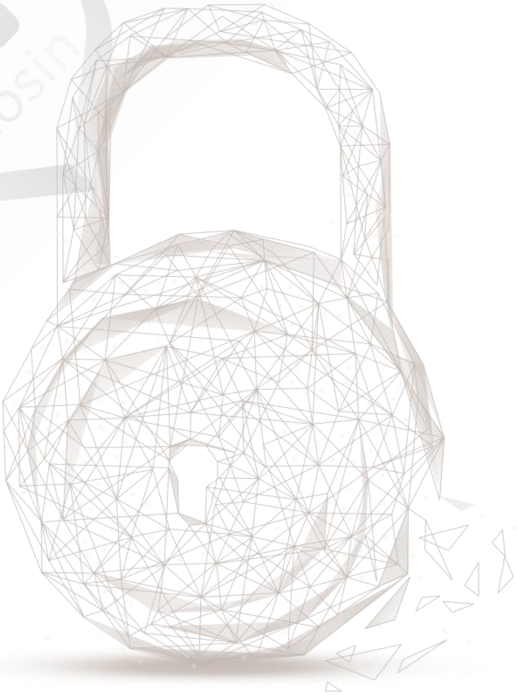
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202009071111

Smart Contract Name：

UniswapRewards

SegmentPowerStrategy

Smart Contract Address：

None

Smart Contract Address Link：

None

Start Date：2020.09.04

Completion Date：2020.09.07

Overall Result：Pass (Distinction)

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 SegmentPowerStrategy & UniswapRewards, including Coding Standards, Security, and Business Logic. **The SegmentPowerStrategy & UniswapRewards contract passed all audit items. The overall result is Pass (Distinction). 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

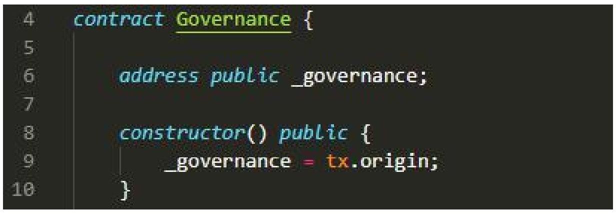
- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Result: Pass

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 Governance use the tx.origin as the initial governance address, it is safe.



```
contract Governance {

    address public _governance;

    constructor() public {
        _governance = tx.origin;
    }
```

Figure 1

- 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

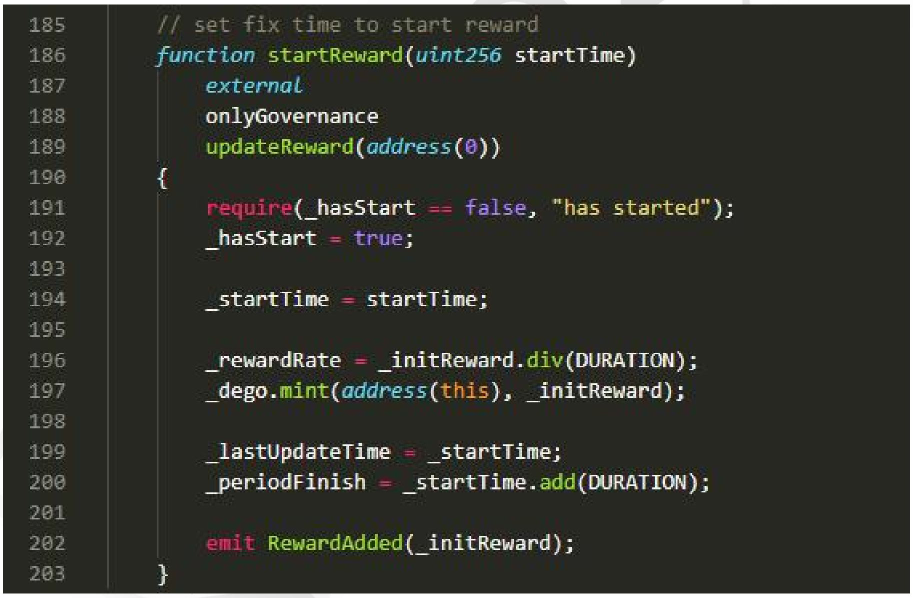
3. Business Security

Check whether the business is secure.

3.1 Stake Initialization

- Description:

  The "stake-reward" mode of the contract needs to initialize the relevant parameters (*_rewardRate, _lastUpdateTime, _periodFinish, _startTime*), call the *startReward* function by the specified governance address, and enter the initial start time used to check whether the "stake-reward" mode has started, initialize the stake and reward related parameters.



```
// set fix time to start reward
function startReward(uint256 startTime)
    external
    onlyGovernance
    updateReward(address(0))
{
    require(_hasStart == false, "has started");
    _hasStart = true;

    _startTime = startTime;

    _rewardRate = _initReward.div(DURATION);
    _dego.mint(address(this), _initReward);

    _lastUpdateTime = _startTime;
    _periodFinish = _startTime.add(DURATION);

    emit RewardAdded(_initReward);
}
```

Figure 2 source code of function startReward

- Related functions: *startReward, updateReward, rewardPerToken, lastTimeRewardApplicable, totalPower*

