting to be staked.
    mapping(address => MaturationStruct) releasing; // money waiting for withdraw.

    struct MaturationStruct {
        uint256 amount;
        uint256 timestamp;
    }

    /// @notice constructor
    /// @param _ctsiAddress address of compatible ERC20
    /// @param _timeToStake time it takes for deposited tokens to become staked.
    /// @param _timeToRelease time it takes from unstake to tokens being unlocked.
    constructor(
        address _ctsiAddress,
        uint256 _timeToStake,
        uint256 _timeToRelease
    ) {
        ctsi = IERC20(_ctsiAddress);
        timeToStake = _timeToStake;
        timeToRelease = _timeToRelease;
```

```
    }

    function stake(uint256 _amount) public override {
        require(_amount > 0, "amount cant be zero");

        // pointers to releasing/maturing structs
        MaturationStruct storage r = releasing[msg.sender];
        MaturationStruct storage m = maturing[msg.sender];

        // check if there are mature coins to be staked
        if (m.timestamp.add(timeToStake) <= block.timestamp) {
            staked[msg.sender] = staked[msg.sender].add(m.amount);
            m.amount = 0;
        }

        // first move tokens from releasing pool to maturing
        // then transfer from wallet
        if (r.amount >= _amount) {
            r.amount = (r.amount).sub(_amount);
        } else {
            // transfer stake to contract
            // from: msg.sender
            // to: this contract
            // value: _amount - releasing[msg.sender].amount
            ctsi.transferFrom(msg.sender, address(this), _amount.sub(r.amount));
            r.amount = 0;

        }

        m.amount = (m.amount).add(_amount);

        m.timestamp = block.timestamp; //SlowMist// Repeated stake will overwrite the timestamp of the
previous stake

        emit Stake(
            m.amount,
            msg.sender,
            block.timestamp.add(timeToStake)
        );
    }
```

```
    function unstake(uint256 _amount) public override {
        require(_amount > 0, "amount cant be zero");

        // pointers to releasing/maturing structs
        MaturationStruct storage r = releasing[msg.sender];
        MaturationStruct storage m = maturing[msg.sender];

        if (m.amount >= _amount) {
            m.amount = (m.amount).sub(_amount);
        } else {
            // safemath.sub guarantees that _amount <= m.amount + staked amount
            staked[msg.sender] = staked[msg.sender].sub(_amount.sub(m.amount));

            m.amount = 0; //SlowMist// When the user unstake, the CTSI token in the maturing will be
unstake first

        }
        // update releasing amount
        r.amount = (r.amount).add(_amount);

        r.timestamp = block.timestamp; //SlowMist// Repeated unstake will overwrite the timestamp of the
previous unstake


        emit Unstake(
            r.amount,
            msg.sender,
            block.timestamp.add(timeToRelease)
        );
    }

    function withdraw(uint256 _amount) public override {
        // pointer to releasing struct
        MaturationStruct storage r = releasing[msg.sender];

        require(_amount > 0, "amount cant be zero");
        require(
            r.timestamp.add(timeToRelease) <= block.timestamp,
            "tokens are not yet ready to be released"
        );

        r.amount = (r.amount).sub(_amount, "not enough tokens waiting to be released;");
```

```
        // withdraw tokens
        // from: this contract
        // to: msg.sender
        // value: bet total withdraw value on toWithdraw
        ctsi.transfer(msg.sender, _amount);
        emit Withdraw(_amount, msg.sender);
    }

    // getters
    function getMaturingTimestamp(
        address _userAddress
    )
    public
    view override
    returns (uint256)
    {
        return maturing[_userAddress].timestamp.add(timeToStake);
    }

    function getMaturingBalance(
        address _userAddress
    )
    public
    view override
    returns (uint256)
    {
        MaturationStruct storage m = maturing[_userAddress];

        if (m.timestamp.add(timeToStake) <= block.timestamp) {
            return 0;
        }

        return m.amount;
    }

    function getReleasingBalance(
        address _userAddress
    )
    public
    view override
    returns (uint256)
```

```
    {
        return releasing[_userAddress].amount;
    }

    function getReleasingTimestamp(
        address _userAddress
    )
    public
    view override
    returns (uint256)
    {
        return releasing[_userAddress].timestamp.add(timeToRelease);
    }

    function getStakedBalance(address _userAddress)
    public
    view override
    returns (uint256)
    {
        MaturationStruct storage m = maturing[_userAddress];

        // if there are mature deposits, treat them as staked
        if (m.timestamp.add(timeToStake) <= block.timestamp) {
            return staked[_userAddress].add(m.amount);
        }

        return staked[_userAddress];
    }
}
```

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist