成都链安
BEOSIN

Smart contract security audit report

Audit Number：202010201514

Report Query Name: OIN

Smart Contract Info：

Smart Contract Name	**Smart Contract Address**	**Smart Contract Address Link**
Esm	0xED39B9a5773a5370471cb89525d914BE9EaFdA2e	https://etherscan.io/address/0xed39b9a5773a5370471cb89525d914be9eafda2e#code
Dparam	0xc9FE746507cE0C0408878f12a955b43760b41216	https://etherscan.io/address/0xc9fe746507ce0c0408878f12a955b43760b41216#code
Oracle	0xDc7d33398342D58b935446F9F6E052b48793a4Ac	https://etherscan.io/address/0xdc7d33398342d58b935446f9f6e052b48793a4ac#code
OinToken	0x9aeB50f542050172359A0e1a25a9933Bc8c01259	https://etherscan.io/address/0x9aeB50f542050172359A0e1a25a9933Bc8c01259#code
USDOToken	0x88EEC3873ac22DA27a836F7f63883DA4771C96F1	https://etherscan.io/address/0x88eec3873ac22da27a836f7f63883da4771c96f1#code
OinStake	0xD539cb51D7662F93b2B2a2D1631b9C9e989b90ec	https://etherscan.io/address/0xD539cb51D7662F93b2B2a2D1631b9C9e989b90ec#code

Start Date：2020.09.21

Completion Date：2020.09.25

Overall Result：Pass

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

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	Compiler Version Security	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		SafeMath Features	Pass
		require/assert Usage	Pass
		Gas Consumption	Pass
		Visibility Specifiers	Pass
		Fallback 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
		Access Control of Owner	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		tx.origin Usage	Pass
		Replay Attack	Pass
		Overriding Variables	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 project OIN, including Coding Standards, Security, and Business Logic. **The OIN project passed all audit items. The overall result is Pass. The smart contract is able to function properly.**

1. Coding Conventions

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

1.1 Compiler Version Security

- Description: Check whether the code implementation of current contract contains the exposed solidity compiler bug.
- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.
- Result: Pass

1.3 Redundant Code

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

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.
- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.
- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- 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 your contract and change state, such as withdrawing ETH.
- 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 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract. In this project, the contract
- Result: Pass

2.10 Replay Attack

- Description: Check the weather the implement possibility of Replay Attack exists in the contract.

- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

3. Token Audit

3.1 Basic token information of OIN

Token name	oinfinance
Token symbol	OIN
decimals	8
totalSupply	100 million (burnable)
Token type	ERC20

Table 1 – Basic Token Information

3.2 Basic Token Information of USDO1

Token name	USD Oin one
Token symbol	USDO1
decimals	8
totalSupply	89,881.59810299 (Mintable without cap; burnable)
Token type	ERC20

Table 2 – Basic Token Information

3.3 Token Vesting Information

Missing

4. Business Security

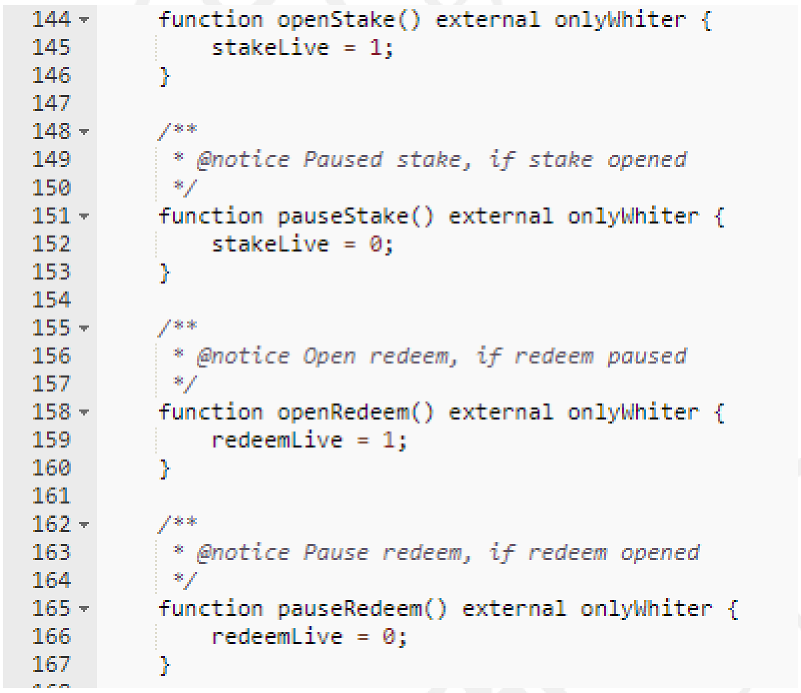
Check whether the business is secure.

In this project, total six smart contracts are implemented or interacted, including *Esm*, *Dparam*, *Oracle*, *OinToken*, *USDOToken* and the main business contract *OinStake*, the corresponding functions and businesses are descripted as following:

4.1 Esm contract

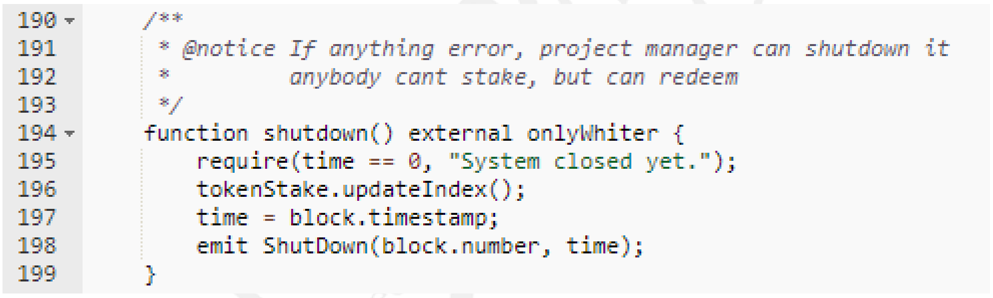
- Description:

  The Esm contract implements the *Ownd* module and *WhiteList* module to resolve the manager permission. The ownership can be transferred through calling functions *nominateNewOwner* and *acceptOwnership*, the current owner nominate a new owner address, then the nominated owner can accept ownership via calling function *acceptOwnership*. In addition the contract owner can call the functions *appendWhiter* and *removeWhiter* to append or remove a specified whitelist address, the whitelist address can control the stake & redeem & close operation available status in OinStake contract.



```
function openStake() external onlyWhiter {
    stakeLive = 1;
}

/**
 * @notice Paused stake, if stake opened
 */
function pauseStake() external onlyWhiter {
    stakeLive = 0;
}

/**
 * @notice Open redeem, if redeem paused
 */
function openRedeem() external onlyWhiter {
    redeemLive = 1;
}

/**
 * @notice Pause redeem, if redeem opened
 */
function pauseRedeem() external onlyWhiter {
    redeemLive = 0;
}
```

Figure 1 source code of functions which only whitelist address can call (1/2 in Esm contract)



```
/**
 * @notice If anything error, project manager can shutdown it
 *         anybody cant stake, but can redeem
 */
function shutdown() external onlyWhiter {
    require(time == 0, "System closed yet.");
    tokenStake.updateIndex();
    time = block.timestamp;
    emit ShutDown(block.number, time);
}
```

Figure 2 source code of functions which only whitelist address can call (2/2 in Esm contract)

- Related functions: *nominateNewOwner*, *acceptOwnership*, *appendWhiter*, *removeWhiter*, *setupTokenStake*, *openStake*, *pauseStake*, *openRedeem*, *pauseRedeem*, *shutdown*

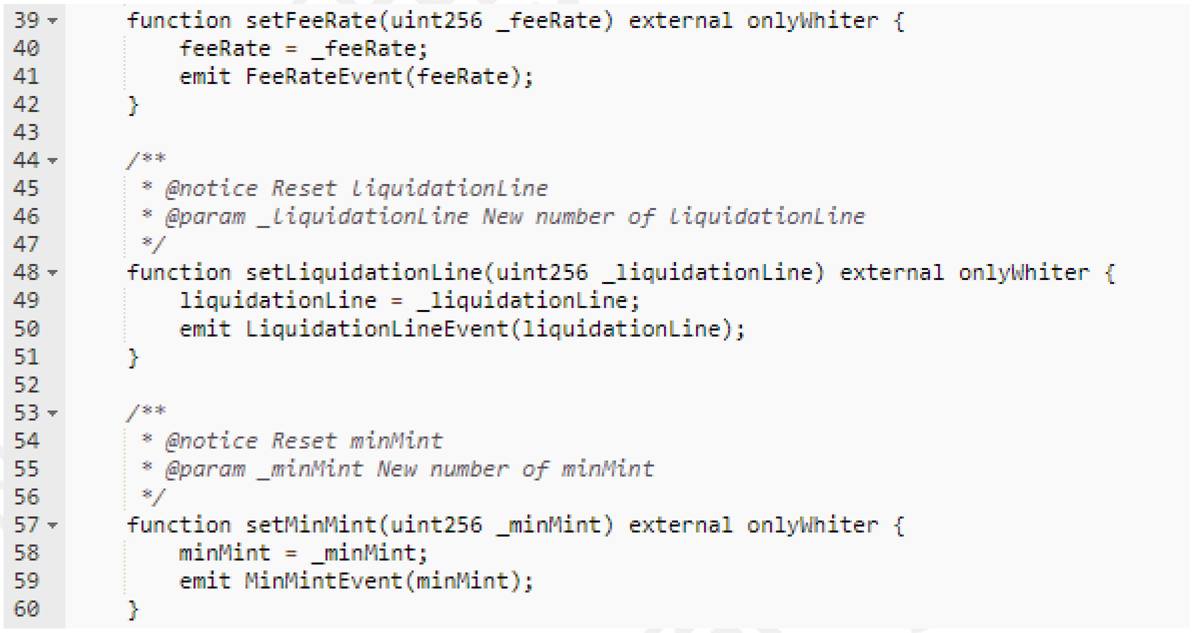
- Result: Pass

4.2 Dparam contract

- Description:

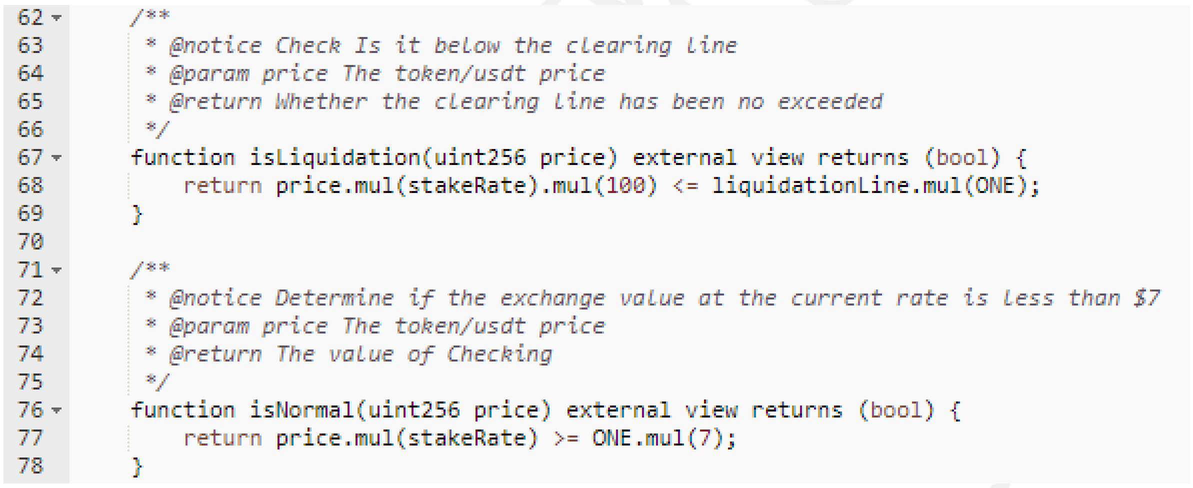
The Dparam contract implements the *Ownd* module and *WhiteList* module to resolve the manager permission. The ownership can be transferred through calling functions *nominateNewOwner* and *acceptOwnership*, the current owner nominate a new owner address, then the nominated owner can accept ownership via calling function *acceptOwnership*. In addition the contract owner can call the functions *appendWhiter* and *removeWhiter* to append or remove a specified whitelist address, the whitelist address can set the other contracts check/use parameters including *feeRate* & *liquidationLine* & *minMint*.

And the functions as shown in Figure 4, which are used to query/check the validity of price, they are called in contracts Oracle & OinStake.



```
    function setFeeRate(uint256 _feeRate) external onlyWhiter {
        feeRate = _feeRate;
        emit FeeRateEvent(feeRate);
    }

    /**
     * @notice Reset liquidationLine
     * @param _liquidationLine New number of liquidationLine
     */
    function setLiquidationLine(uint256 _liquidationLine) external onlyWhiter {
        liquidationLine = _liquidationLine;
        emit LiquidationLineEvent(liquidationLine);
    }

    /**
     * @notice Reset minMint
     * @param _minMint New number of minMint
     */
    function setMinMint(uint256 _minMint) external onlyWhiter {
        minMint = _minMint;
        emit MinMintEvent(minMint);
    }
```

Figure 3 source code of functions which only whitelist address can call (in Dparam contract)



```
    /**
     * @notice Check Is it below the clearing line
     * @param price The token/usdt price
     * @return Whether the clearing line has been no exceeded
     */
    function isLiquidation(uint256 price) external view returns (bool) {
        return price.mul(stakeRate).mul(100) <= liquidationLine.mul(ONE);
    }

    /**
     * @notice Determine if the exchange value at the current rate is less than $7
     * @param price The token/usdt price
     * @return The value of Checking
     */
    function isNormal(uint256 price) external view returns (bool) {
        return price.mul(stakeRate) >= ONE.mul(7);
    }
```

Figure 4 source code of functions which are used to query/check validity of price

- Related functions: *nominateNewOwner*, *acceptOwnership*, *appendWhiter*, *removeWhiter*, *setFeeRate*, *setLiquidationLine*, *setMinMint*, *isLiquidation*, *isNormal*

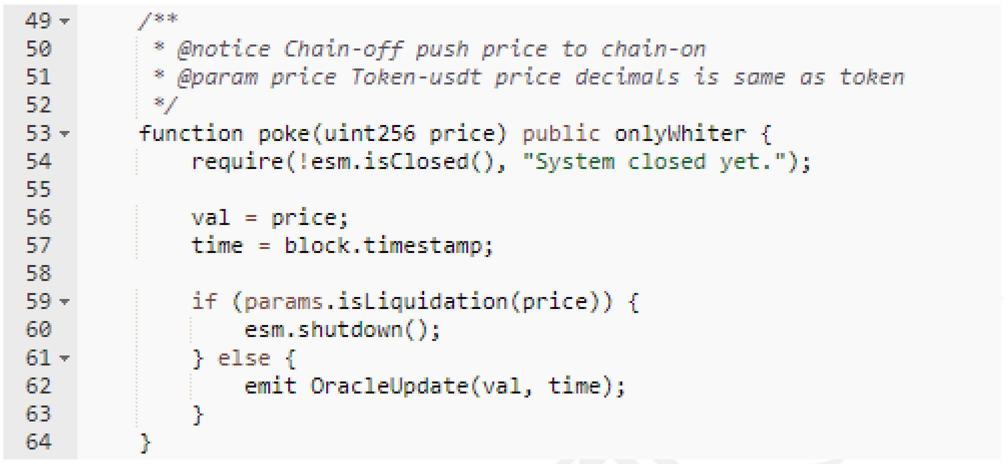
- Result: Pass

4.3 Oracle contract

- Description:

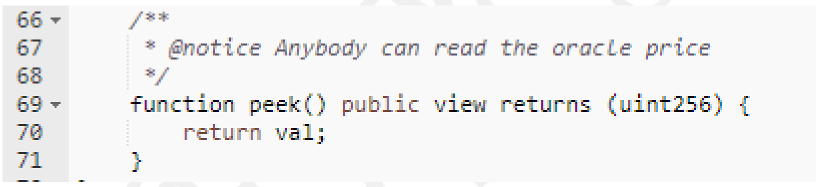
The Oracle contract implements the *Ownd* module and *WhiteList* module to resolve the manager permission. The ownership can be transferred through calling functions *nominateNewOwner* and *acceptOwnership*, the current owner nominate a new owner address, then the nominated owner can accept ownership via calling function *acceptOwnership*. In addition the contract owner can call the functions *appendWhiter* and *removeWhiter* to append or remove a specified whitelist address, the whitelist address can call the function *poke* to push a token-usdt price from chain-off to chain-on, and the validity of price is checked here, according to the check result to execute the different operations.