- Result: Pass

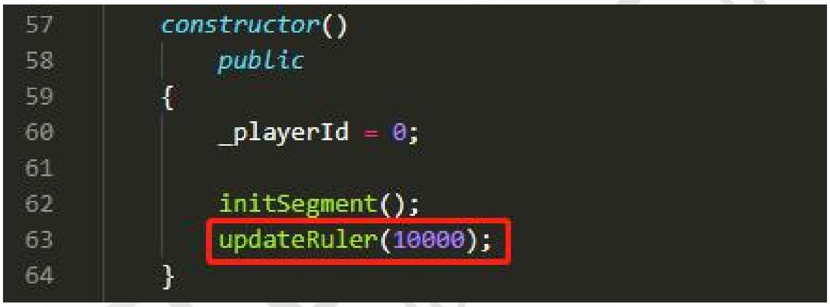
3.2 Stake LIQUIDITY POOL tokens

- Description:

  The UniswapRewards contract implements the *stake* function to stake the LIQUIDITY POOL tokens. The stake player pre-approve the contract address. By calling the *transferFrom* function in the specified LIQUIDITY POOL token contract, the contract address transfers the specified amount of

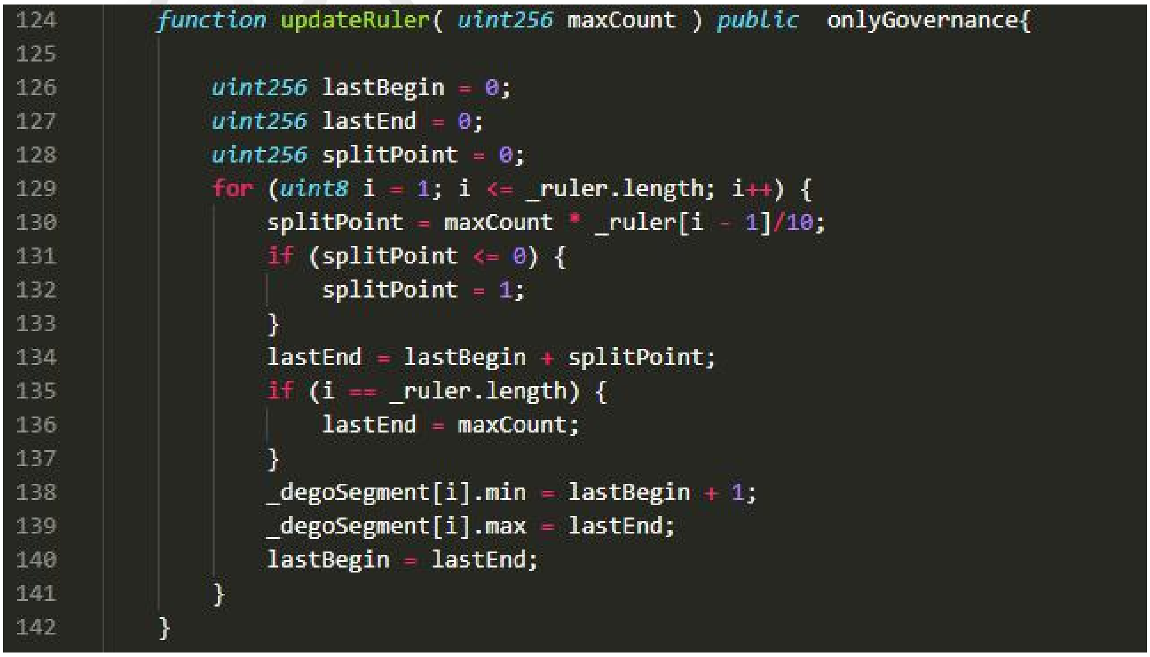
LIQUIDITY POOL tokens to the contract address on behalf of the stake player; This function restricts the stake player to call only after the "stake-reward" mode is turned on (the specified time is reached); each time this function is called to stake tokens, the reward related data is updated through the modifier *updateReward*; and each call is checked whether the *periodFinish* is reached by the modifier *checkhalve*, and the reward halving operation is performed and the *rewardRate* and the *periodFinish* are updated.

The SegmentPowerStrategy contract implements the *lpIn* function to record/update the stake information, the stake players are divided into 3 level in the SegmentPowerStrategy contract, and the reward is calculated according to their corresponding level (segment/segIndex). As shown in the Figure 3&4 below, the constructor uses the value of 10000 to initialize the segment ruler, it does not convert decimals with 1e^{18}. If player stake at least 9.001*1e^{-15} LIQUIDITY POOL tokens, he/she will join the high level (segment), it will cause the high level (segment) slot to be full firstly. The function *updateRuler* can be called by governance address to update segment standard manually.



```
    constructor()
        public
    {
        _playerId = 0;

        initSegment();
        updateRuler(10000);
    }
```

Figure 3 source code of constructor in contract SegmentPowerStrategy



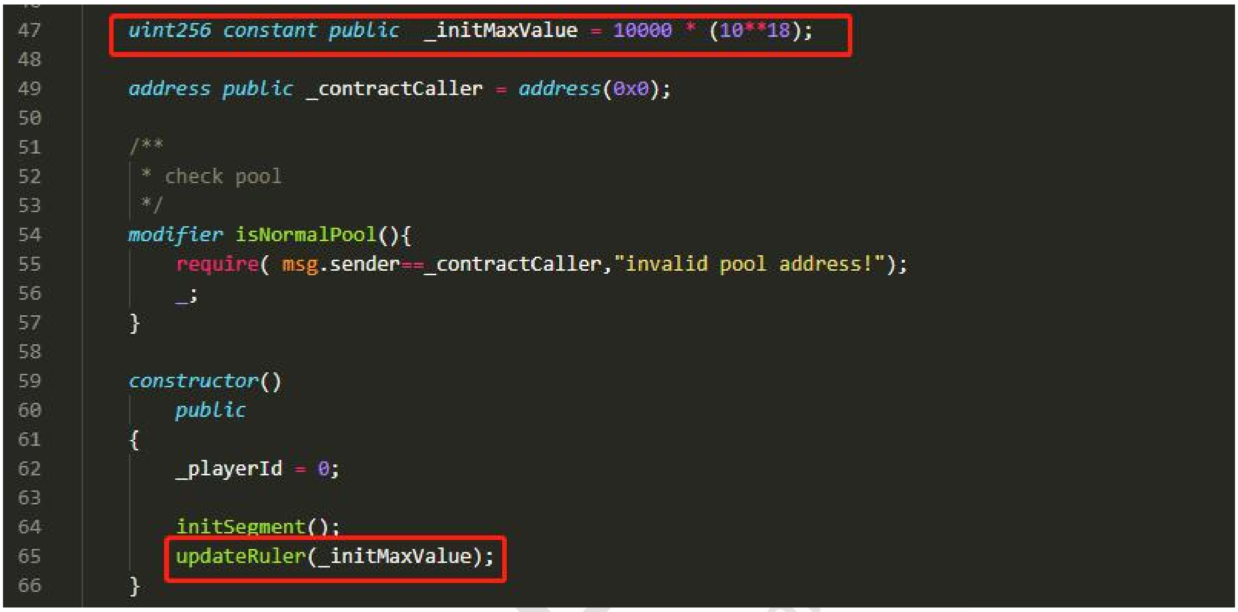
```
    function updateRuler( uint256 maxCount ) public  onlyGovernance{

        uint256 lastBegin = 0;
        uint256 lastEnd = 0;
        uint256 splitPoint = 0;
        for (uint8 i = 1; i <= _ruler.length; i++) {
            splitPoint = maxCount * _ruler[i - 1]/10;
            if (splitPoint <= 0) {
                splitPoint = 1;
            }
            lastEnd = lastBegin + splitPoint;
            if (i == _ruler.length) {
                lastEnd = maxCount;
            }
            _degoSegment[i].min = lastBegin + 1;
            _degoSegment[i].max = lastEnd;
            lastBegin = lastEnd;
        }
    }
```

Figure 4 source code of function updateRuler in contract SegmentPowerStrategy

- Related functions: *updateRuler*

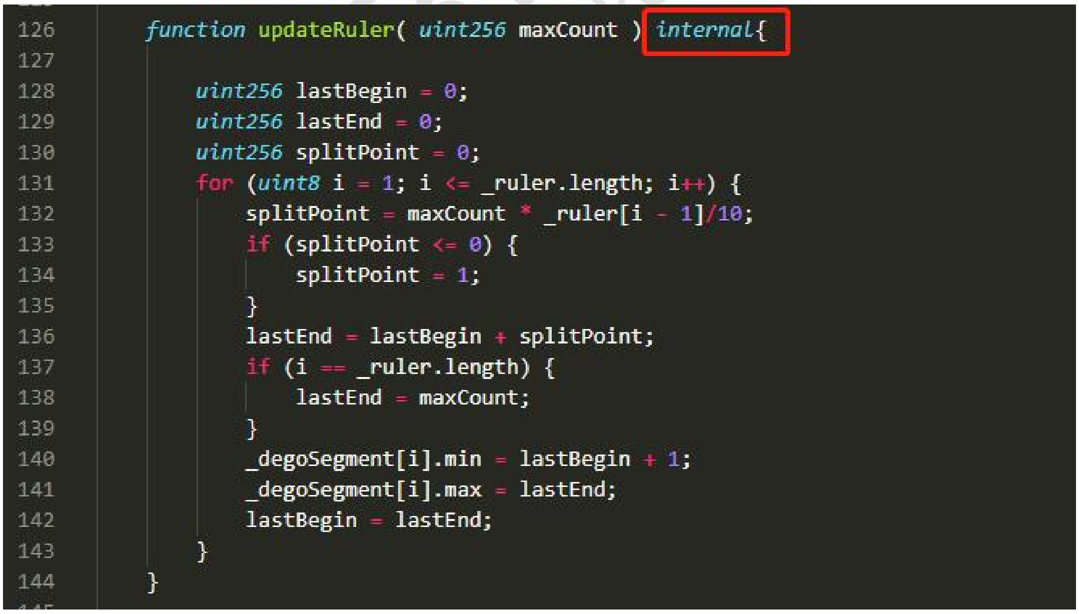
- Modify Recommendation: Converting decimals with 1e^{18} in the function *updateRuler* is recommended.

- Result: Fixed. The fixed code is shown in Figure 5 below. After update, the visibility of function *updateRuler* is change to internal (Figure 6), it means that the segment ruler cannot be updated manually.



```
uint256 constant public  _initMaxValue = 10000 * (10**18);

address public _contractCaller = address(0x0);

/**
 * check pool
 */
modifier isNormalPool(){
    require( msg.sender==_contractCaller,"invalid pool address!");
    _;
}

constructor()
    public
{
    _playerId = 0;

    initSegment();
    updateRuler(_initMaxValue);
}
```

Figure 5 the fixed source code of initializing segment ruler

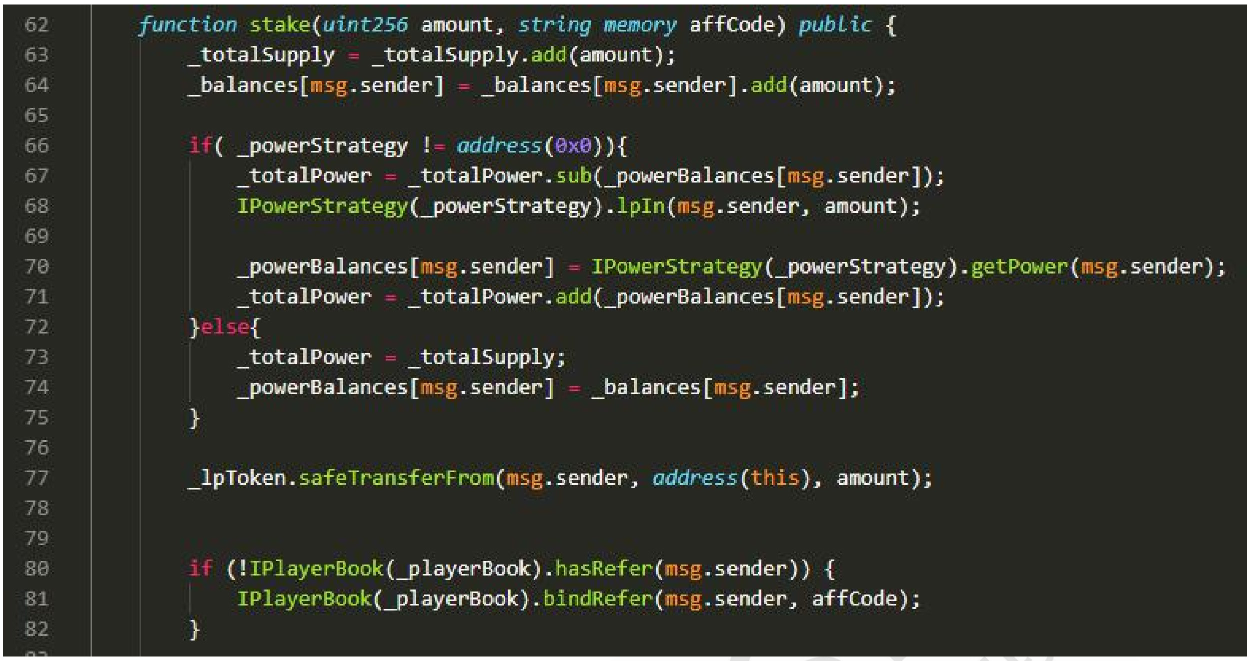


```
function updateRuler( uint256 maxCount ) internal{

    uint256 lastBegin = 0;
    uint256 lastEnd = 0;
    uint256 splitPoint = 0;
    for (uint8 i = 1; i <= _ruler.length; i++) {
        splitPoint = maxCount * _ruler[i - 1]/10;
        if (splitPoint <= 0) {
            splitPoint = 1;
        }
        lastEnd = lastBegin + splitPoint;
        if (i == _ruler.length) {
            lastEnd = maxCount;
        }
        _degoSegment[i].min = lastBegin + 1;
        _degoSegment[i].max = lastEnd;
        lastBegin = lastEnd;
    }
}
```

Figure 6 The source code of fixed function updateRuler in contract SegmentPowerStrategy

