Beosin

成都链安

BEOSIN

Smart contract security audit report

Audit Number：202010191800

Smart Contract Name：

StakingManager

Smart Contract Address Link：

https://github.com/RAMP-DEFI/RAMP_IOST/blob/master/src/contracts/StakingManager.js

Commit Hash:

Origin Version: 76286a646c70e3195272332d30d2e60896cab3e0

Final Version: 806fc6ab48c90505d963f2e7d76805b1b0d2e080

Start Date：2020.09.21

Completion Date：2020.10.19

Overall Result: Pass

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	Redundant Code	Pass
		SafeMath Features	Pass
		Exception Usage	Pass
		Gas Consumption	Pass
		ABI Specifiers	Pass
		Update Usage	Pass
2	General Vulnerability	Integer Overflow/Underflow	Pass
		Reentrancy	Pass
		Pseudo-random Number Generator (PRNG)	Pass
		Transaction-Ordering Dependence	Pass
		DoS (Denial of Service)	Pass
		call/callWithAuth Security	Pass
		mapKeys/mapLen/globalMapkeys/globalMapLen Usage	Pass
3	Business Security	Business Logics	Pass
		Business Implementations	Pass

Note: Audit results and suggestions in code comments

Disclaimer: This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin (Chengdu LianAn) Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin (Chengdu LianAn) Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contracts StakingManager, including Coding Standards, Security, and Business Logic. **The StakingManager contract passed all audit items. The overall result is Pass. The smart contract is able to function properly.** Please find below the basic information of the smart contract:

1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Redundant Code

- Description: Check whether the contract code has redundant codes.
- Result: Pass

1.2 SafeMath Features

- Description: Check whether prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.3 Exception Usage

- Description: Check the use reasonability of exception in the contract.
- Result: Pass

1.4 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation, or whether it can be greatly reduced.
- Result: Pass

1.5 ABI Interface Definition

- Description: Check whether the ABI interface definition conforms to the contract function.
- Result: Pass

1.6 Update Usage

- Description: Check whether the *can_update* function is set correctly, including permission management and return value setting.
- Result: Pass

2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into contract and change state.
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Result: Pass

2.7 call/callWithAuth Security

- Description: Check whether the usage of functions *call/callWithAuth* have vulnerabilities.
- Result: Pass

2.8 mapKeys/mapLen/globalMapkeys/globalMapLen Usage

- Description: Check whether the fields corresponding to a key exceed 256 when the four APIs of *mapKeys/mapLen/globalMapkeys/globalMapLen* are called.
- Result: Pass

3. Business Security

Check whether the business is secure.

3.1 Update contract

- Description: The StakingManager contract supports the upgrade contract function, which can only be called by *contractOwner*.
- Related Function: *can_update*
- Result: Pass

3.2 Configure contract parameters

- Description: The StakingManager contract supports hot configuration updates. The configurable items are: team fee, minimum trade-in percentage, team fee account, trade-in account, and transfer contract whitelist. Only *contractOwner* can call this function.
- Related Function: *setConfig*
- Result: Pass

3.3 Stake

- Description: When creating a stake, the user is issued wIOST tokens that can be used later as stake in the RAMP ecosystem. The user needs to specify a valid producer, the number of stakes, the percentage used for team fee, and the Ethereum address of trade-in.
- Related Function: *stake*
- Result: Pass

3.4 Remove the stake

- Description: The user can remove the stake. When removing a stake, only the stake corresponding to the stake ID can be removed. The ID must be a valid stake that belongs to *tx.publisher*, and the same ID can only be removed once. The removing operation cannot be performed during the processing of staking rewards. When a stake is removed, the unclaimed reward corresponding to the stake ID will be automatically claimed and sent to the stake user, and then the corresponding vote will be cancelled from the contract on the corresponding producer. The corresponding stake amount will be frozen for 3 days. After 3 days, the stake user can claim this stake amount from the contract.
- Related Function: *unstake*

- Result: Pass

3.5 Process rewards

- Description: The user stakes to this contract, and the contract votes the stake amount to the producer, and then gets voting rewards. The *contractOwner* claims voting rewards into this contract by calling *processProducerBonus* function. After that, any user can call *processStakeRewards* function to increase the rewards of all producers in the data table of each user according to the corresponding coefficient. Each stake reward will be divided into three parts, team fee, trade-in, and netRewards, which are distributed according to the proportion of storage when the stake is created.

- Related Function: *processProducerBonus, processStakeRewards*

- Result: Pass

3.5 Transfer frozen withdrawals

- Description: Checks any frozen withdrawals if they are ready for release. Transfers and removes frozen withdrawal record once possible.

- Related Function: *transferFrozenWithdrawals*

- Result: Pass

3.6 Claim rewards

- Description: Users to claim all available IOST rewards.

- Related Function: *claimRewards*

- Result: Pass

3.7 Withdraw team fees

- Description: Withdraw the team fees accumulated in the contract. Only *contractOwner* can call this function.

- Related Function: *withdrawTeamFees*

- Result: Pass

3.8 Withdraw trade-in amounts

- Description: Withdraw the trade-in amounts accumulated in the contract. Only *contractOwner* can call this function.

- Related Function: *withdrawTradeIn*

- Result: Pass

3.9 Transfer wIOST

- Description: Only the contract address in the whitelist can call related functions of this contract to send wIOST. Ordinary users and other contract addresses cannot send wIOST directly.

- Related Function: *transfer, transferFreeze*

- Result: Pass

3.10 Pause

- Description: The *contractOwner* can set the pause state *PAUSED*. When *PAUSED* is set to true, the user cannot call *stake, unstake, transferFrozenWithdrawals, claimRewards*.

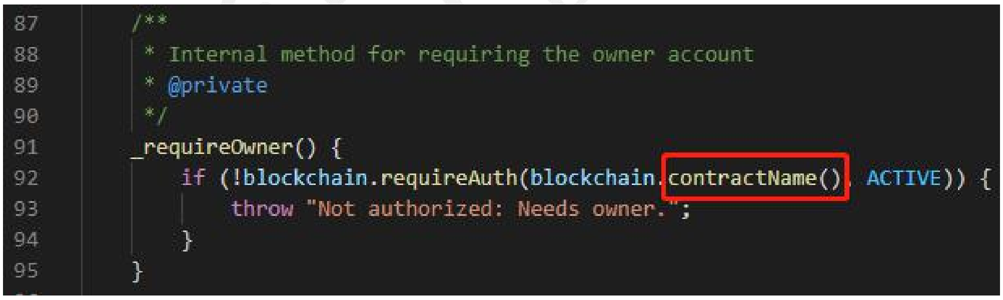
- Related Function: *setPaused, stake, unstake, transferFrozenWithdrawals, claimRewards*

- Result: Pass

4. Details of audit results

4.1 _requireOwner

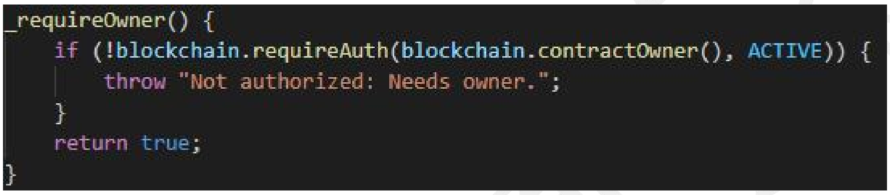
- Description: The comment indicates that this function is used for checking contract ownership. However, the parameter of *requireAuth* is *blockchain.contractName* instead of *blockchain.contractOwner*. This will cause exceptions to all other functions that call this function.



```
/**
 * Internal method for requiring the owner account
 * @private
 */
_requireOwner() {
    if (!blockchain.requireAuth(blockchain.contractName(), ACTIVE)) {
        throw "Not authorized: Needs owner.";
    }
}
```

Figure 1 Source code of *_requireOwner*

- Fix Result: Fixed. The final code is shown below.

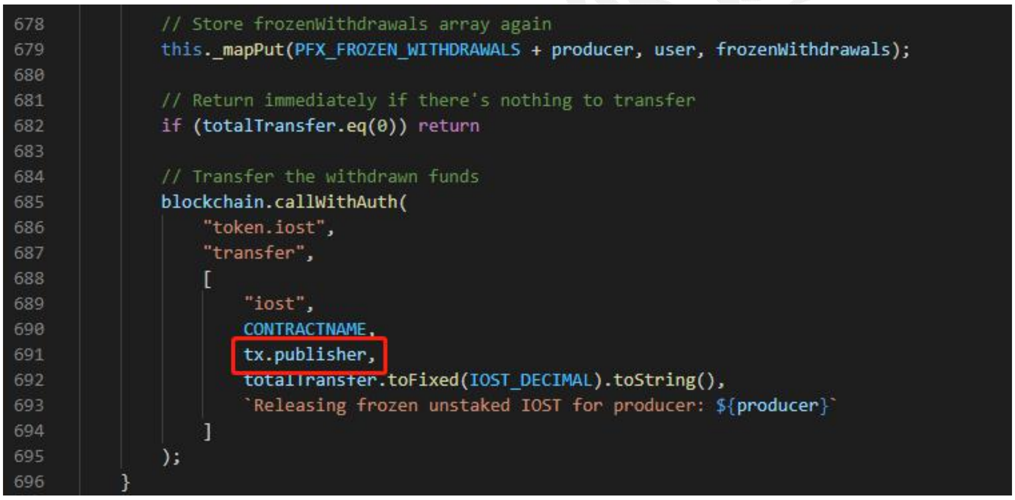


```
_requireOwner() {
    if (!blockchain.requireAuth(blockchain.contractOwner(), ACTIVE)) {
        throw "Not authorized: Needs owner.";
    }
    return true;
}
```

Figure 2 Source code of *_requireOwner*

4.2 transferFrozenWithdrawal

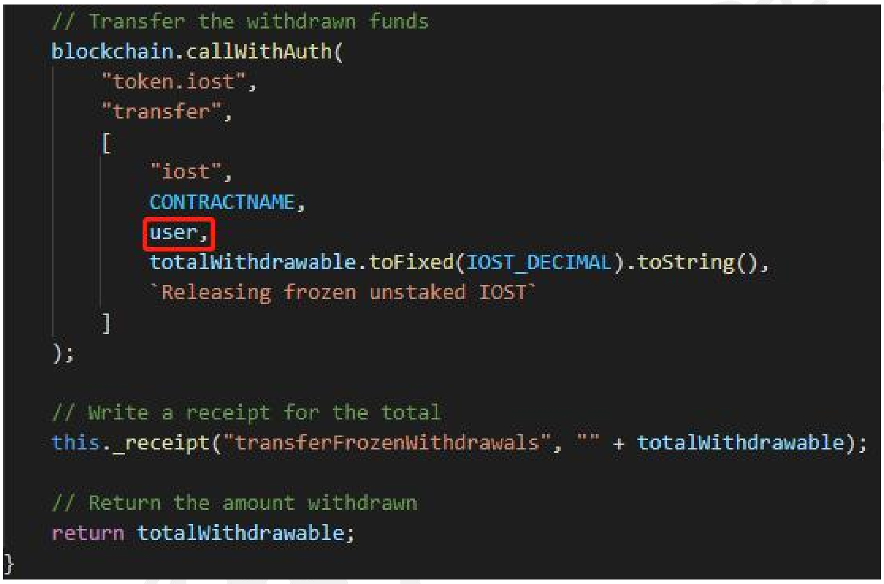
- Description: Anyone can call the *transferFrozenWithdrawal* function to receive the frozen IOST of any user. In addition, if distributeRewards calls the *transferFrozenWithdrawal* function, all rewards will be sent to *contractOwner*. It is recommended to change the *tx.publisher* of Line 691 to user, so that the unfrozen IOST can be sent to user correctly.



```
        // Store frozenWithdrawals array again
        this._mapPut(PFX_FROZEN_WITHDRAWALS + producer, user, frozenWithdrawals);

        // Return immediately if there's nothing to transfer
        if (totalTransfer.eq(0)) return

        // Transfer the withdrawn funds
        blockchain.callWithAuth(
            "token.iost",
            "transfer",
            [
                "iost",
                CONTRACTNAME,
                tx.publisher,
                totalTransfer.toFixed(IOST_DECIMAL).toString(),
                `Releasing frozen unstaked IOST for producer: ${producer}`
            ]
        );
    }
```

Figure 3 Source code of *transferFrozenWithdrawal*

- Fix Result: Fixed. The final code is shown below.

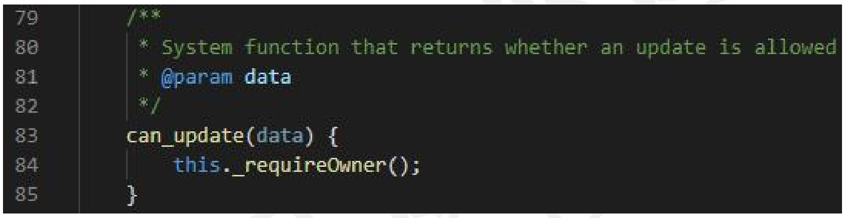


```
    // Transfer the withdrawn funds
    blockchain.callWithAuth(
        "token.iost",
        "transfer",
        [
            "iost",
            CONTRACTNAME,
            user,
            totalWithdrawable.toFixed(IOST_DECIMAL).toString(),
            `Releasing frozen unstaked IOST`
        ]
    );

    // Write a receipt for the total
    this._receipt("transferFrozenWithdrawals", "" + totalWithdrawable);

    // Return the amount withdrawn
    return totalWithdrawable;
}
```

Figure 4 Source code of *transferFrozenWithdrawals*

4.3 can_update

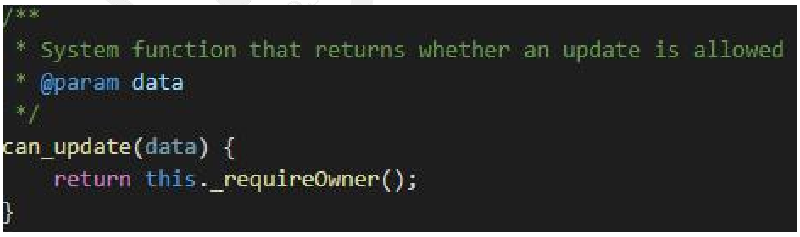
- Description: The *can_update* function is used to upgrade the contract, but after the *contractOwner* permission check is performed, the function does not return true. As a result, calling this function during the contract upgrade process will always return false and the contract cannot be upgraded. It is recommended to return true.



```
    /**
     * System function that returns whether an update is allowed
     * @param data
     */
    can_update(data) {
        this._requireOwner();
    }
```

Figure 5 Source code of *can_update*

- Fix Result: Fixed. The final code is shown below.

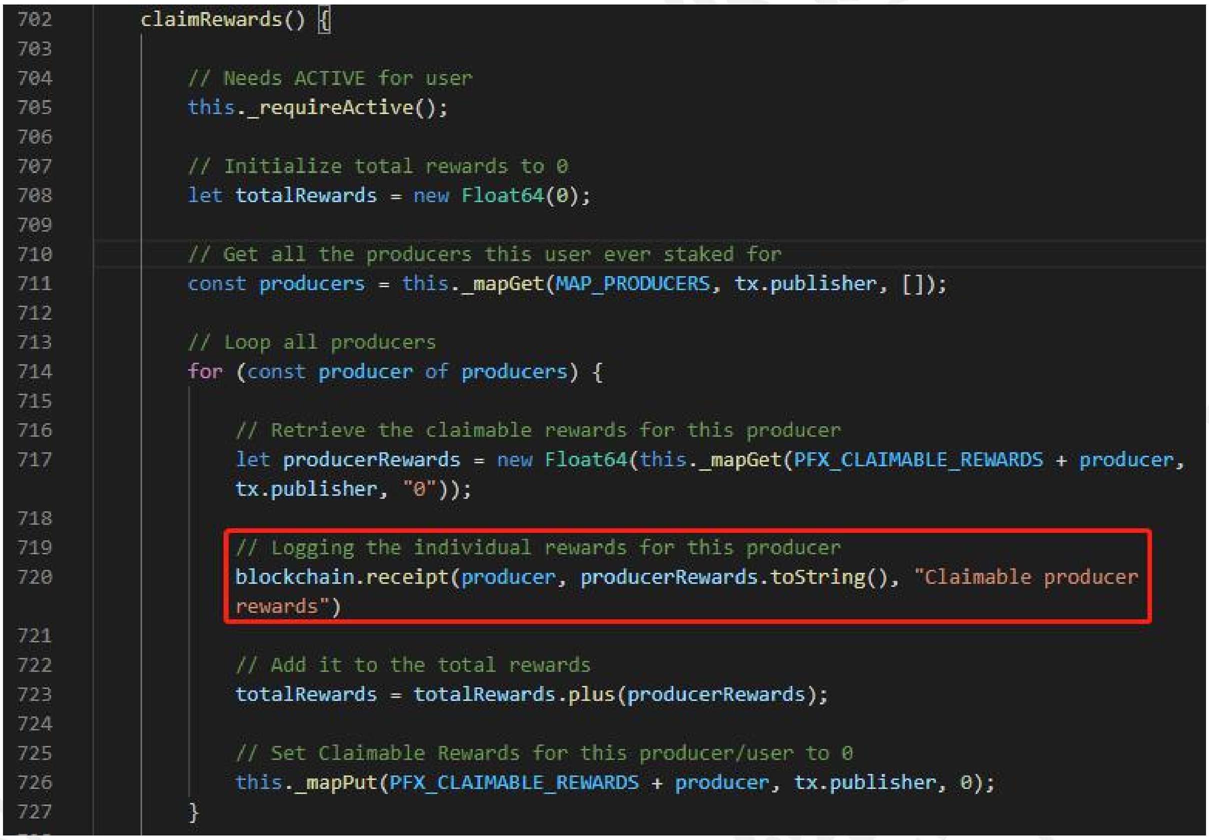


```
/**
 * System function that returns whether an update is allowed
 * @param data
 */
can_update(data) {
    return this._requireOwner();
}
```

Figure 6 Source code of *can_update*

4.4 claimRewards

- Description: *blockchain.receipt* only accepts one parameter, it is recommended to splice the parameter into json format and then pass it as a parameter to *blockchain.receipt.*



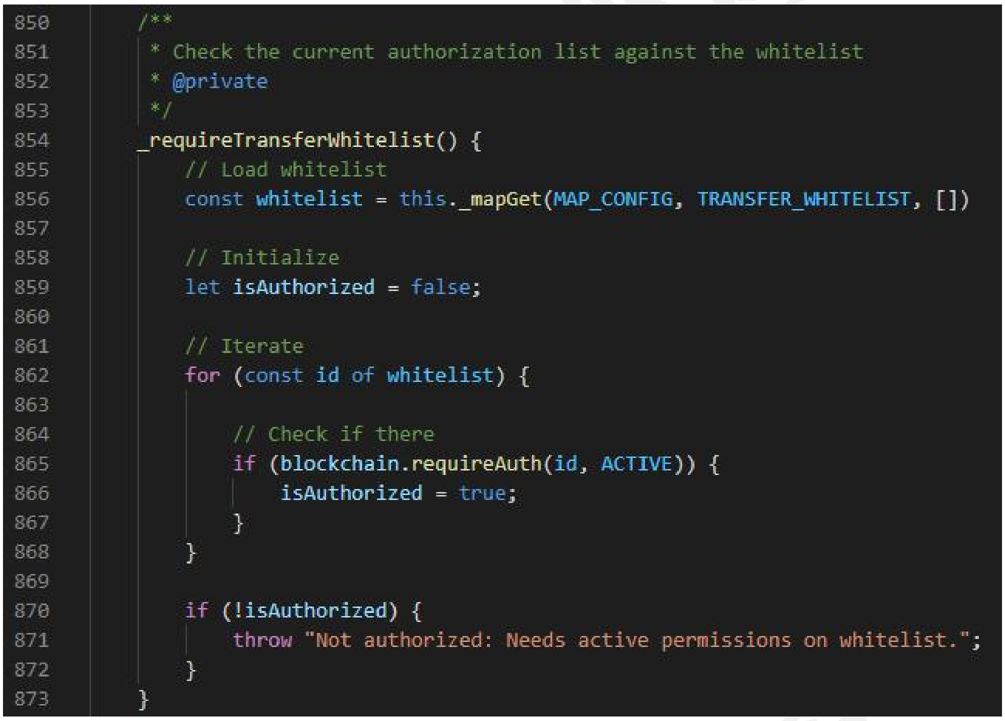
```
    claimRewards() {

        // Needs ACTIVE for user
        this._requireActive();

        // Initialize total rewards to 0
        let totalRewards = new Float64(0);

        // Get all the producers this user ever staked for
        const producers = this._mapGet(MAP_PRODUCERS, tx.publisher, []);

        // Loop all producers
        for (const producer of producers) {

            // Retrieve the claimable rewards for this producer
            let producerRewards = new Float64(this._mapGet(PFX_CLAIMABLE_REWARDS + producer,
            tx.publisher, "0"));

            // Logging the individual rewards for this producer
            blockchain.receipt(producer, producerRewards.toString(), "Claimable producer
            rewards")

            // Add it to the total rewards
            totalRewards = totalRewards.plus(producerRewards);

            // Set Claimable Rewards for this producer/user to 0
            this._mapPut(PFX_CLAIMABLE_REWARDS + producer, tx.publisher, 0);
        }
```

Figure 7 Source code of *claimRewards*

- Fix Result: Fixed.

4.5 _requireTransferWhitelist

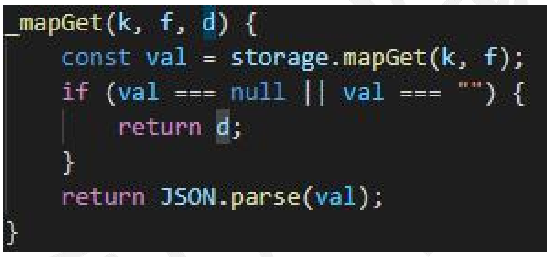
- Description: *_requireTransferWhitelist* does not convert the whitelist to an array when traversing the whitelist, so when traversing the ID, it will be treated as a string, causing the whitelist function to fail to take effect. This problem will affect the function of the *transfer/transferFreeze* function. It is recommended to use *JSON.parse* to process the whitelist before looping.



```
    /**
     * Check the current authorization list against the whitelist
     * @private
     */
    _requireTransferWhitelist() {
        // Load whitelist
        const whitelist = this._mapGet(MAP_CONFIG, TRANSFER_WHITELIST, [])

        // Initialize
        let isAuthorized = false;

        // Iterate
        for (const id of whitelist) {

            // Check if there
            if (blockchain.requireAuth(id, ACTIVE)) {
                isAuthorized = true;
            }
        }

        if (!isAuthorized) {
            throw "Not authorized: Needs active permissions on whitelist.";
        }
    }
```

Figure 8 Source code of *_requireTransferWhitelist*

- Fix Result: Fixed. The final code is shown below.

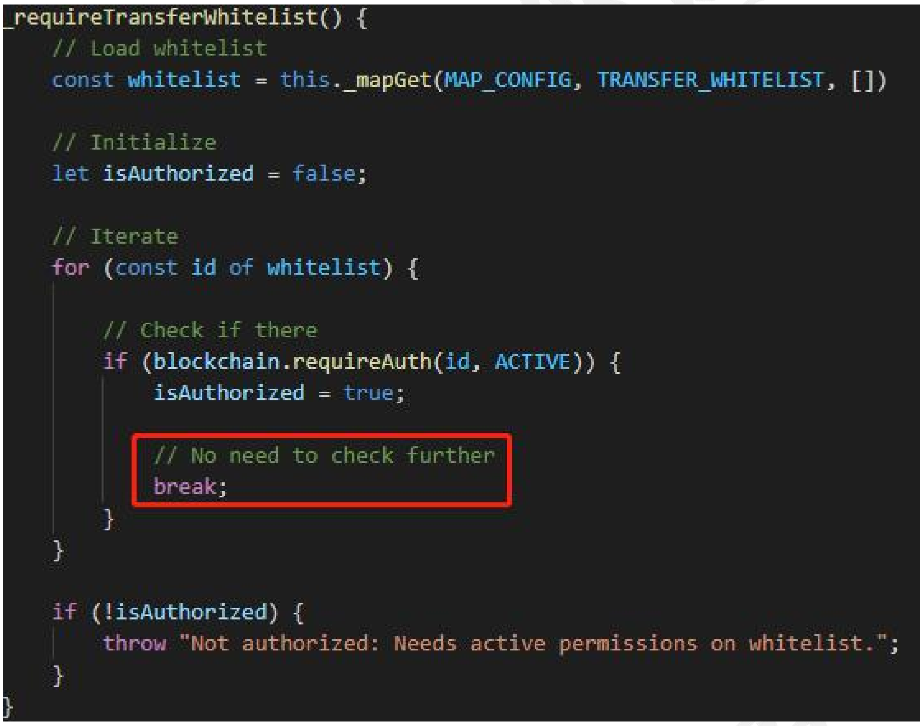


```
_mapGet(k, f, d) {
    const val = storage.mapGet(k, f);
    if (val === null || val === "") {
        return d;
    }
    return JSON.parse(val);
}
```

Figure 9 Source code of *_mapGet*

- Description: The *_requireTransferWhitelist* functioncan can add a piece of code *‘break;’* on line after *‘isAuthorized = true;’*. As long as it matches a permission in the whitelist, the transfer operation can be performed without traversing all the whitelists.

- Fix Result: Fixed. The final code is shown below.

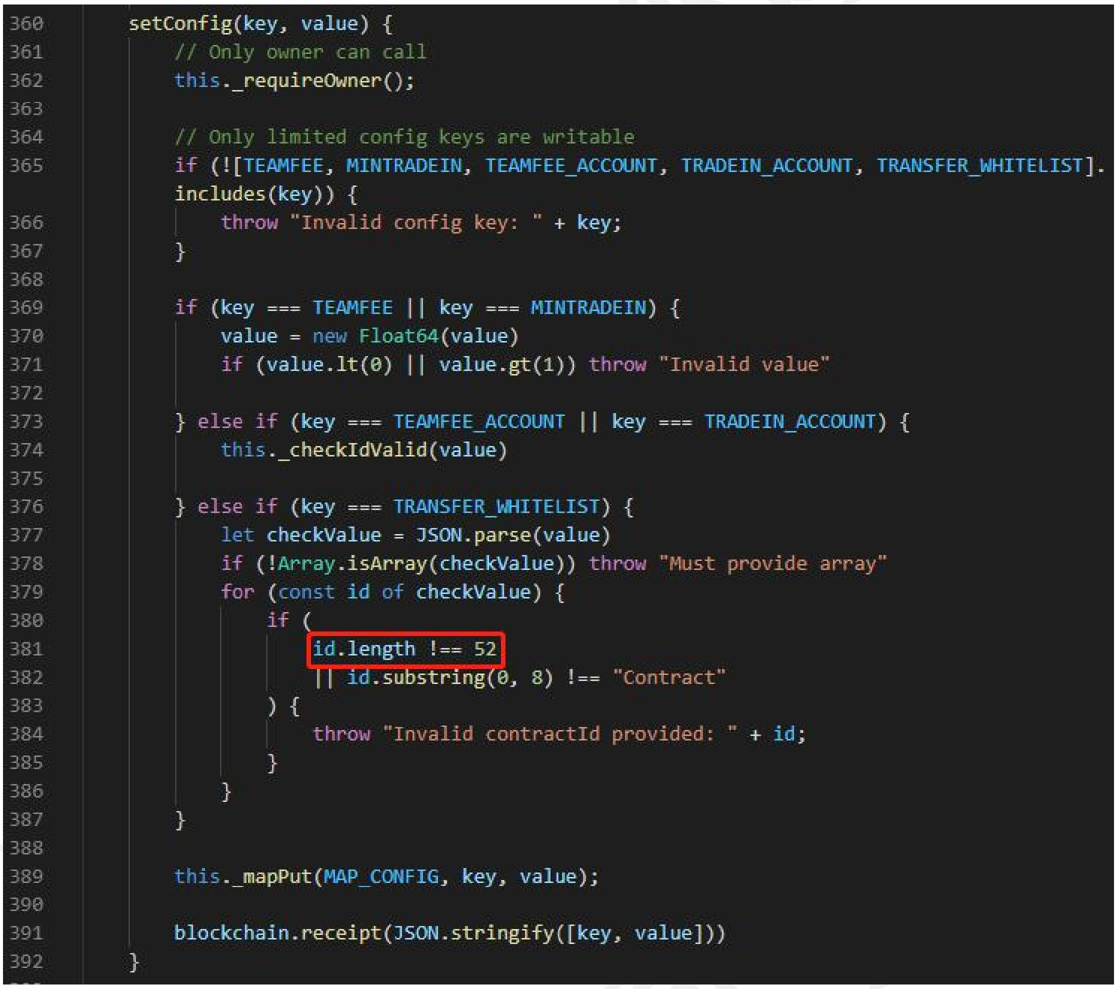


```
_requireTransferWhitelist() {
    // Load whitelist
    const whitelist = this._mapGet(MAP_CONFIG, TRANSFER_WHITELIST, [])

    // Initialize
    let isAuthorized = false;

    // Iterate
    for (const id of whitelist) {

        // Check if there
        if (blockchain.requireAuth(id, ACTIVE)) {
            isAuthorized = true;

            // No need to check further
            break;
        }
    }

    if (!isAuthorized) {
        throw "Not authorized: Needs active permissions on whitelist.";
    }
}
```