And correspondingly, the function *peek* is implemented to query the current price.



```
    /**
     * @notice Chain-off push price to chain-on
     * @param price Token-usdt price decimals is same as token
     */
    function poke(uint256 price) public onlyWhiter {
        require(!esm.isClosed(), "System closed yet.");

        val = price;
        time = block.timestamp;

        if (params.isLiquidation(price)) {
            esm.shutdown();
        } else {
            emit OracleUpdate(val, time);
        }
    }
```

Figure 5 source code of function poke in Oracle contract



```
    /**
     * @notice Anybody can read the oracle price
     */
    function peek() public view returns (uint256) {
        return val;
    }
```

Figure 6 source code of function peek in Oracle contract

- Related functions: *nominateNewOwner, acceptOwnership, appendWhiter, removeWhiter, poke, peek*
- Result: Pass

4.4 OinToken contract

- Description:

The OinToken contract implements a standard ERC20 token which has *burn* & *burnFrom* functions. The total token supply is not constant. Beware that changing an allowance with function *approve* brings the risk that someone may use both the old and the new allowance by unfortunate transaction ordering. It is recommended that users reset the allowance to zero, and then set a new allowance.

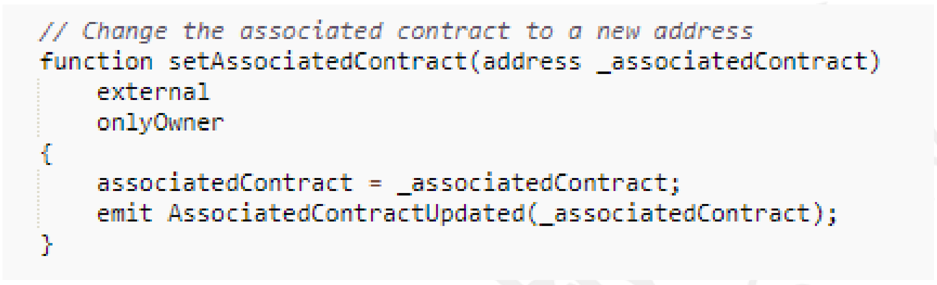
- Related functions: *totalSupply, balanceOf, allowance, transfer, transferFrom, approve, approveAndCall, burn, burnFrom*

- Result: Pass

4.5 USDOToken contract

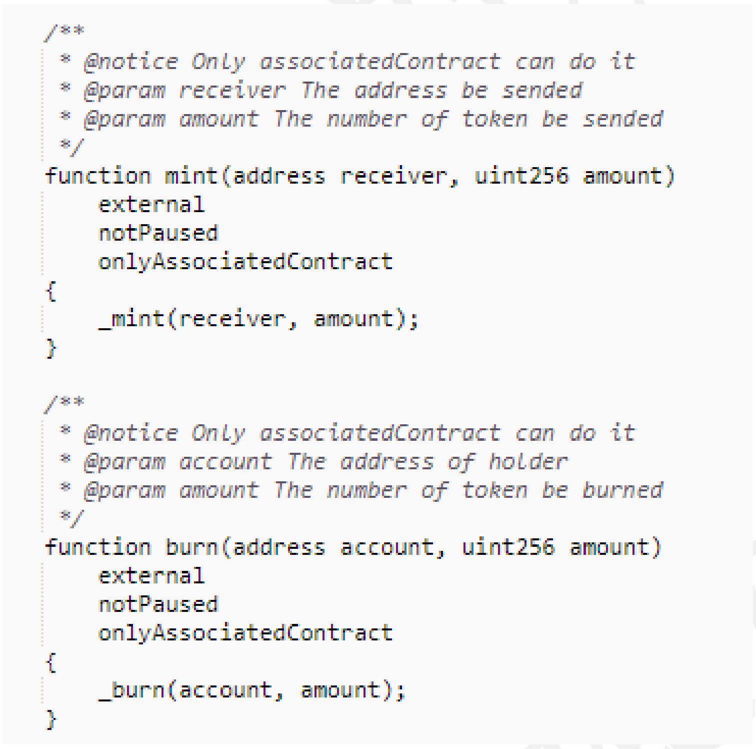
- Description:

The USDOToken contract implements a standard ERC20 token which has *Pausable* module, *mint*, and *burn* functions. The total token supply is not constant. In addition, the USDOToken contract implements the *Ownd* module to resolve the manager permission. The ownership can be transferred through calling functions *nominateNewOwner* and *acceptOwnership*, the current owner nominate a new owner address, then the nominated owner can accept ownership via calling function *acceptOwnership*, and the owner can set the specified *associatedContract* address. The mint & burn functions can only be called by the specified *associatedContract* address. Beware that changing an allowance with function *approve* brings the risk that someone may use both the old and the new allowance by unfortunate transaction ordering. Using function *increaseAllowance* and *decreaseAllowance* to alter allowance is recommended.



```
// Change the associated contract to a new address
function setAssociatedContract(address _associatedContract)
    external
    onlyOwner
{
    associatedContract = _associatedContract;
    emit AssociatedContractUpdated(_associatedContract);
}
```

Figure 7 source code of function setAssociatedContract



```
/**
 * @notice Only associatedContract can do it
 * @param receiver The address be sended
 * @param amount The number of token be sended
 */
function mint(address receiver, uint256 amount)
    external
    notPaused
    onlyAssociatedContract
{
    _mint(receiver, amount);
}

/**
 * @notice Only associatedContract can do it
 * @param account The address of holder
 * @param amount The number of token be burned
 */
function burn(address account, uint256 amount)
    external
    notPaused
    onlyAssociatedContract
{
    _burn(account, amount);
}
```

Figure 8 source code of functions mint & burn (in USDOToken contract)

- Related functions: *totalSupply*, *balanceOf*, *allowance*, *transfer*, *transferFrom*, *approve*, *increaseAllowance*, *decreaseAllowance*, *mint*, *burn*, *nominateNewOwner*, *acceptOwnership*, *setAssociatedContract*

- Result: Pass

4.6 OinStake contract

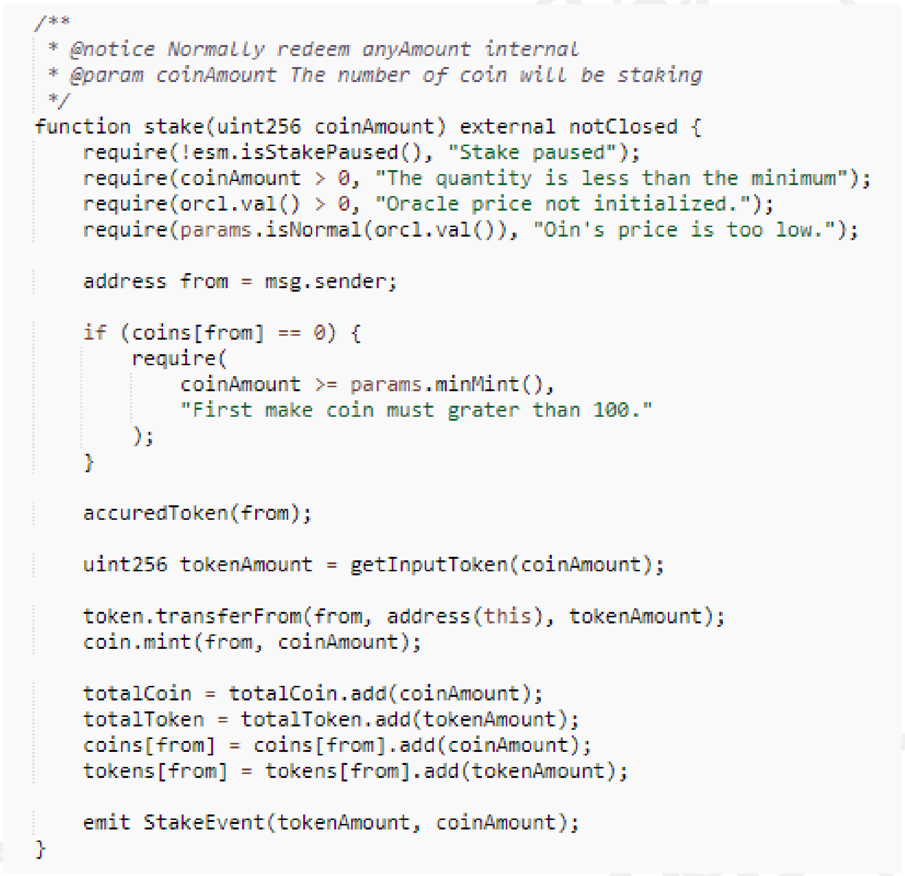
- Description:

The OinStake implements the main "stake-reward" business logic. The detailed description of each function is divided into the following part:

4.6.1 Stake tokens

- Description:

The OinStake implements the *stake* function to stake OIN tokens. This function requires that the stake should not be paused and the corresponding environment variable like the OIN price in Oracle contract should be valid. The caller pre-approve the contract address. By calling the *transferFrom* function in the OIN token contract, the contract address transfers the specified amount of OIN tokens to the contract address on behalf of the caller. And the corresponding amount of USDO1 tokens will be minted to the caller address, especially, if the USDO1 token balance of caller is 0, the minimum minting amount is required.



```
/**
 * @notice Normally redeem anyAmount internal
 * @param coinAmount The number of coin will be staking
 */
function stake(uint256 coinAmount) external notClosed {
    require(!esm.isStakePaused(), "Stake paused");
    require(coinAmount > 0, "The quantity is less than the minimum");
    require(orcl.val() > 0, "Oracle price not initialized.");
    require(params.isNormal(orcl.val()), "Oin's price is too low.");

    address from = msg.sender;

    if (coins[from] == 0) {
        require(
            coinAmount >= params.minMint(),
            "First make coin must grater than 100."
        );
    }

    accuredToken(from);

    uint256 tokenAmount = getInputToken(coinAmount);

    token.transferFrom(from, address(this), tokenAmount);
    coin.mint(from, coinAmount);

    totalCoin = totalCoin.add(coinAmount);
    totalToken = totalToken.add(tokenAmount);
    coins[from] = coins[from].add(coinAmount);
    tokens[from] = tokens[from].add(tokenAmount);

    emit StakeEvent(tokenAmount, coinAmount);
}
```

Figure 9 source code of function stake

- Related functions: *stake, isStakePaused, isNormal, val, minMint, getInputToken, transferFrom, mint*
- Result: Pass

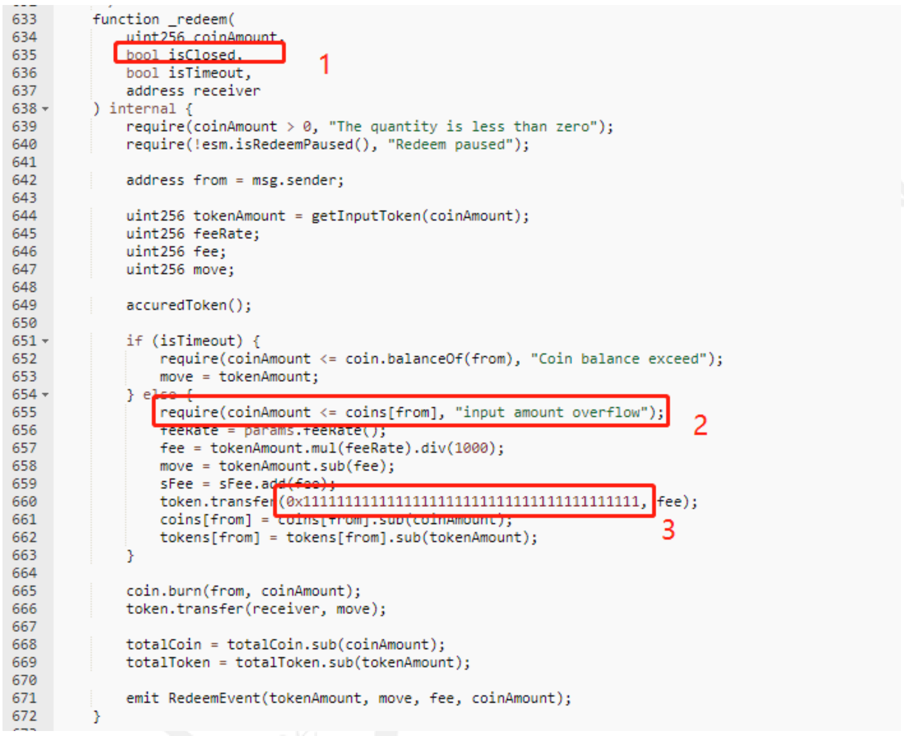
4.6.2 Redeem tokens

- Description:

The internal function *_redeem* is used to redeem staked OIN tokens. As shown in the red mark 1, the parameter is unused;

As shown in the red mark 2, when *isTimeout* is false, the number of USDO1 tokens to be destroyed in the else branch is restricted to be no greater than the number of recorded minted USDO1s;

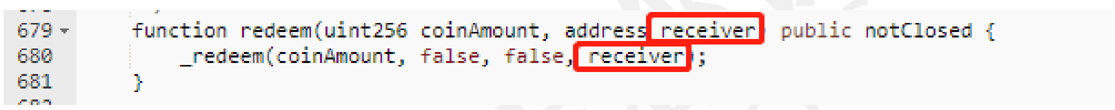
As shown by the red mark 3, after the actual deployment, the address is hard-coded to be the designated fee acceptance address and cannot be changed.



```
function _redeem(
    uint256 coinAmount,
    bool isClosed,
    bool isTimeout,
    address receiver
) internal {
    require(coinAmount > 0, "The quantity is less than zero");
    require(!esm.isRedeemPaused(), "Redeem paused");

    address from = msg.sender;

    uint256 tokenAmount = getInputToken(coinAmount);
    uint256 feeRate;
    uint256 fee;
    uint256 move;

    accuredToken();

    if (isTimeout) {
        require(coinAmount <= coin.balanceOf(from), "Coin balance exceed");
        move = tokenAmount;
    } else {
        require(coinAmount <= coins[from], "input amount overflow");
        feeRate = params.feeRate();
        fee = tokenAmount.mul(feeRate).div(1000);
        move = tokenAmount.sub(fee);
        sFee = sFee.add(fee);
        token.transfer(0x1111111111111111111111111111111111111111, fee);
        coins[from] = coins[from].sub(coinAmount);
        tokens[from] = tokens[from].sub(tokenAmount);
    }

    coin.burn(from, coinAmount);
    token.transfer(receiver, move);

    totalCoin = totalCoin.sub(coinAmount);
    totalToken = totalToken.sub(tokenAmount);

    emit RedeemEvent(tokenAmount, move, fee, coinAmount);
}
```

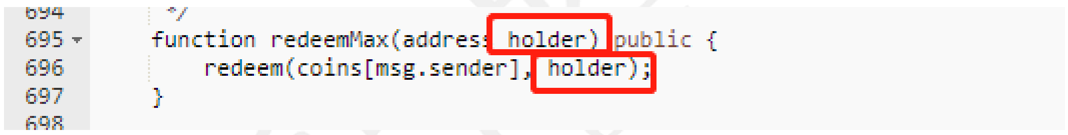
Figure 10 source code of function _redeem

As shown in Figure 11-13 below, when redeeming staked OIN tokens through the following functions, user can specify the receiving address and send the corresponding amount of OIN tokens to this address. The OIN tokens in the OinStake contract address are composed of two parts, one is the part that the owner injects into the contract address to issue rewards, and the other part is staked into the contract by the user; if the amount is withdrawn by mistake, this part of the tokens will be sent to the contract address, and OIN token do not have the function of minting, which will make this part of the tokens locked in the contract (this contract do not have the function of extracting tokens).



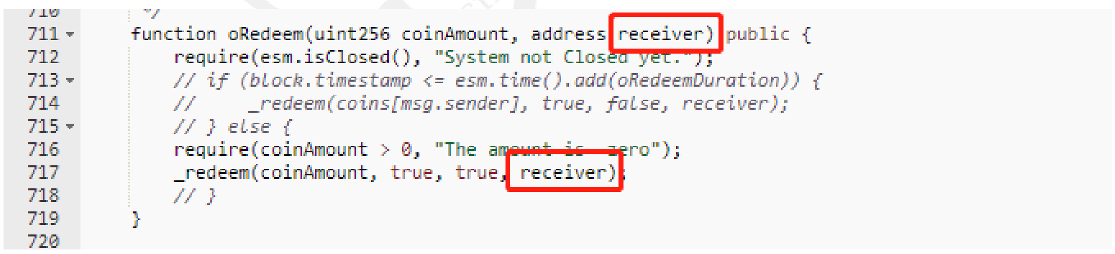
```
function redeem(uint256 coinAmount, address receiver) public notClosed {
    _redeem(coinAmount, false, false, receiver);
}
```

Figure 11 source code of function redeem (with 2 parameters)



```
 */
function redeemMax(address holder) public {
    redeem(coins[msg.sender], holder);
}
```

Figure 12 source code of function redeemMax



```
 */
function oRedeem(uint256 coinAmount, address receiver) public {
    require(esm.isClosed(), "System not Closed yet.");
    // if (block.timestamp <= esm.time().add(oRedeemDuration)) {
    //     _redeem(coins[msg.sender], true, false, receiver);
    // } else {
    require(coinAmount > 0, "The amount is zero");
    _redeem(coinAmount, true, true, receiver);
    // }
}
```

Figure 13 source code of function oRedeem

- Modify Recommendation:

1) To confirm whether it is necessary. If there is no actual use requirement, it is recommended to delete the unused parameter and correspondingly modify the call parameter input of the internal function in other functions.

2) To confirm whether the USDO1 tokens are in circulation, and if so, when the from (msg.sender) obtained USDO1 tokens from other ways, they cannot be redeemed according to the actual amount (all balance of from);

3) It is recommended to change the address to owner or other addresses that can be changed to prevent the inconvenience caused by the loss of the private key.