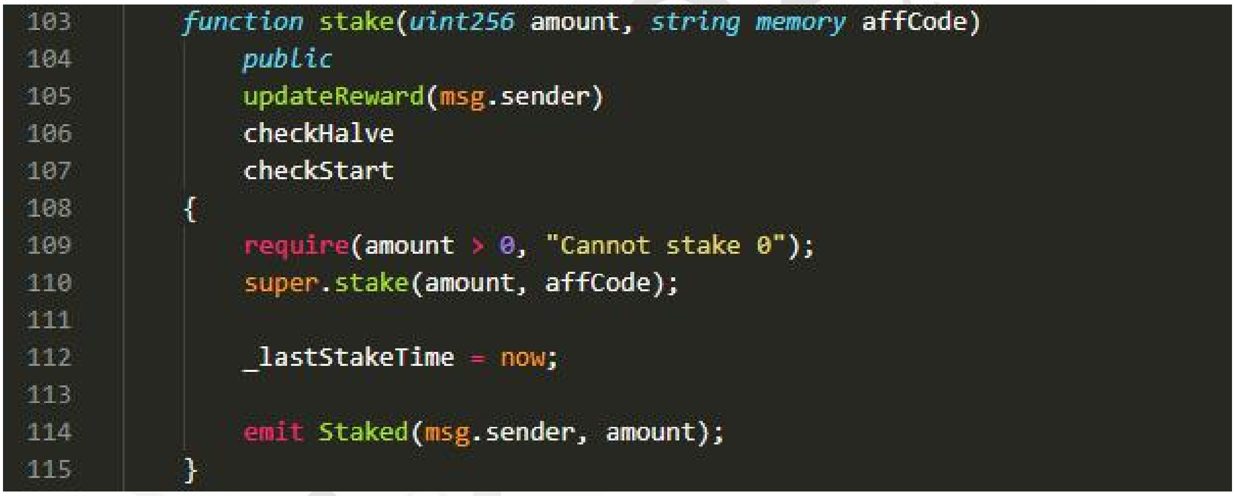
- Description:

  The PlayerBook contract implements the corresponding functions to record the referral information after stake player staked LIQUIDITY POOL tokens.



```
    function stake(uint256 amount, string memory affCode) public {
        _totalSupply = _totalSupply.add(amount);
        _balances[msg.sender] = _balances[msg.sender].add(amount);

        if( _powerStrategy != address(0x0)){
            _totalPower = _totalPower.sub(_powerBalances[msg.sender]);
            IPowerStrategy(_powerStrategy).lpIn(msg.sender, amount);

            _powerBalances[msg.sender] = IPowerStrategy(_powerStrategy).getPower(msg.sender);
            _totalPower = _totalPower.add(_powerBalances[msg.sender]);
        }else{
            _totalPower = _totalSupply;
            _powerBalances[msg.sender] = _balances[msg.sender];
        }

        _lpToken.safeTransferFrom(msg.sender, address(this), amount);


        if (!IPlayerBook(_playerBook).hasRefer(msg.sender)) {
            IPlayerBook(_playerBook).bindRefer(msg.sender, affCode);
        }
```

Figure 7 source code of function stake in contract LPTokenWrapper



```
    function stake(uint256 amount, string memory affCode)
        public
        updateReward(msg.sender)
        checkHalve
        checkStart
    {
        require(amount > 0, "Cannot stake 0");
        super.stake(amount, affCode);

        _lastStakeTime = now;

        emit Staked(msg.sender, amount);
    }
```

Figure 8 source code of function stake in contract UniswapReward

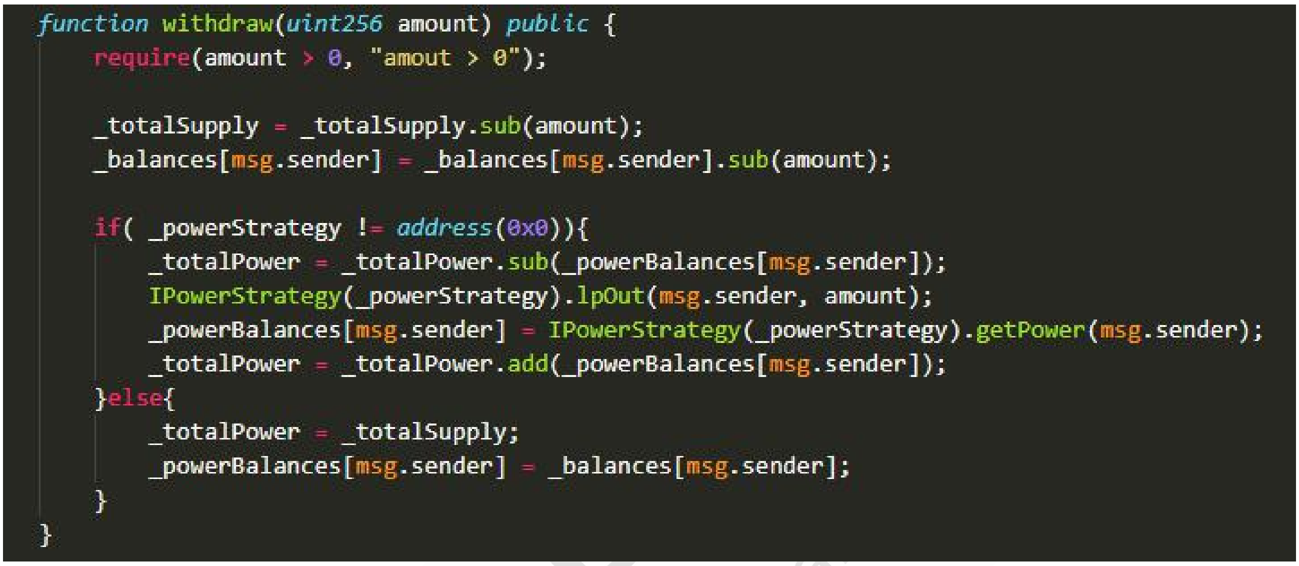
- Related functions: *stake, rewardPerToken, lastTimeRewardApplicable, earned, balanceOfPower, totalPower, lpIn, getPower*
- Result: Pass

