BEOSIN
Blockchain Security

Smart contract security audit report

Audit Number：202103261340

Report query name: Goswap

Smart Contract Info：

Smart Contract Name	Smart Contract Address	Smart Contract Address Link
MasterChef	0x7dCeBC34F55b52df742C91581089ebD0BCBD254F	https://hecoinfo.com/address/0x7dCeBC34F55b52df742C91581089ebD0BCBD254F#code
LPTokenPool	0xC33F68fBBCB529faB10aB5FcFD77BaD7cE9fbfFA	https://hecoinfo.com/address/0xC33F68fBBCB529faB10aB5FcFD77BaD7cE9fbfFA#code

Start Date：2021.03.22

Completion Date：2021.03.26

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

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 Goswap, including Coding Standards, Security, and Business Logic. **The Goswap project passed all audit items. The overall result is Pass. The smart contract is able to function properly.**

Audit Contents:

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 HT.
- 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.
- 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

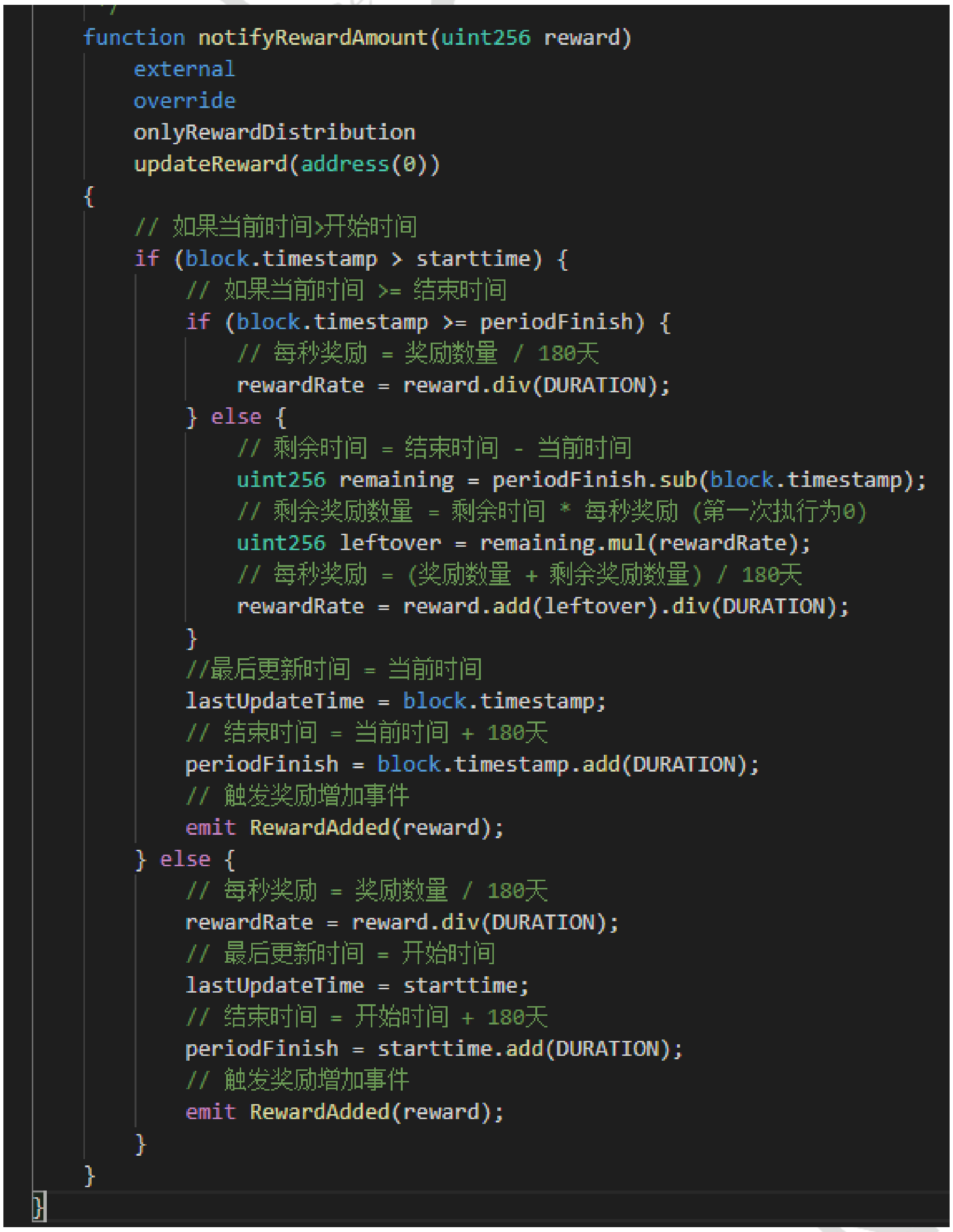
- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

3. Business Security

3.1 LPTokenPool Contract audit

(1) notifyRewardAmount function

- Description: The "Stake-Award" mode of the contract needs to initialize the relevant parameters (award ratio *rewardRate*, first update time *lastUpdateTime*, phase completion time *periodFinish*), calls the *notifyRewardAmount* function through the specified award allocation administrator address *rewardDistribution*, enter the initial award value reward, used to calculate the award ratio initialize the stake and award related parameters.