4) It is recommended to restrict this address to prevent misoperation.

- Fix Result:

1) Fixed. The redeem operation is changed to new functions as shown in the fixed code.

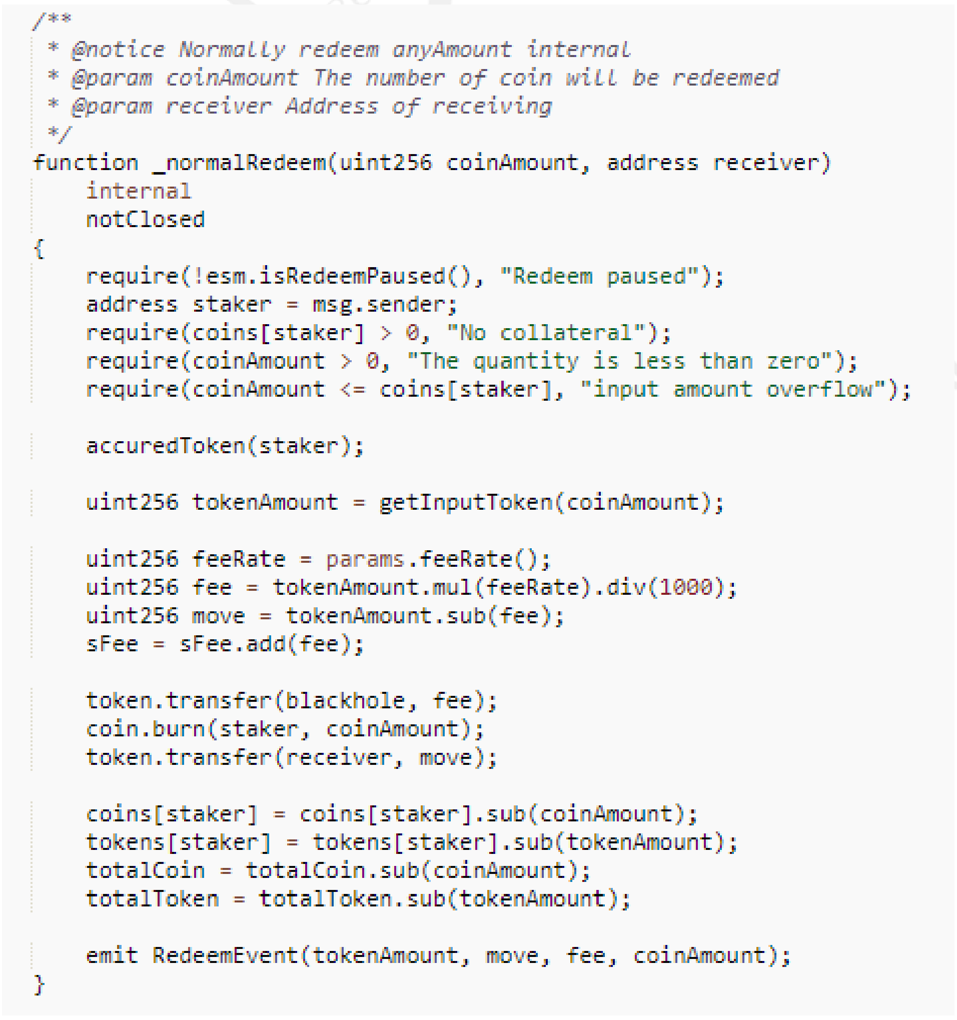
2) Ignore. The function restricts that only the staked OIN corresponding to the recorded USDO1 can be redeemed. And the code is changed as shown in the fixed code below.

3) Ignore.

4) Ignore.

- Description of fixed code:

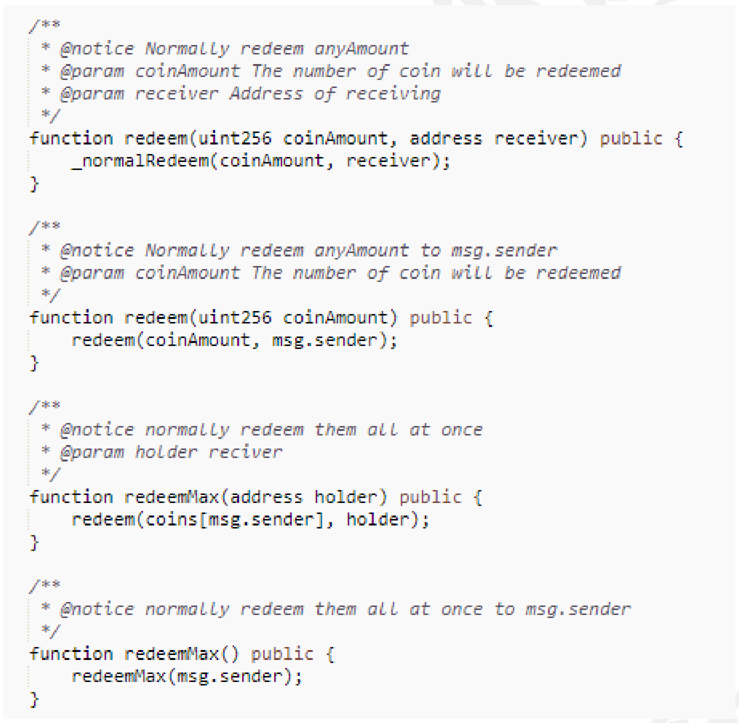
The OinStake implements the *_normalRedeem* and *_abnormalRedeem* functions to do the token redeem related operations. The *_normalRedeem* function requires that the redeem operation can only be executed when the “stake-reward” mode is not closed and the token redeem is not paused. After checking the validity of redeem amount, the *accuredToken* function is called to update the reward storage data, then the corresponding redeem fee deduction and OIN token redeem and USDO1 token destruction are performed. This internal function is called in different redeem functions (Figure 15), the main difference of these callings is that the redeem receiver address is passed differently or whether the all staked OIN tokens need to be redeemed or both of two cases.



```
/**
 * @notice Normally redeem anyAmount internal
 * @param coinAmount The number of coin will be redeemed
 * @param receiver Address of receiving
 */
function _normalRedeem(uint256 coinAmount, address receiver)
    internal
    notClosed
{
    require(!esm.isRedeemPaused(), "Redeem paused");
    address staker = msg.sender;
    require(coins[staker] > 0, "No collateral");
    require(coinAmount > 0, "The quantity is less than zero");
    require(coinAmount <= coins[staker], "input amount overflow");

    accuredToken(staker);

    uint256 tokenAmount = getInputToken(coinAmount);

    uint256 feeRate = params.feeRate();
    uint256 fee = tokenAmount.mul(feeRate).div(1000);
    uint256 move = tokenAmount.sub(fee);
    sFee = sFee.add(fee);

    token.transfer(blackhole, fee);
    coin.burn(staker, coinAmount);
    token.transfer(receiver, move);

    coins[staker] = coins[staker].sub(coinAmount);
    tokens[staker] = tokens[staker].sub(tokenAmount);
    totalCoin = totalCoin.sub(coinAmount);
    totalToken = totalToken.sub(tokenAmount);

    emit RedeemEvent(tokenAmount, move, fee, coinAmount);
}
```