3.3 Withdraw LIQUIDITY POOL tokens

- Description:

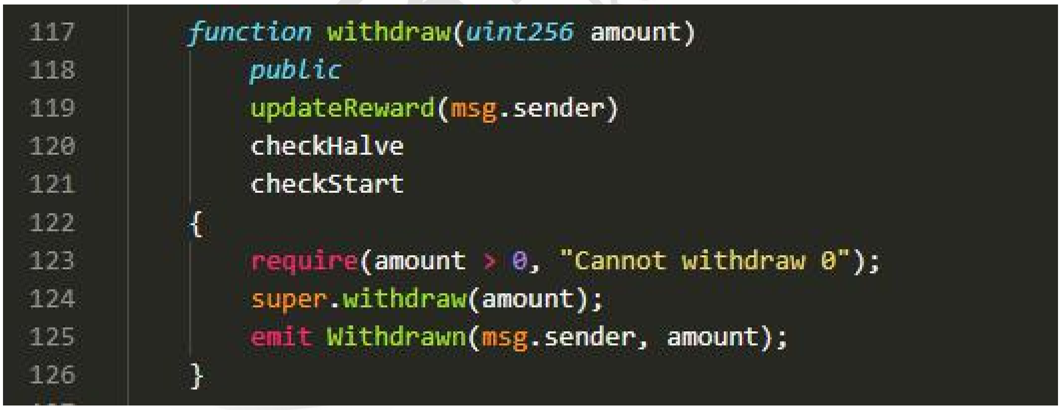
The UniswapRewards contract implements the *withdraw* function to withdraw the LIQUIDITY POOL tokens. **This function lacks the important operation of transferring LIQUIDITY POOL tokens to the caller.** This function restricts the stake player to call only after the "stake-reward" mode is turned on (the specified time is reached); each time this function is called to stake tokens, the reward related data is updated through the modifier *updateReward*; and each call is checked whether the *periodFinish* is reached by the modifier *checkhalve*, and the reward halving operation is performed and the *rewardRate* and the *periodFinish* are updated.

The SegmentPowerStrategy contract implements the *lpOut* function to update the stake information, the stake players are divided into 3 level in the SegmentPowerStrategy contract, and the reward is calculated according to their corresponding level (segment/segIndex).



```
function withdraw(uint256 amount) public {
    require(amount > 0, "amout > 0");

    _totalSupply = _totalSupply.sub(amount);
    _balances[msg.sender] = _balances[msg.sender].sub(amount);

    if( _powerStrategy != address(0x0)){
        _totalPower = _totalPower.sub(_powerBalances[msg.sender]);
        IPowerStrategy(_powerStrategy).lpOut(msg.sender, amount);
        _powerBalances[msg.sender] = IPowerStrategy(_powerStrategy).getPower(msg.sender);
        _totalPower = _totalPower.add(_powerBalances[msg.sender]);
    }else{
        _totalPower = _totalSupply;
        _powerBalances[msg.sender] = _balances[msg.sender];
    }
}
```

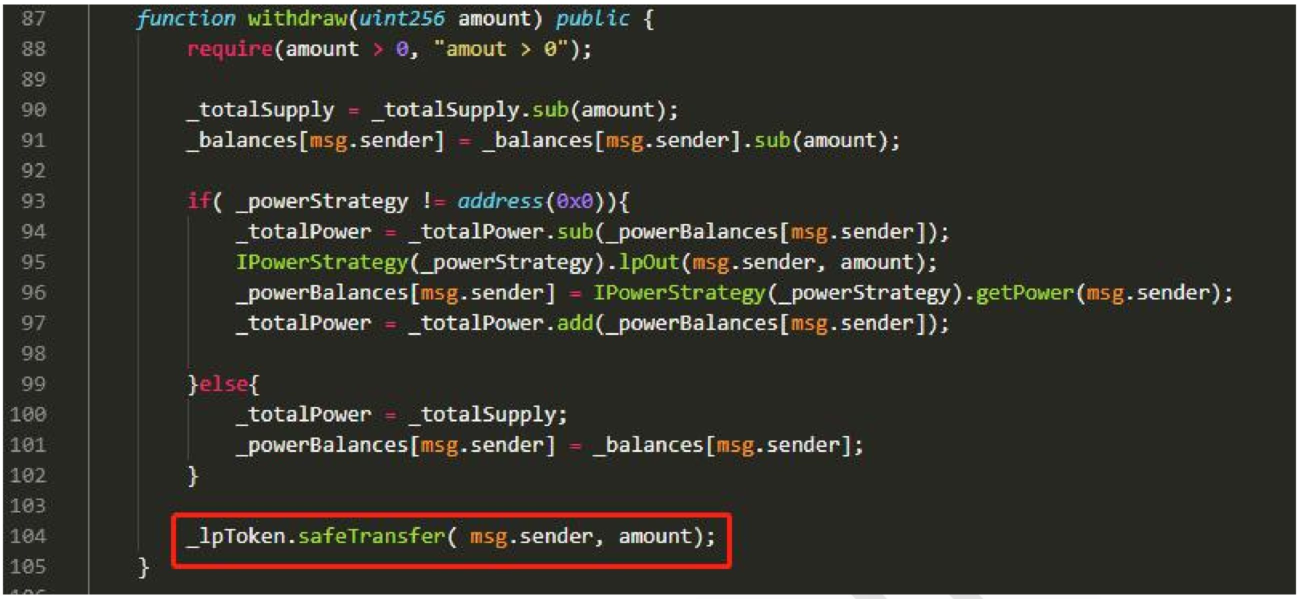
Figure 9 source code of function withdraw in contract LPTokenWrapper



```
    function withdraw(uint256 amount)
        public
        updateReward(msg.sender)
        checkHalve
        checkStart
    {
        require(amount > 0, "Cannot withdraw 0");
        super.withdraw(amount);
        emit Withdrawn(msg.sender, amount);
    }
```