Figure 10 Source code of *_requireTransferWhitelist*

4.6 setConfig

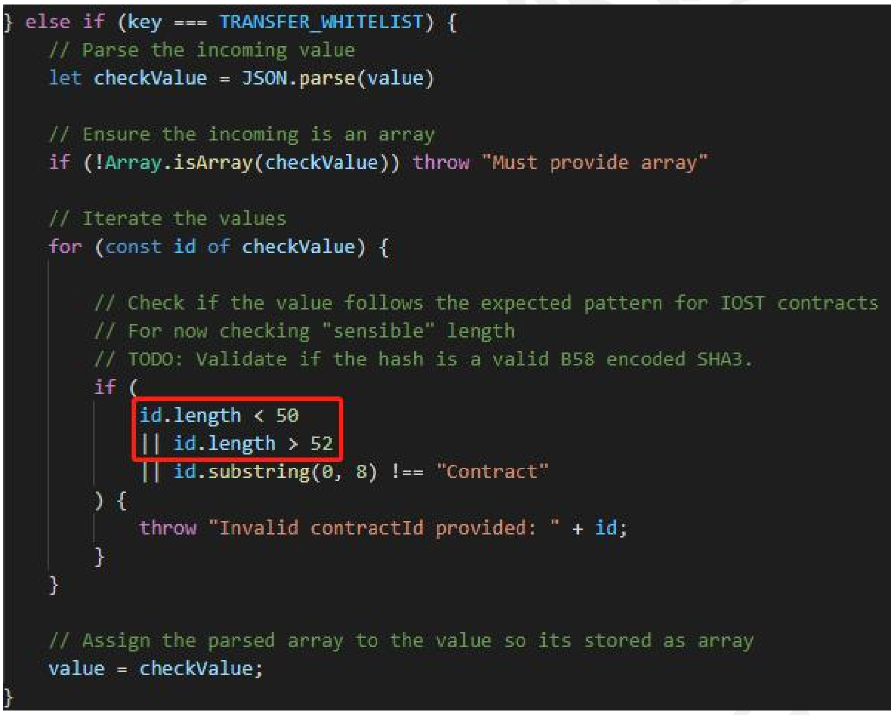
- Description: When configuring the *TRANSFER_WHITELIST* option, the length of the contract address is required to be 52, but IOST does not specify that the length of all contract addresses is 52. For example, some contract addresses have a length of 51.



```
setConfig(key, value) {
    // Only owner can call
    this._requireOwner();

    // Only limited config keys are writable
    if (![TEAMFEE, MINTRADEIN, TEAMFEE_ACCOUNT, TRADEIN_ACCOUNT, TRANSFER_WHITELIST].includes(key)) {
        throw "Invalid config key: " + key;
    }

    if (key === TEAMFEE || key === MINTRADEIN) {
        value = new Float64(value)
        if (value.lt(0) || value.gt(1)) throw "Invalid value"

    } else if (key === TEAMFEE_ACCOUNT || key === TRADEIN_ACCOUNT) {
        this._checkIdValid(value)

    } else if (key === TRANSFER_WHITELIST) {
        let checkValue = JSON.parse(value)
        if (!Array.isArray(checkValue)) throw "Must provide array"
        for (const id of checkValue) {
            if (
                id.length !== 52
                || id.substring(0, 8) !== "Contract"
            ) {
                throw "Invalid contractId provided: " + id;
            }
        }
    }

    this._mapPut(MAP_CONFIG, key, value);

    blockchain.receipt(JSON.stringify([key, value]))
}
```

Figure 11 Source code of *setConfig*

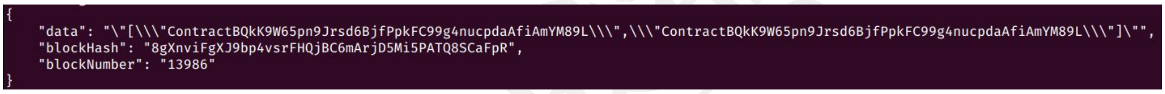
- Fix Result: Fixed. The final code is shown below.



```
} else if (key === TRANSFER_WHITELIST) {
    // Parse the incoming value
    let checkValue = JSON.parse(value)

    // Ensure the incoming is an array
    if (!Array.isArray(checkValue)) throw "Must provide array"

    // Iterate the values
    for (const id of checkValue) {

        // Check if the value follows the expected pattern for IOST contracts
        // For now checking "sensible" length
        // TODO: Validate if the hash is a valid B58 encoded SHA3.
        if (
            id.length < 50
            || id.length > 52
            || id.substring(0, 8) !== "Contract"
        ) {
            throw "Invalid contractId provided: " + id;
        }
    }

    // Assign the parsed array to the value so its stored as array
    value = checkValue;
}
```

Figure 12 Part of the source code of function *setConfig*

- Description: *TRANSFER_WHITELIST* is not deduplicated, and the same contract address can be added multiple times. However, this function is called by *contractOwner* and can be checked manually before entering the parameters.



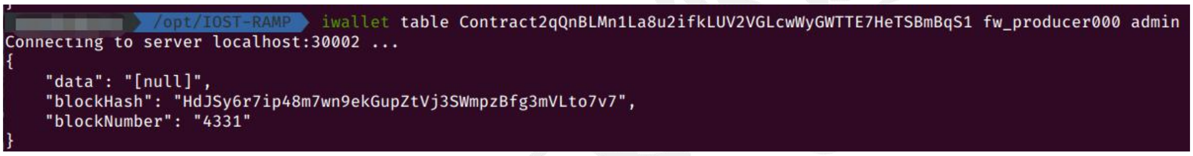
```
{
    "data": "\"[\\\"ContractBQkK9W65pn9Jrsd6BjfPpkFC99g4nucpdaAfiAmYM89L\\\",\\\"ContractBQkK9W65pn9Jrsd6BjfPpkFC99g4nucpdaAfiAmYM89L\\\"]\"",
    "blockHash": "8gXnviFgXJ9bp4vsrFHQjBC6mArjD5Mi5PATQ8SCaFpR",
    "blockNumber": "13986"
}
```

Figure 13 Duplicate whitelist

- Fix Result: Ignore.

4.7 transferFrozenWithdrawals

- Description: In the map of *transferFrozenWithdrawal*, it doesn’t check whether *frozenWithdrawal* is [null]. In the following case, *transferFrozenWithdrawal* will throw an exception: If the user unstakes first, after the freezing time has passed, the user claims the frozen IOST. The corresponding table value [fw_producer, user] will become [null], because null is also an element and will enter the code block from line 663 to line 677. But at line 666, reading the data of the [0] index of the null element will throw an exception. At the same time, because *distributeRewards* will also call *transferFrozenWithdrawal*, this problem will cause *contractOwner* to fail to distribute voting rewards. It is recommended to add null element detection.



```
[redacted] /opt/IOST-RAMP  iwallet table Contract2qQnBLMn1La8u2ifkLUV2VGLcwWyGWTTE7HeTSBmBqS1 fw_producer000 admin
Connecting to server localhost:30002 ...
{
    "data": "[null]",
    "blockHash": "HdJSy6r7ip48m7wn9ekGupZtVj3SWmpzBfg3mVLto7v7",
    "blockNumber": "4331"
}
```

Figure 14 Screenshot of null array

- Fix Result: Fixed.

4.8 _checkToken

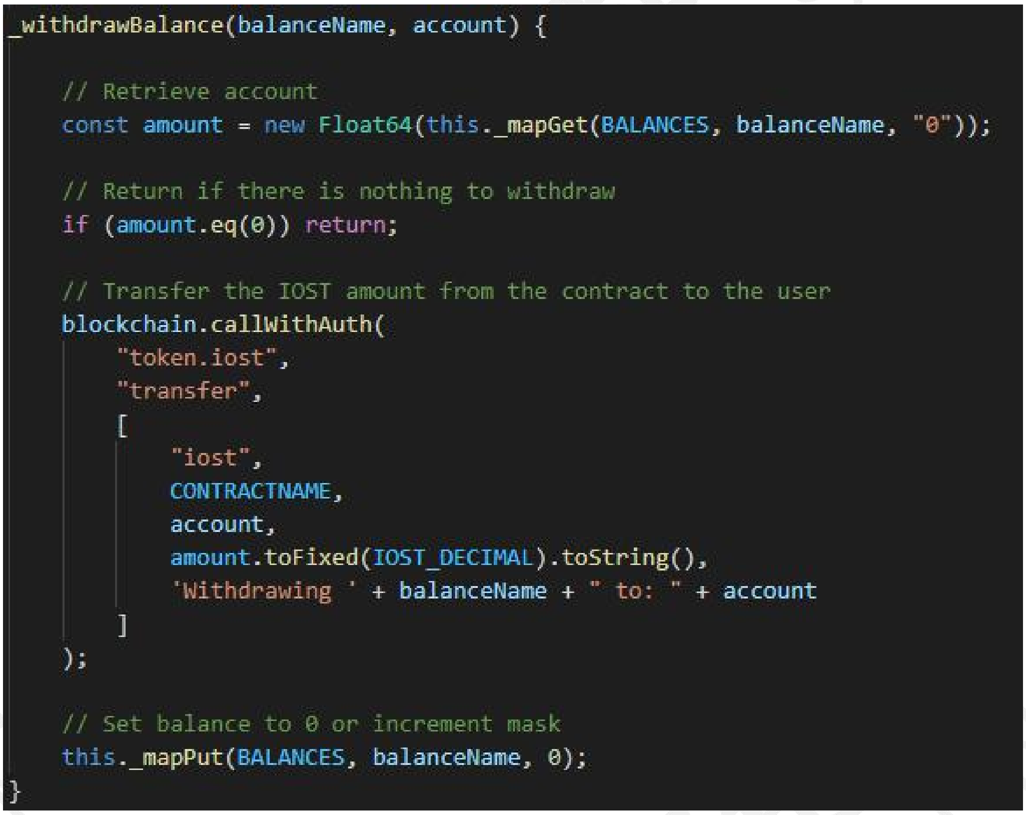
- Description: All functions that use _checkToken can directly specify tokenSymbol as *TOKEN_SYMBOL* without user input.