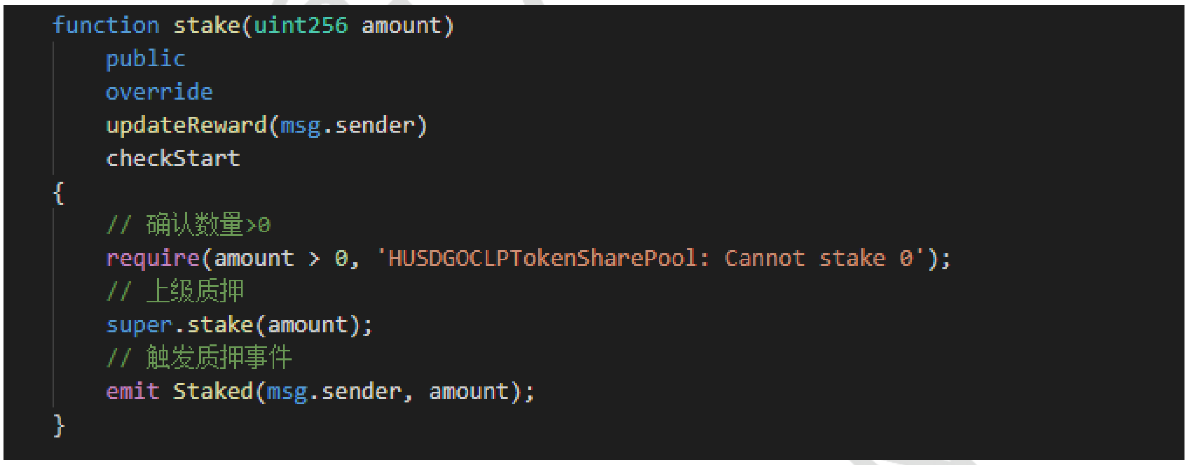
```
    function notifyRewardAmount(uint256 reward)
        external
        override
        onlyRewardDistribution
        updateReward(address(0))
    {
        // 如果当前时间>开始时间
        if (block.timestamp > starttime) {
            // 如果当前时间 >= 结束时间
            if (block.timestamp >= periodFinish) {
                // 每秒奖励 = 奖励数量 / 180天
                rewardRate = reward.div(DURATION);
            } else {
                // 剩余时间 = 结束时间 - 当前时间
                uint256 remaining = periodFinish.sub(block.timestamp);
                // 剩余奖励数量 = 剩余时间 * 每秒奖励 (第一次执行为0)
                uint256 leftover = remaining.mul(rewardRate);
                // 每秒奖励 = (奖励数量 + 剩余奖励数量) / 180天
                rewardRate = reward.add(leftover).div(DURATION);
            }
            //最后更新时间 = 当前时间
            lastUpdateTime = block.timestamp;
            // 结束时间 = 当前时间 + 180天
            periodFinish = block.timestamp.add(DURATION);
            // 触发奖励增加事件
            emit RewardAdded(reward);
        } else {
            // 每秒奖励 = 奖励数量 / 180天
            rewardRate = reward.div(DURATION);
            // 最后更新时间 = 开始时间
            lastUpdateTime = starttime;
            // 结束时间 = 开始时间 + 180天
            periodFinish = starttime.add(DURATION);
            // 触发奖励增加事件
            emit RewardAdded(reward);
        }
    }
}
```

Figure 1 source code of *notifyRewardAmount*

- Related functions: *notifyRewardAmount*
- Result: Pass

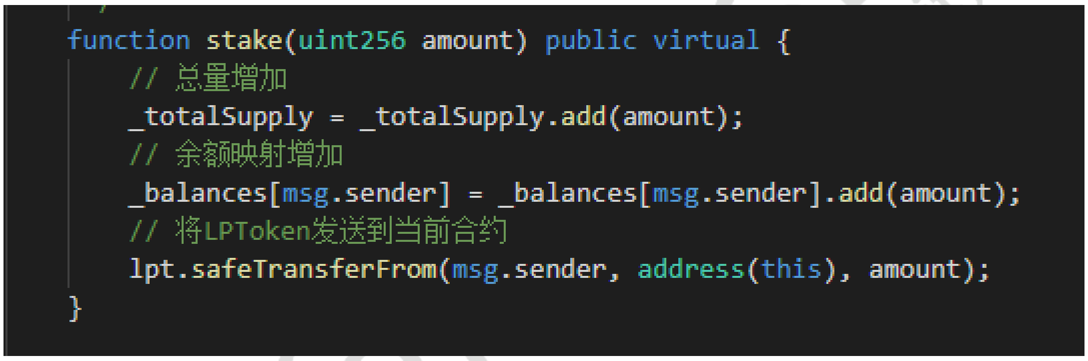
(2) Stake function(lpt)

- Description: The contract implements the *stake* function for the stake token(lpt), and the user authorizes the contract address in advance. By calling the *safetransferFrom* function in the contract, each time the function stake token is called(Check whether the start time is reached through the decorator *checkStart*), the reward-related data is updated through the modifier *updateReward*.



```
function stake(uint256 amount)
    public
    override
    updateReward(msg.sender)
    checkStart
{
    // 确认数量>0
    require(amount > 0, 'HUSDGOCLPTokenSharePool: Cannot stake 0');
    // 上级质押
    super.stake(amount);
    // 触发质押事件
    emit Staked(msg.sender, amount);
}
```

Figure 2 source code of *stake*(1/2)



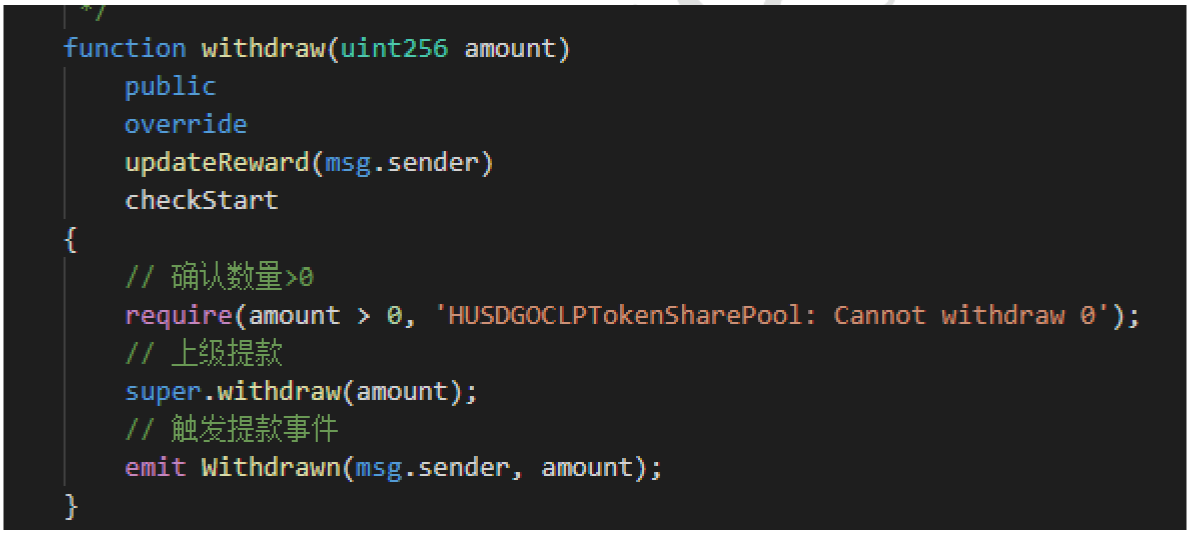
```
function stake(uint256 amount) public virtual {
    // 总量增加
    _totalSupply = _totalSupply.add(amount);
    // 余额映射增加
    _balances[msg.sender] = _balances[msg.sender].add(amount);
    // 将LPToken发送到当前合约
    lpt.safeTransferFrom(msg.sender, address(this), amount);
}
```

Figure 3 source code of *stake*(2/2)