Figure 10 source code of function withdraw in contract UniswapReward

- Related functions: *withdraw, rewardPerToken, lastTimeRewardApplicable, earned, balanceOfPower, totalPower, lpOut, getPower*
- Modify Recommendation: Adding the corresponding operation of transferring out (withdrawing) the staked LIQUIDITY POOL tokens to caller is recommended.
- Result: Fixed. The fixed code is shown in Figure 11 below.



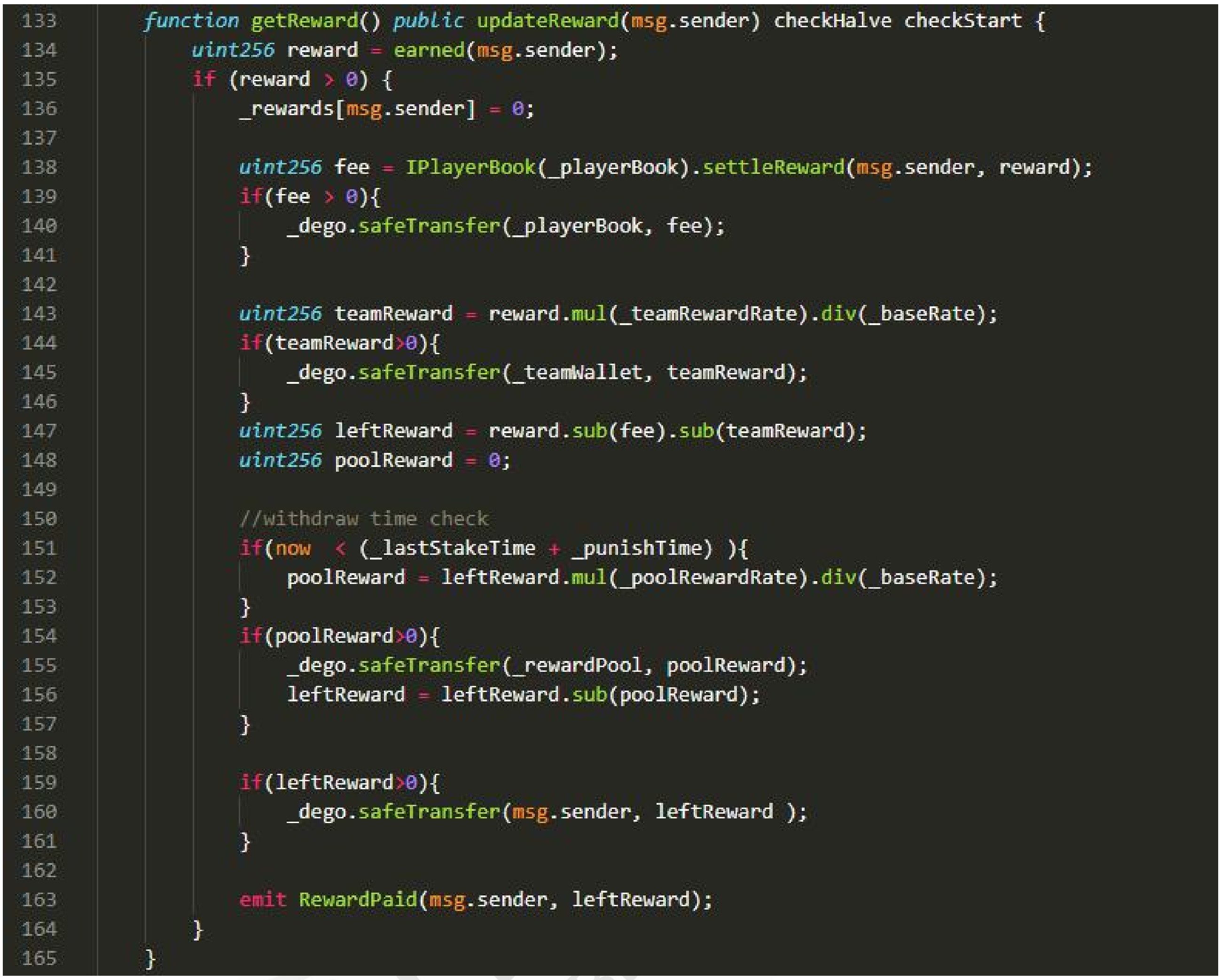
```
function withdraw(uint256 amount) public {
    require(amount > 0, "amout > 0");

    _totalSupply = _totalSupply.sub(amount);
    _balances[msg.sender] = _balances[msg.sender].sub(amount);

    if( _powerStrategy != address(0x0)){
        _totalPower = _totalPower.sub(_powerBalances[msg.sender]);
        IPowerStrategy(_powerStrategy).lpOut(msg.sender, amount);
        _powerBalances[msg.sender] = IPowerStrategy(_powerStrategy).getPower(msg.sender);
        _totalPower = _totalPower.add(_powerBalances[msg.sender]);

    }else{
        _totalPower = _totalSupply;
        _powerBalances[msg.sender] = _balances[msg.sender];
    }

    _lpToken.safeTransfer( msg.sender, amount);
}
```