- Fix Result: Ignore.

4.9 withdrawTeamFees & withdrawTradeIn

- Description: After *withdrawTeamFees* & *withdrawTradeIn* were called to withdraw the corresponding IOST, it did not reset the data in the corresponding table to 0. As a result, *contractOwner* can always use these two functions to send IOST to the team Account and trade-in Account addresses.

- Fix Result: Fixed. The final code is shown below.



```
_withdrawBalance(balanceName, account) {

    // Retrieve account
    const amount = new Float64(this._mapGet(BALANCES, balanceName, "0"));

    // Return if there is nothing to withdraw
    if (amount.eq(0)) return;

    // Transfer the IOST amount from the contract to the user
    blockchain.callWithAuth(
        "token.iost",
        "transfer",
        [
            "iost",
            CONTRACTNAME,
            account,
            amount.toFixed(IOST_DECIMAL).toString(),
            'Withdrawing ' + balanceName + " to: " + account
        ]
    );

    // Set balance to 0 or increment mask
    this._mapPut(BALANCES, balanceName, 0);
}
```

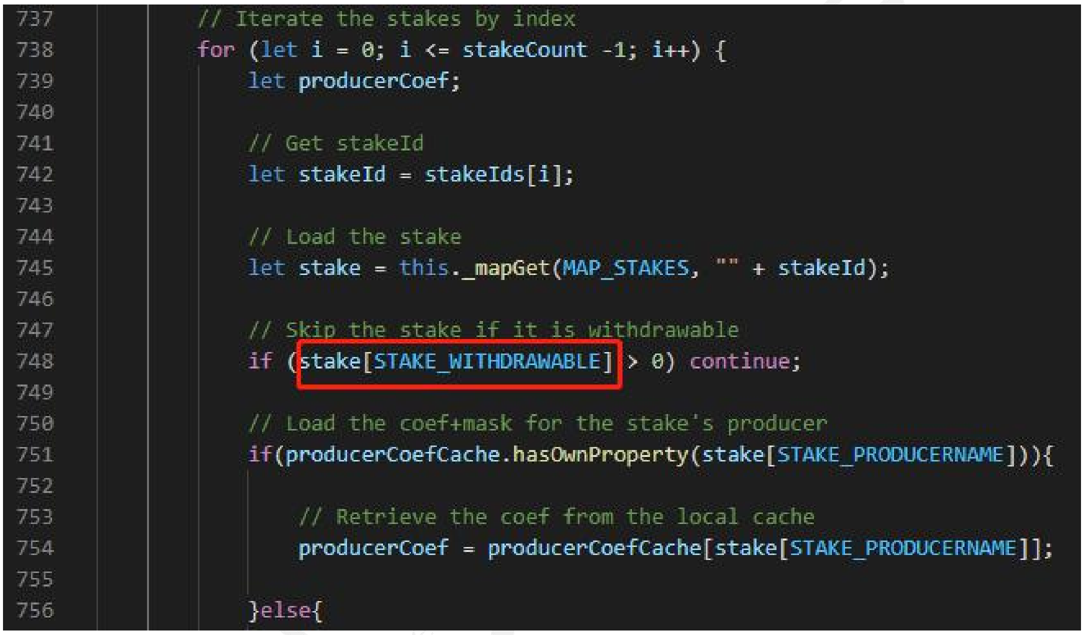
Figure 15 Part of the source code of function *_withdrawBalance*

4.10 processStakeRewards

- Description: *processStakeRewards* has problems in the following situation: Stake generates 3 id => processProducerBonus(contractOwner call) => processStakeRewards (anyone can call) => [0].staker.claimRewards => processProducerBonus => processStakeRewards => [0].staker.claimRewards. In the case of no new bonus, stake user can still receive the stake bonus repeatedly with the updated data.

- Fix Result: Fixed.

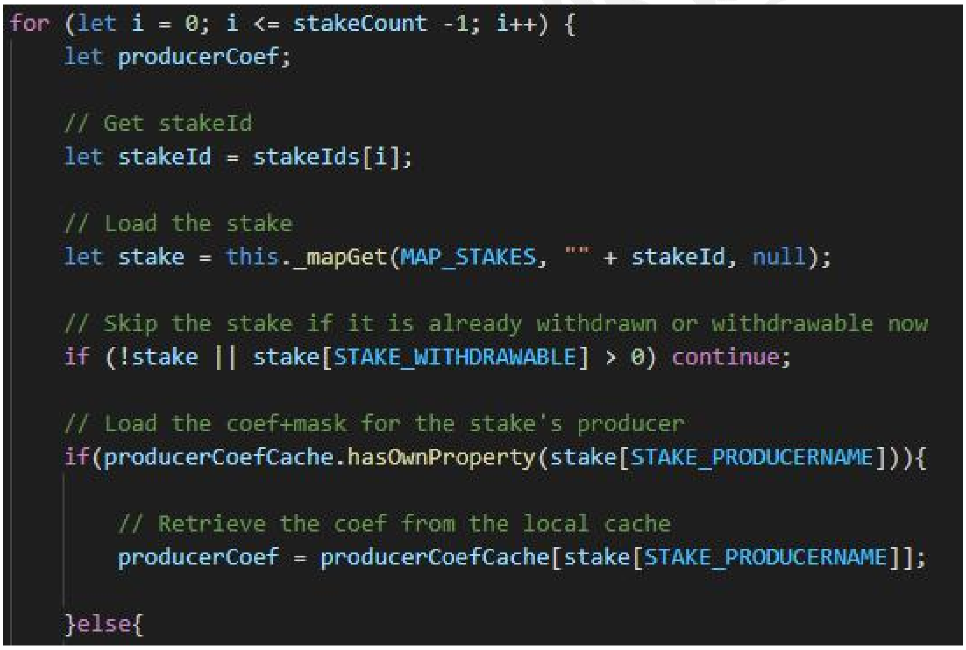
- Description: All *STAKE_IDS* will be put into *PROCESSING_STAKE_IDS* in *processProducerBonus* function. However, if a user calls *transferFrozenWithdrawals* after this, the stake information corresponding to *unstake* will be deleted. This will cause when the *processStakeRewards* function is called, on line 748 of the code, *stake[STAKE_WITHDRAWABLE]* will be error and throw an exception (because stake=null), which will cause *processStakeRewards* to fail to be called. It is recommended to modify the processing logic of *processStakeRewards*, such as adding a check for whether stake is null.



```
737         // Iterate the stakes by index
738         for (let i = 0; i <= stakeCount -1; i++) {
739             let producerCoef;
740 
741             // Get stakeId
742             let stakeId = stakeIds[i];
743 
744             // Load the stake
745             let stake = this._mapGet(MAP_STAKES, "" + stakeId);
746 
747             // Skip the stake if it is withdrawable
748             if (stake[STAKE_WITHDRAWABLE] > 0) continue;
749 
750             // Load the coef+mask for the stake's producer
751             if(producerCoefCache.hasOwnProperty(stake[STAKE_PRODUCERNAME])){
752 
753                 // Retrieve the coef from the local cache
754                 producerCoef = producerCoefCache[stake[STAKE_PRODUCERNAME]];
755 
756             }else{
```

Figure 16 Part of the source code of function *processStakeRewards*

- Fix Result: Fixed. The final code is shown below.