Figure 14 source code of function _normalRedeem

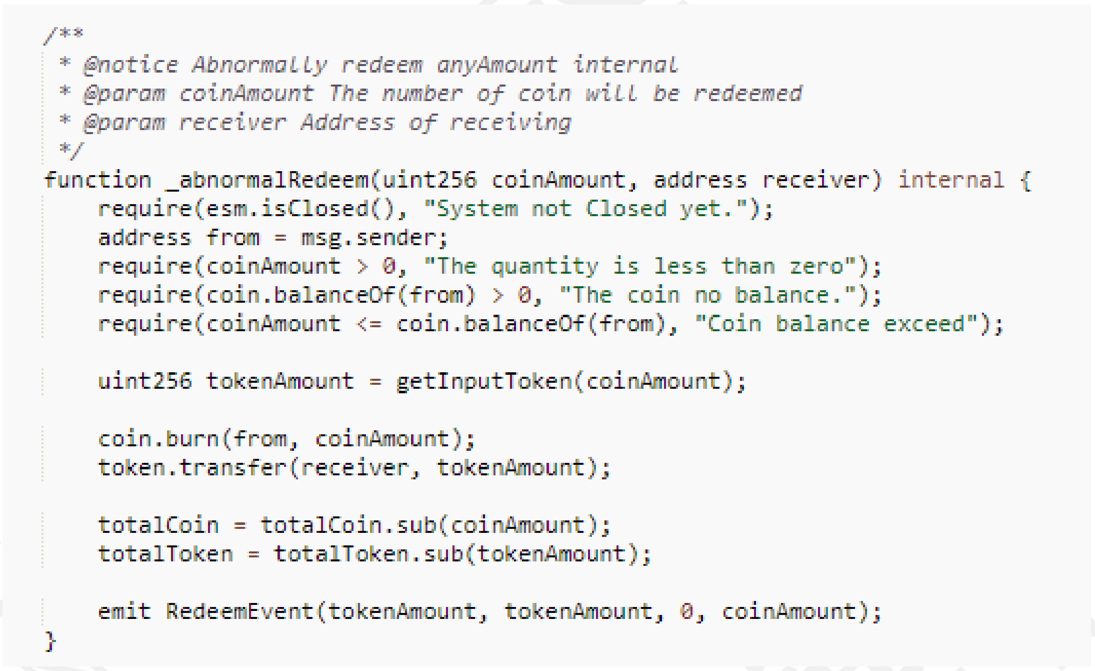
As shown in the figure below, the *_normalRedeem* function is called as different use in the public *redeem* functions (with different parameter number) and *redeemMax* functions (with different parameter number).



```
/**
 * @notice Normally redeem anyAmount
 * @param coinAmount The number of coin will be redeemed
 * @param receiver Address of receiving
 */
function redeem(uint256 coinAmount, address receiver) public {
    _normalRedeem(coinAmount, receiver);
}

/**
 * @notice Normally redeem anyAmount to msg.sender
 * @param coinAmount The number of coin will be redeemed
 */
function redeem(uint256 coinAmount) public {
    redeem(coinAmount, msg.sender);
}

/**
 * @notice normally redeem them all at once
 * @param holder reciver
 */
function redeemMax(address holder) public {
    redeem(coins[msg.sender], holder);
}

/**
 * @notice normally redeem them all at once to msg.sender
 */
function redeemMax() public {
    redeemMax(msg.sender);
}
```

Figure 15 source code of redeem functions and redeemMax functions

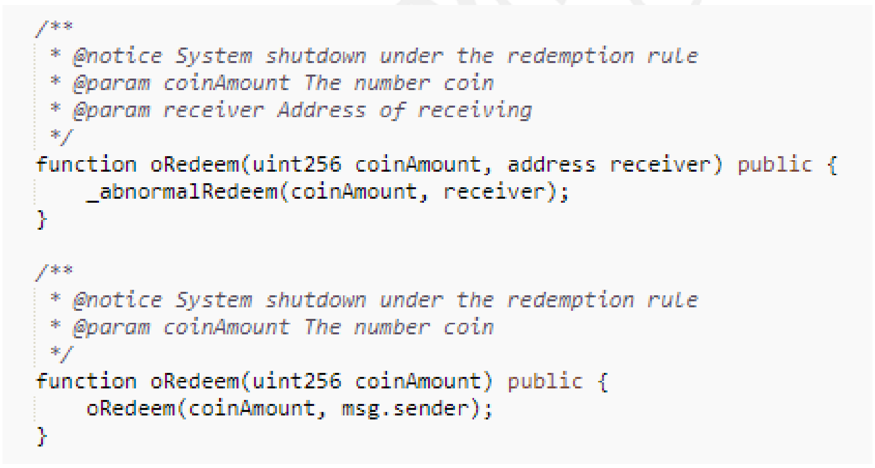
The *_abnormalRedeem* function requires that the redeem operation can only be executed when the “stake-reward” mode is closed. After checking the validity of redeem amount, then the corresponding OIN token redeem and USDO1 token destruction are performed. This internal function is called in different *oRedeem* functions (Figure 17), the main difference of these callings is that the redeem receiver address is passed differently.



```
/**
 * @notice Abnormally redeem anyAmount internal
 * @param coinAmount The number of coin will be redeemed
 * @param receiver Address of receiving
 */
function _abnormalRedeem(uint256 coinAmount, address receiver) internal {
    require(esm.isClosed(), "System not Closed yet.");
    address from = msg.sender;
    require(coinAmount > 0, "The quantity is less than zero");
    require(coin.balanceOf(from) > 0, "The coin no balance.");
    require(coinAmount <= coin.balanceOf(from), "Coin balance exceed");

    uint256 tokenAmount = getInputToken(coinAmount);

    coin.burn(from, coinAmount);
    token.transfer(receiver, tokenAmount);

    totalCoin = totalCoin.sub(coinAmount);
    totalToken = totalToken.sub(tokenAmount);

    emit RedeemEvent(tokenAmount, tokenAmount, 0, coinAmount);
}
```

Figure 16 source code of function _abnormalRedeem

As shown in the figure below, the *_abnormalRedeem* function is called as different use in the public o*Redeem* functions (with different parameter number).



```
/**
 * @notice System shutdown under the redemption rule
 * @param coinAmount The number coin
 * @param receiver Address of receiving
 */
function oRedeem(uint256 coinAmount, address receiver) public {
    _abnormalRedeem(coinAmount, receiver);
}

/**
 * @notice System shutdown under the redemption rule
 * @param coinAmount The number coin
 */
function oRedeem(uint256 coinAmount) public {
    oRedeem(coinAmount, msg.sender);
}
```

Figure 17 source code of oRedeem functions

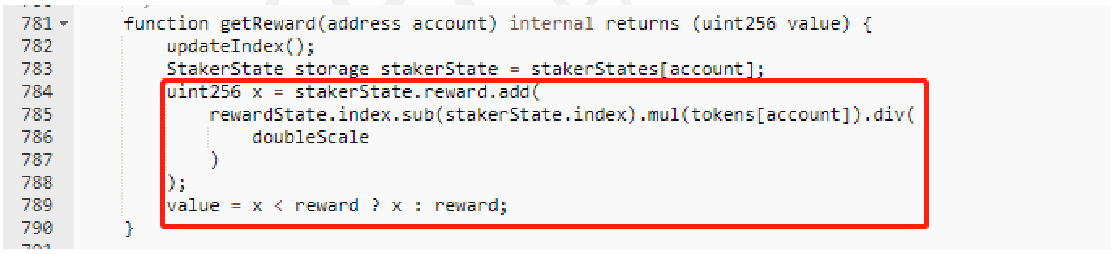
● Related functions: *_normalRedeem*, *_abnormalRedeem*, *isRedeemPaused*, *accuredToken*, *getInputToken*, *feeRate*, *updateIndex*, *getBlockNumber*, *_getReward*, *transfer*, *burn*, *redeem*, *redeemMax*, *oRedeem*

● Result: Pass

4.6.3 Reward update

● Description:

As shown in the figure below, the number of rewards obtained by this function is the current latest reward, and as shown in Figure 21, when the issue in Figure 21 is corrected, only the previously recorded reward quantity is deducted when this part of the reward is claimed. Not deducted according to the actual amount of rewards.



```
function getReward(address account) internal returns (uint256 value) {
    updateIndex();
    StakerState storage stakerState = stakerStates[account];
    uint256 x = stakerState.reward.add(
        rewardState.index.sub(stakerState.index).mul(tokens[account]).div(
            doubleScale
        )
    );
    value = x < reward ? x : reward;
}
```

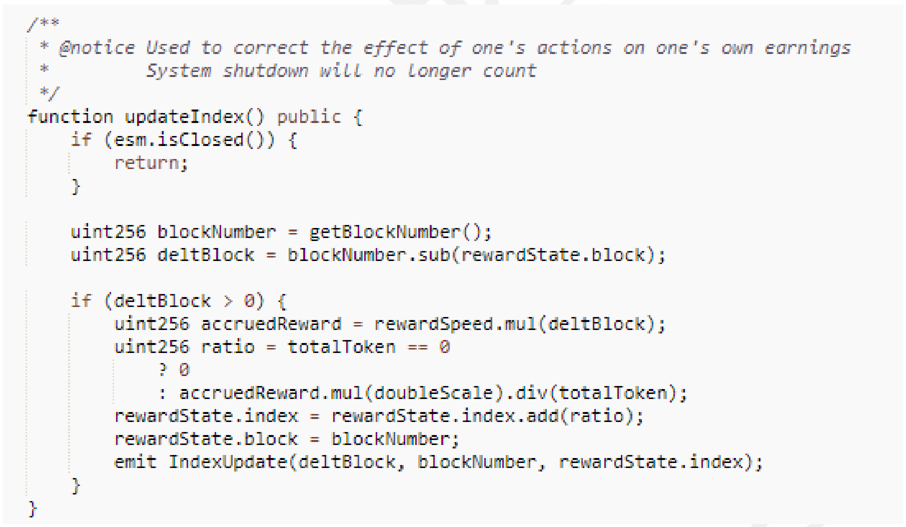
Figure 18 source code of function getReward (unfixed)

● Modify Recommendation: It is recommended to update the number of rewards recorded in this function, making the issue shown in Figure 7 can be deducted normally when deducting.