Figure 11 The source code of fixed function withdraw in contract LPTokenWrapper

3.4 Withdraw rewards

- Description:

The UniswapReward contract implements the *getReward* function to withdraw the rewards (DEGO token). By calling the *transfer* function in the DEGO contract, the contract address transfers the specified amount (left rewards of caller) of DEGO tokens to the stake player; This function calculates and deducts the referral reward amount from the caller total rewards, the all referral rewards are sent to the PlayerBook contract for the corresponding referral's claiming; This function calculates and deducts the team reward amount from the caller total rewards, the all team rewards are sent to the specified team wallet address. In addition, if the current time is earlier than the timestamp of 3 days later from the last stake time, the pool reward will be calculated according the specified data, the all poll rewards are sent to the specified reward pool address. This function restricts the stake player to call only after the "stake-reward" mode is turned on (the specified time is reached); each time this function is called to stake tokens, the reward related data is updated through the modifier *updateReward*; and each call is checked whether the *periodFinish* is reached by the modifier *checkhalve*, and the reward halving operation is performed and the *rewardRate* and the *periodFinish* are updated.



```
    function getReward() public updateReward(msg.sender) checkHalve checkStart {
        uint256 reward = earned(msg.sender);
        if (reward > 0) {
            _rewards[msg.sender] = 0;

            uint256 fee = IPlayerBook(_playerBook).settleReward(msg.sender, reward);
            if(fee > 0){
                _dego.safeTransfer(_playerBook, fee);
            }

            uint256 teamReward = reward.mul(_teamRewardRate).div(_baseRate);
            if(teamReward>0){
                _dego.safeTransfer(_teamWallet, teamReward);
            }
            uint256 leftReward = reward.sub(fee).sub(teamReward);
            uint256 poolReward = 0;

            //withdraw time check
            if(now  < (_lastStakeTime + _punishTime) ){
                poolReward = leftReward.mul(_poolRewardRate).div(_baseRate);
            }
            if(poolReward>0){
                _dego.safeTransfer(_rewardPool, poolReward);
                leftReward = leftReward.sub(poolReward);
            }

            if(leftReward>0){
                _dego.safeTransfer(msg.sender, leftReward );
            }

            emit RewardPaid(msg.sender, leftReward);
        }
    }
```

Figure 12 source code of function getReward

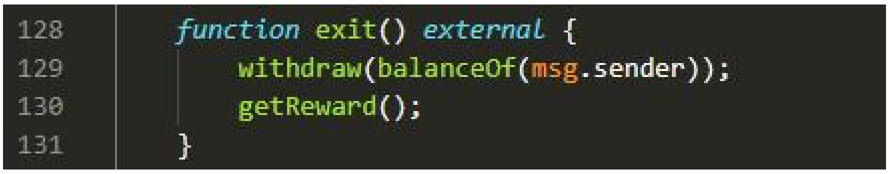
- Related functions: *getReward, rewardPerToken, lastTimeRewardApplicable, earned, balanceOfPower, totalPower, settleReward*

- Result: Pass

3.5 Exit the stake participation

- Description:

The contract implements the *exit* function to close the participation of "stake-reward" mode. Call the *withdraw* function to withdraw all stake LIQUIDITY POOL tokens, call the *getReward* function to receive all rewards. The stake player address cannot get new rewards because the balance of LIQUIDITY POOL token tokens already staked is empty.



```
    function exit() external {
        withdraw(balanceOf(msg.sender));
        getReward();
    }
```

Figure 13 source code of function exit

- Related functions: *exit, withdraw, getReward, rewardPerToken, lastTimeRewardApplicable, earned, balanceOfPower, totalPower, settleReward*

- Result: Pass

3.6 Reward related data query function

- Description:

Contract stake players can query the earliest timestamp between the current timestamp and the *periodFinish* by calling the *lastTimeRewardApplicable* function; calling the *rewardPerToken* function can query the gettable rewards for each stake LIQUIDITY POOL token; calling the *earned* function can query the total gettable stake rewards of the specified address.

- Related functions: *lastTimeRewardApplicable, rewardPerToken, earned*

- Result: Pass

Beosin

BEOSIN

Blockchain Security

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com