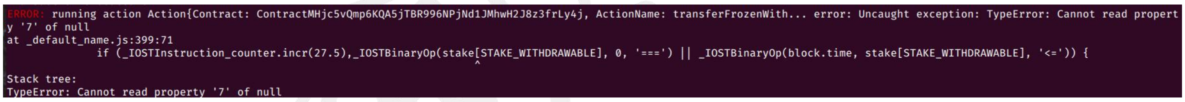


```
for (let i = 0; i <= stakeCount -1; i++) {
    let producerCoef;

    // Get stakeId
    let stakeId = stakeIds[i];

    // Load the stake
    let stake = this._mapGet(MAP_STAKES, "" + stakeId, null);

    // Skip the stake if it is already withdrawn or withdrawable now
    if (!stake || stake[STAKE_WITHDRAWABLE] > 0) continue;

    // Load the coef+mask for the stake's producer
    if(producerCoefCache.hasOwnProperty(stake[STAKE_PRODUCERNAME])){

        // Retrieve the coef from the local cache
        producerCoef = producerCoefCache[stake[STAKE_PRODUCERNAME]];

    }else{
```

Figure 17 Part of the source code of function *processStakeRewards*

4.11 transferFrozenWithdrawals

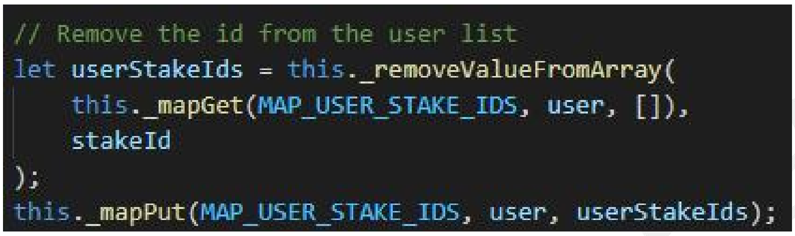
- Description: On line 909 of *transferFrozenWithdrawals* function, when the user transfers all frozen withdrawals, the corresponding id should be deleted from the user's list, but because it uses map storage, the *this._get* used by *_removeValueFromFieldArray* cannot find the corresponding array. And, after the corresponding user calls *transferFrozenWithdrawals* again, the element with the null value will be read, and therefore an exception will be thrown. It is recommended to use *_mapGet/_mapPut* to read and write the remaining id.



```
ERROR: running action Action{Contract: ContractMHjc5vQmp6KQA5jTBR996NPjNd1JMhwH2J8z3frLy4j, ActionName: transferFrozenWith... error: Uncaught exception: TypeError: Cannot read propert
y '7' of null
at _default_name.js:399:71
            if (_IOSTInstruction_counter.incr(27.5),_IOSTBinaryOp(stake[STAKE_WITHDRAWABLE], 0, '===') || _IOSTBinaryOp(block.time, stake[STAKE_WITHDRAWABLE], '<=')) {
                                                                ^

Stack tree:
TypeError: Cannot read property '7' of null
```

Figure 18 Screenshot of error message

- Fix Result: Fixed. The final code is shown below.



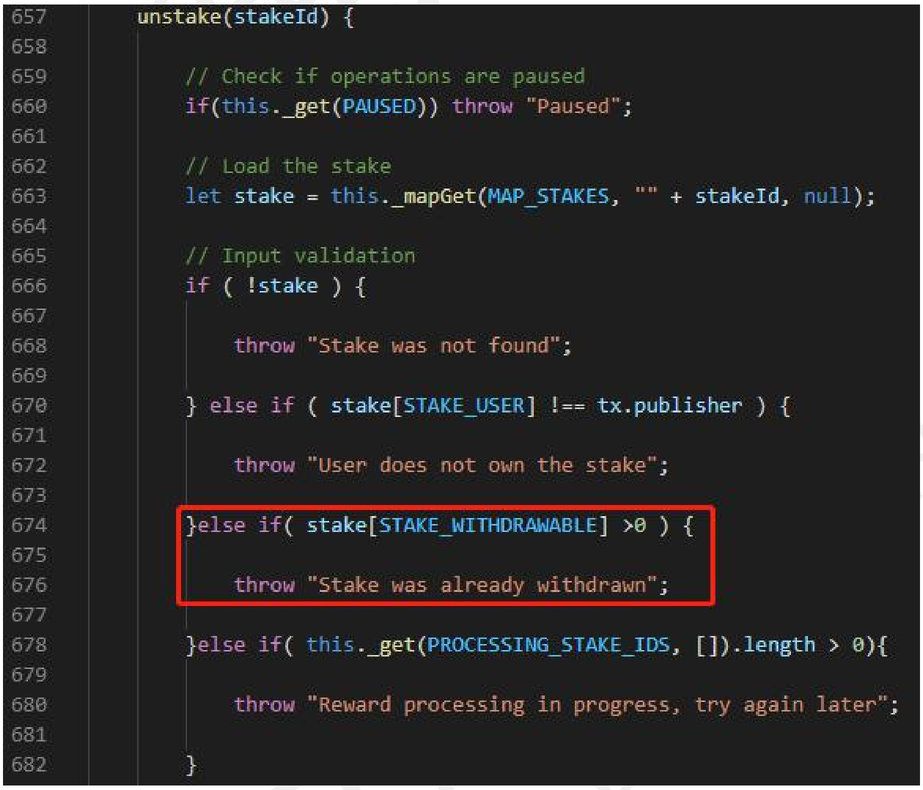
```
// Remove the id from the user list
let userStakeIds = this._removeValueFromArray(
    this._mapGet(MAP_USER_STAKE_IDS, user, []),
    stakeId
);
this._mapPut(MAP_USER_STAKE_IDS, user, userStakeIds);
```

Figure 19 Part of the source code of function *transferFrozenWithdrawals*

4.12 unstake

- Description: When *unstake* is called again, the *stake[STAKE_WITHDRAWABLE]* is not checked as 0, which results in the user being able to unstake the stake with the same id repeatedly, destroy the corresponding wIOST multiple times and cancel the corresponding vote. These wIOSTs and votes are the same user’s stake acquisitions of other ID’s wIOST and voting. As a result, users cannot unstake all because the corresponding wIOST and IOST votes are insufficient.