● Fix Result: Fixed. And the function is changed to the name of _getReward (Figure 20).

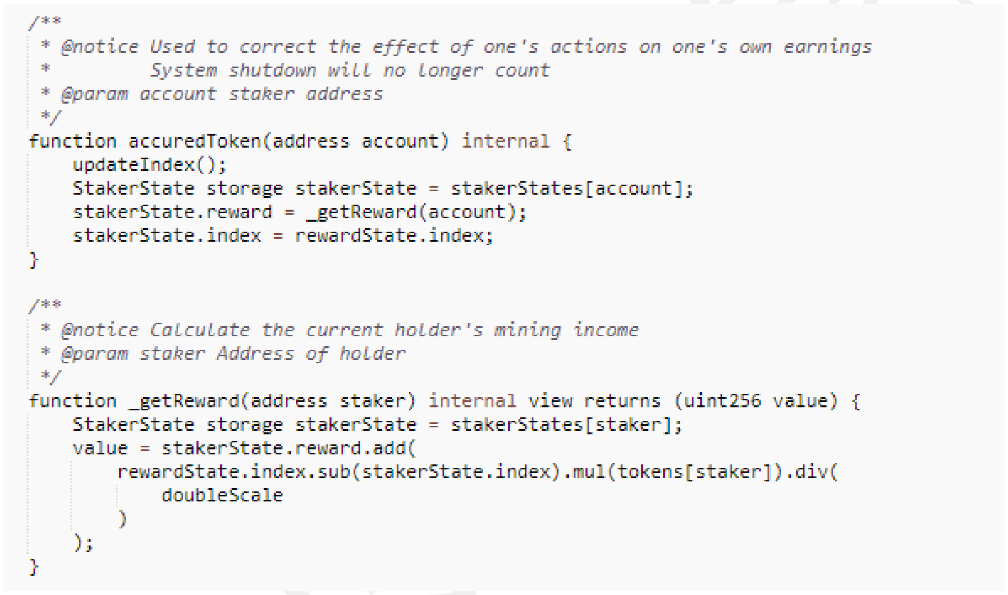
● Description of fixed code:

The OinStake implements the *updateIndex* and *accuredToken* functions to update the reward data. The *updateIndex* function updates the reward calculation related parameter data and the *accuredToken* function is called in other public/external functions to update the reward data of specified address.



```
/**
 * @notice Used to correct the effect of one's actions on one's own earnings
 *         System shutdown will no longer count
 */
function updateIndex() public {
    if (esm.isClosed()) {
        return;
    }

    uint256 blockNumber = getBlockNumber();
    uint256 deltBlock = blockNumber.sub(rewardState.block);

    if (deltBlock > 0) {
        uint256 accruedReward = rewardSpeed.mul(deltBlock);
        uint256 ratio = totalToken == 0
            ? 0
            : accruedReward.mul(doubleScale).div(totalToken);
        rewardState.index = rewardState.index.add(ratio);
        rewardState.block = blockNumber;
        emit IndexUpdate(deltBlock, blockNumber, rewardState.index);
    }
}
```

Figure 19 source code of function updateIndex



```
/**
 * @notice Used to correct the effect of one's actions on one's own earnings
 *         System shutdown will no longer count
 * @param account staker address
 */
function accuredToken(address account) internal {
    updateIndex();
    StakerState storage stakerState = stakerStates[account];
    stakerState.reward = _getReward(account);
    stakerState.index = rewardState.index;
}

/**
 * @notice Calculate the current holder's mining income
 * @param staker Address of holder
 */
function _getReward(address staker) internal view returns (uint256 value) {
    StakerState storage stakerState = stakerStates[staker];
    value = stakerState.reward.add(
        rewardState.index.sub(stakerState.index).mul(tokens[staker]).div(
            doubleScale
        )
    );
}
```

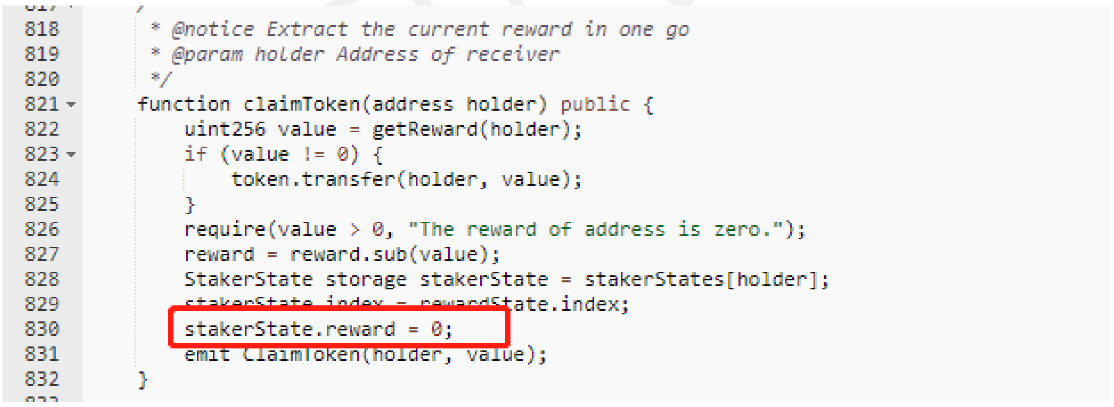
Figure 20 source code of functions accuredToken & _getReward

- Related functions: *updateIndex, getBlockNumber, totalToken, accuredToken, _getReward*
- Result: Pass

4.6.4 Claim reward tokens

- Description:

As shown in the figure below, when the user calls the *claimToken* function to extract the reward, because the OIN token balance of the contract address is not enough for rewards, the value obtained (actually reward) is less than the reward that the actual holder deserves, but the reward quantity is emptied after the claimed, resulting in the actually unreceived reward cannot be claimed; and this function can be called by any user. When the contract rewards (OIN token balance of this contract) are insufficient, other malicious users can perform the above operations to clear the rewards of the specified holder.



```
818      * @notice Extract the current reward in one go
819      * @param holder Address of receiver
820      */
821     function claimToken(address holder) public {
822         uint256 value = getReward(holder);
823         if (value != 0) {
824             token.transfer(holder, value);
825         }
826         require(value > 0, "The reward of address is zero.");
827         reward = reward.sub(value);
828         StakerState storage stakerState = stakerStates[holder];
829         stakerState.index = rewardState.index;
830         stakerState.reward = 0;
831         emit ClaimToken(holder, value);
832     }
```

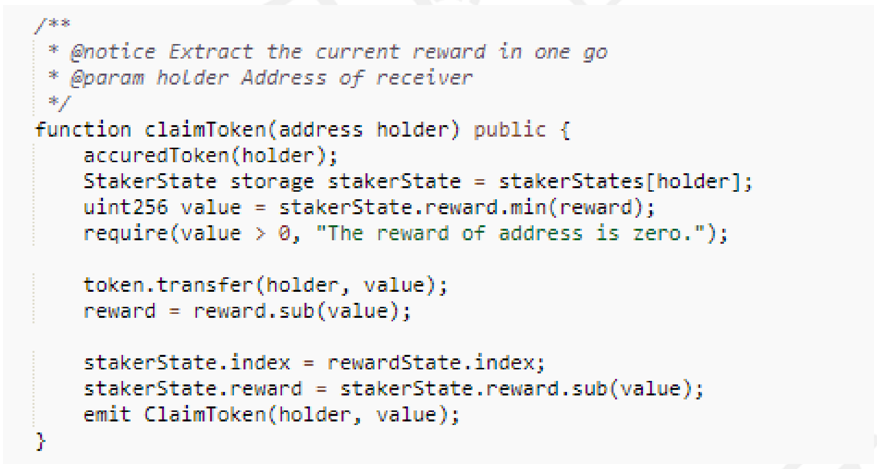
Figure 21 source code of function claimToken (unfixed)

- Modify Recommendation: It is recommended to change the algorithm in red marker box to subtraction. And replace the require check (line 826) to line 823, and delete the ‘if’ check with the same function to simplify code writing.

- Fix Result: Fixed. The new code is shown as following.