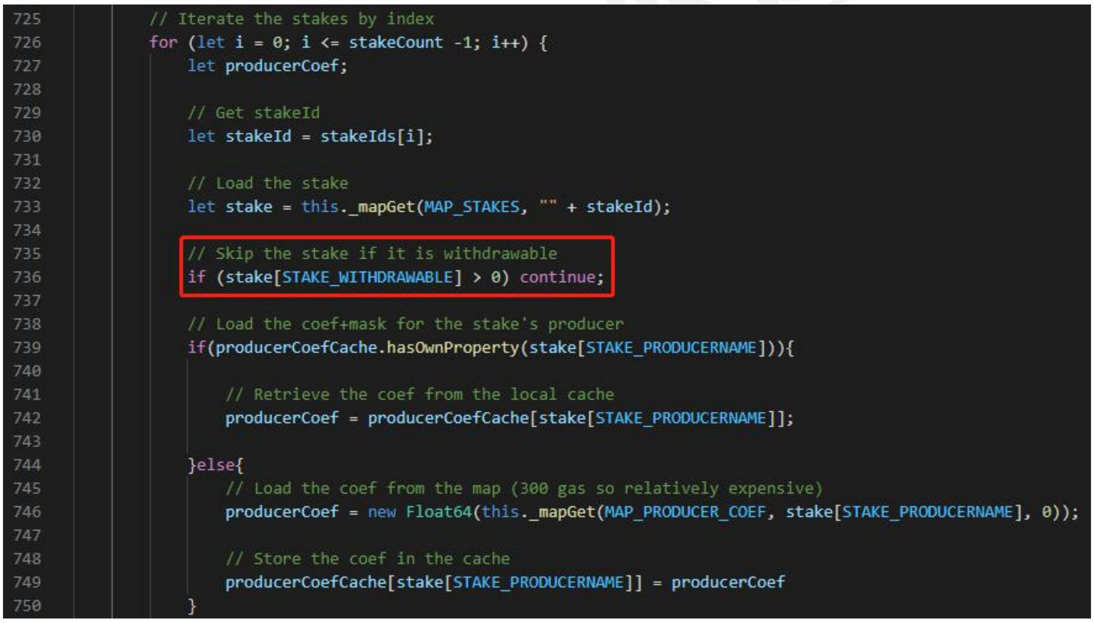
- Fix Result: Fixed. The final code is shown below.



```
    unstake(stakeId) {

        // Check if operations are paused
        if(this._get(PAUSED)) throw "Paused";

        // Load the stake
        let stake = this._mapGet(MAP_STAKES, "" + stakeId, null);

        // Input validation
        if ( !stake ) {

            throw "Stake was not found";

        } else if ( stake[STAKE_USER] !== tx.publisher ) {

            throw "User does not own the stake";

        }else if( stake[STAKE_WITHDRAWABLE] >0 ) {

            throw "Stake was already withdrawn";

        }else if( this._get(PROCESSING_STAKE_IDS, []).length > 0){

            throw "Reward processing in progress, try again later";

        }
```

Figure 20 Source code of *unstake*

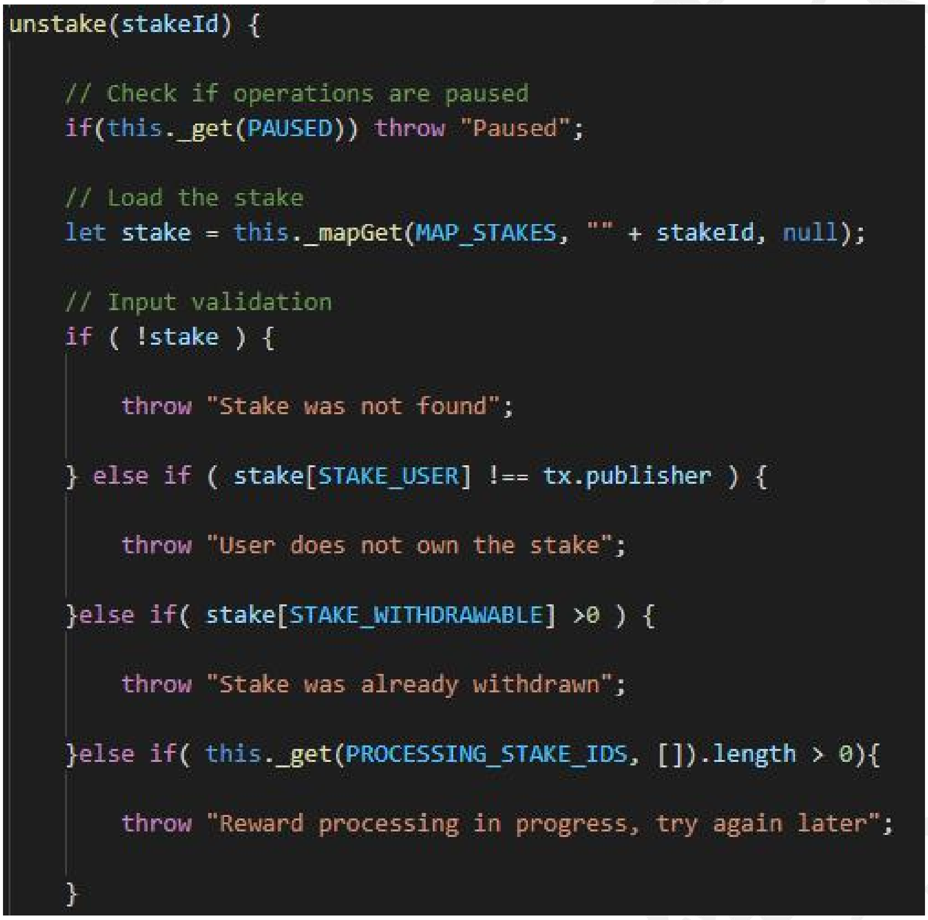
- Description: If *unstake* is called immediately after *processProducerBonus*, and then *processStakeRewards* is called, the reward corresponding to the stake will not be allocated to the corresponding stake user, team, trade-in because of line 736 of the code *‘if(stake[STAKE_WITHDRAWABLE]>0) continue;’*, But stay in the contract account and cannot be withdrawn.



```
        // Iterate the stakes by index
        for (let i = 0; i <= stakeCount -1; i++) {
            let producerCoef;

            // Get stakeId
            let stakeId = stakeIds[i];

            // Load the stake
            let stake = this._mapGet(MAP_STAKES, "" + stakeId);

            // Skip the stake if it is withdrawable
            if (stake[STAKE_WITHDRAWABLE] > 0) continue;

            // Load the coef+mask for the stake's producer
            if(producerCoefCache.hasOwnProperty(stake[STAKE_PRODUCERNAME])){

                // Retrieve the coef from the local cache
                producerCoef = producerCoefCache[stake[STAKE_PRODUCERNAME]];

            }else{
                // Load the coef from the map (300 gas so relatively expensive)
                producerCoef = new Float64(this._mapGet(MAP_PRODUCER_COEF, stake[STAKE_PRODUCERNAME], 0));

                // Store the coef in the cache
                producerCoefCache[stake[STAKE_PRODUCERNAME]] = producerCoef
            }
```

Figure 21 Part of the source code of function *processStakeRewards*

- Fix Result: Fixed. The final code is shown below.



```
unstake(stakeId) {

    // Check if operations are paused
    if(this._get(PAUSED)) throw "Paused";

    // Load the stake
    let stake = this._mapGet(MAP_STAKES, "" + stakeId, null);

    // Input validation
    if ( !stake ) {

        throw "Stake was not found";

    } else if ( stake[STAKE_USER] !== tx.publisher ) {

        throw "User does not own the stake";

    }else if( stake[STAKE_WITHDRAWABLE] >0 ) {

        throw "Stake was already withdrawn";

    }else if( this._get(PROCESSING_STAKE_IDS, []).length > 0){

        throw "Reward processing in progress, try again later";

    }
```

Figure 22 Part of the source code of function *unstake*

4.13 Matters needing attention in canceling voting freeze

- Description: IOST's default freezing time for canceling voting is 7 days, and the freezing time set by this contract is 3 days. The user can apply for withdrawing the frozen IOST from the contract 4 days in advance, and the contract provides the user with a 4-day IOST freezing time. The source of IOST withdrawn in advance is the IOST stored in the contract address, and its possible source is voting rewards or unstakes that have been unfrozen. If there is no IOST in the contract, the withdrawal will fail and the *contractOwner* will not be able to distribute rewards. In addition, if there are multiple such operations, most of the IOST of the contract will be locked, and the corresponding amount of voting rewards will not be able to be received.

- Fix Result: Ignore.

5 Conclusion

Beosin (Chengdu LianAn) conducted a detailed audit on the design and code implementation of the StakingManager project. All the problems found in the audit process were notified to the project party, and got quick feedback and repair from the project party. Beosin (Chengdu LianAn) confirms that all the problems found have been properly fixed or have reached an agreement with the project party has on how to deal with it. The overall result of this StakingManager audit is **Pass.**

Beosin

成都链安
BEOSIN

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com