- Description of fixed code:

The OinStake implements the *claimToken* function for users to claim the OIN reward tokens. The *accuredToken* function is called to update the specified holder’s reward data, and the corresponding reward is deducted from the reward pool (when sufficient).



```
/**
 * @notice Extract the current reward in one go
 * @param holder Address of receiver
 */
function claimToken(address holder) public {
    accuredToken(holder);
    StakerState storage stakerState = stakerStates[holder];
    uint256 value = stakerState.reward.min(reward);
    require(value > 0, "The reward of address is zero.");

    token.transfer(holder, value);
    reward = reward.sub(value);

    stakerState.index = rewardState.index;
    stakerState.reward = stakerState.reward.sub(value);
    emit ClaimToken(holder, value);
}
```

Figure 22 source code of function claimToken (fixed)

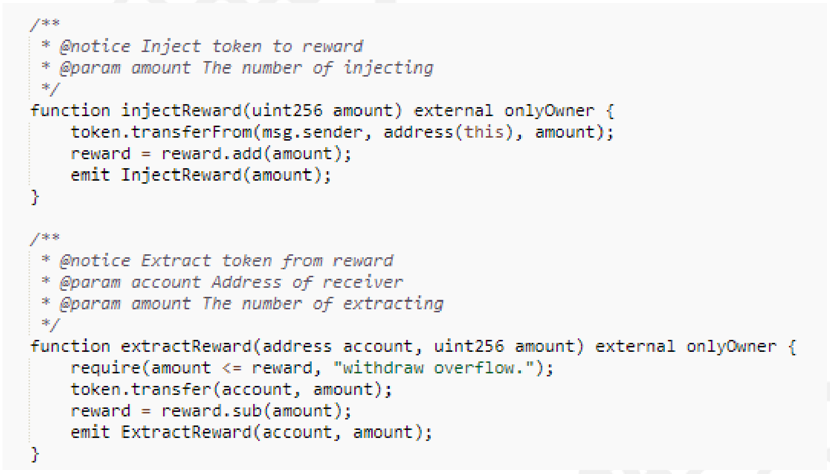
- Related functions: *claimToken*, *updateIndex*, *getBlockNumber*, *totalToken*, *accuredToken*, *_getReward*, *transfer*

- Result: Pass

4.6.5 Inject/extract reward pool

- Description:

The OinStake implements the *injectReward* and *extractReward* functions to operate the OIN reward amount in this contract. The contract owner call the function *injectReward* to transfer OIN tokens to this contract for reward distribution. And the *extractReward* function is called by the contract owner to extract the left OIN tokens in this contract to a specified address.



```
/**
 * @notice Inject token to reward
 * @param amount The number of injecting
 */
function injectReward(uint256 amount) external onlyOwner {
    token.transferFrom(msg.sender, address(this), amount);
    reward = reward.add(amount);
    emit InjectReward(amount);
}

/**
 * @notice Extract token from reward
 * @param account Address of receiver
 * @param amount The number of extracting
 */
function extractReward(address account, uint256 amount) external onlyOwner {
    require(amount <= reward, "withdraw overflow.");
    token.transfer(account, amount);
    reward = reward.sub(amount);
    emit ExtractReward(account, amount);
}
```

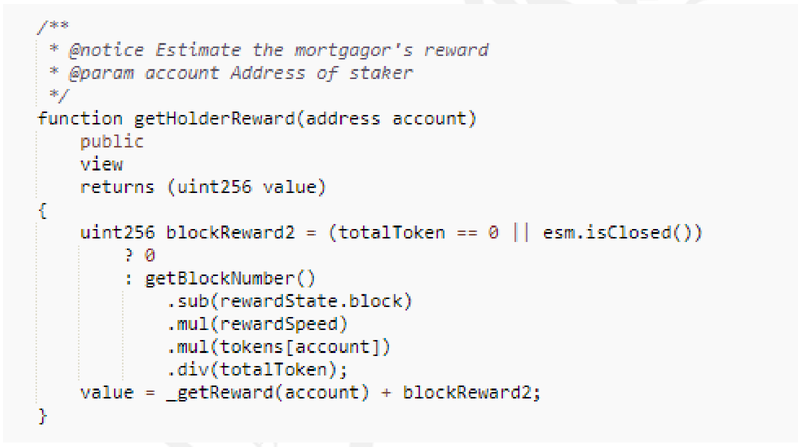
Figure 23 source code of functions injectReward & extractReward

- Related functions: *injectReward*, *extractReward*, *transferFrom*, *transfer*
- Result: Pass

4.6.6 Query functions

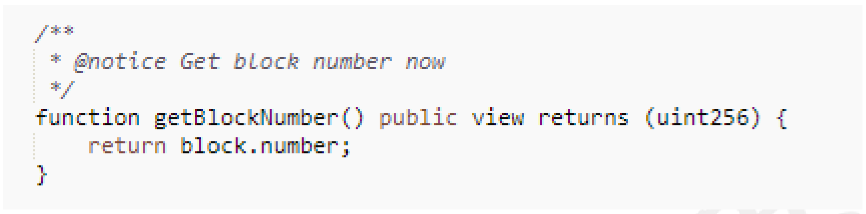
- Description:

The OinStake implements the following functions to query related data. Call the function *getHolderReward* to query the newest reward amount of specified address. Call the function *getBlockNumber* to query the current block number. Call the function *getInputToken* to query the corresponding calculated OIN token amount according to the specified USDO1 token amount. Call the function *debtOf* to query the staked OIN token amount and the correspondingly minted USDO1 token amount.



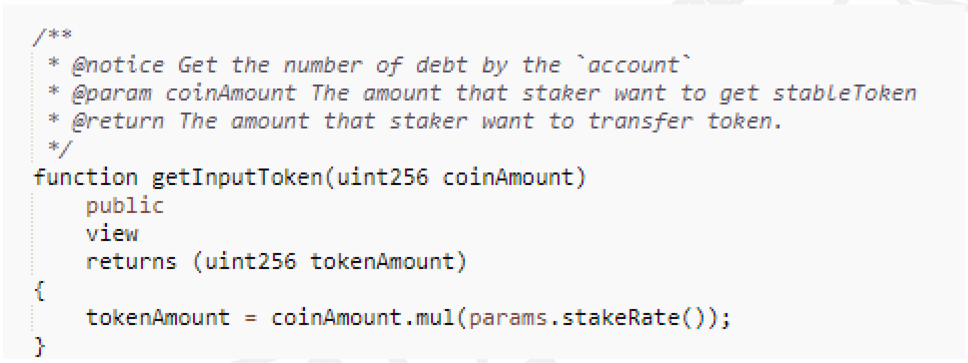
```
/**
 * @notice Estimate the mortgagor's reward
 * @param account Address of staker
 */
function getHolderReward(address account)
    public
    view
    returns (uint256 value)
{
    uint256 blockReward2 = (totalToken == 0 || esm.isClosed())
        ? 0
        : getBlockNumber()
            .sub(rewardState.block)
            .mul(rewardSpeed)
            .mul(tokens[account])
            .div(totalToken);
    value = _getReward(account) + blockReward2;
}
```

Figure 24 source code of function getHolderReward



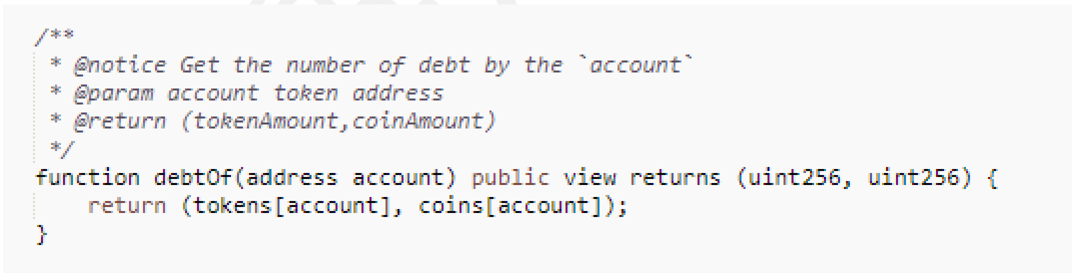
```
/**
 * @notice Get block number now
 */
function getBlockNumber() public view returns (uint256) {
    return block.number;
}
```

Figure 25 source code of function getBlockNumber



```
/**
 * @notice Get the number of debt by the `account`
 * @param coinAmount The amount that staker want to get stableToken
 * @return The amount that staker want to transfer token.
 */
function getInputToken(uint256 coinAmount)
    public
    view
    returns (uint256 tokenAmount)
{
    tokenAmount = coinAmount.mul(params.stakeRate());
}
```

Figure 26 source code of function getInputToken



```
/**
 * @notice Get the number of debt by the `account`
 * @param account token address
 * @return (tokenAmount,coinAmount)
 */
function debtOf(address account) public view returns (uint256, uint256) {
    return (tokens[account], coins[account]);
}
```

Figure 27 source code of function debtOf

- Related functions: *getHolderReward, getBlockNumber, getInputToken, debtOf*
- Result: Pass

5. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts project OinStake. All issues found during the audit have been repaired after being informed to the project side. The overall audit result of the smart contract project OinStake is **Pass**.

Beosin

成都链安

BEOSIN

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com