- Related functions: *stake*, *updateReward*, *safetransferFrom*
- Result: Pass

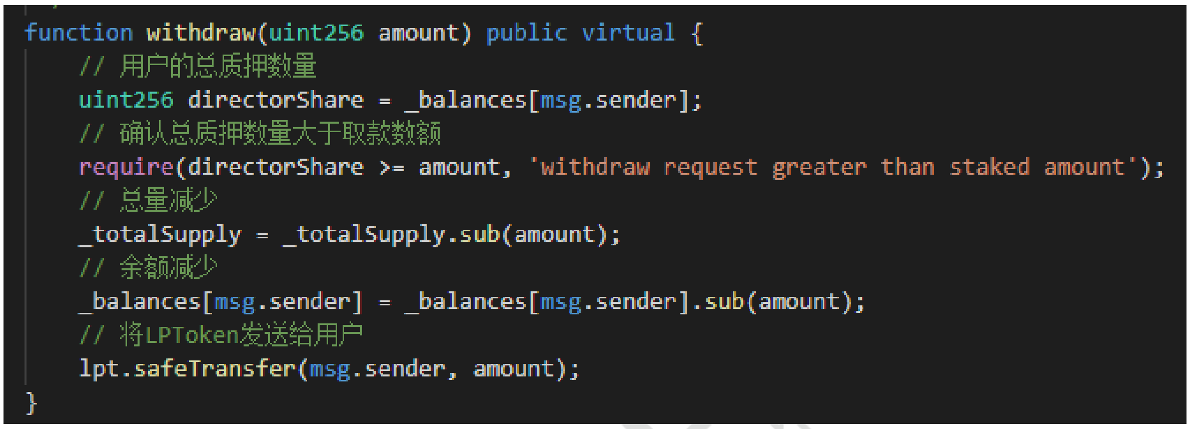
(3) Withdraw function(lpt)

- Description: The contract implements the *withdraw* function to extract the staked token(lpt). By calling the *transfer* function in the contract, the contract address transfers the specified number of tokens to the function caller (user) address; each time the function is called to extract the token(Check whether the start time is reached through the decorator *checkStart*), the reward data is updated through the modifier *updateReward*.



```
    */
    function withdraw(uint256 amount)
        public
        override
        updateReward(msg.sender)
        checkStart
    {
        // 确认数量>0
        require(amount > 0, 'HUSDGOCLPTokenSharePool: Cannot withdraw 0');
        // 上级提款
        super.withdraw(amount);
        // 触发提款事件
        emit Withdrawn(msg.sender, amount);
    }
```

Figure 4 source code of *withdraw*(1/2)



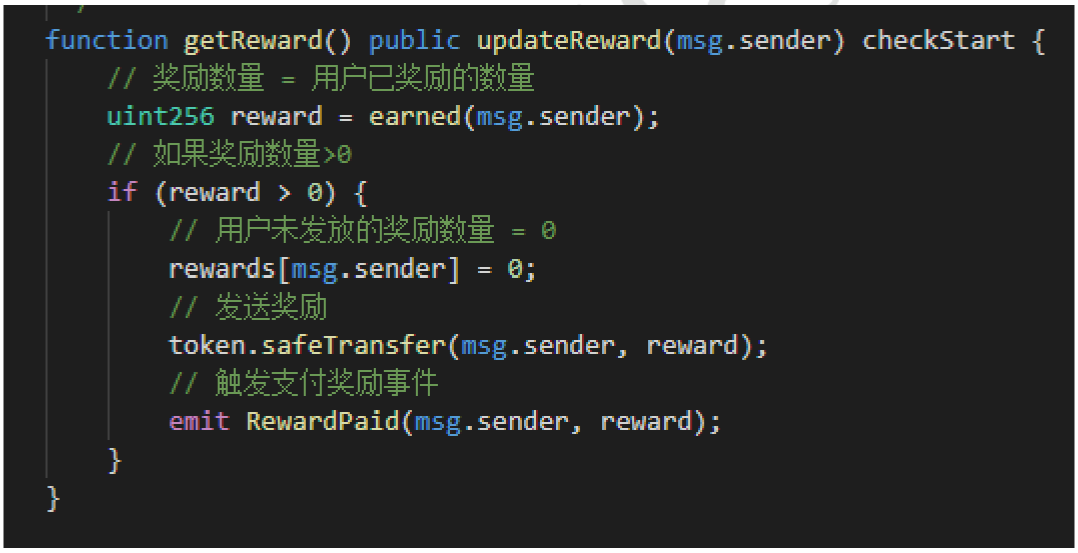
```
function withdraw(uint256 amount) public virtual {
    // 用户的总质押数量
    uint256 directorShare = _balances[msg.sender];
    // 确认总质押数量大于取款数额
    require(directorShare >= amount, 'withdraw request greater than staked amount');
    // 总量减少
    _totalSupply = _totalSupply.sub(amount);
    // 余额减少
    _balances[msg.sender] = _balances[msg.sender].sub(amount);
    // 将LPToken发送给用户
    lpt.safeTransfer(msg.sender, amount);
}
```

Figure 5 source code of *withdraw*(2/2)

- Related functions: *withdraw*, *updateReward*, *safetransfer*
- Result: Pass

(4) Get reward function

- Description: The contract implements the *getReward* function to receive the stake reward. By calling the *safeTransfer* function in the contract, the contract address transfers the specified number of tokens (all stake rewards of the user) to the function caller (user) address; each time the function stake token is called(Check whether the start time is reached through the decorator *checkStart*), the reward-related data is updated through the modifier *updateReward*.



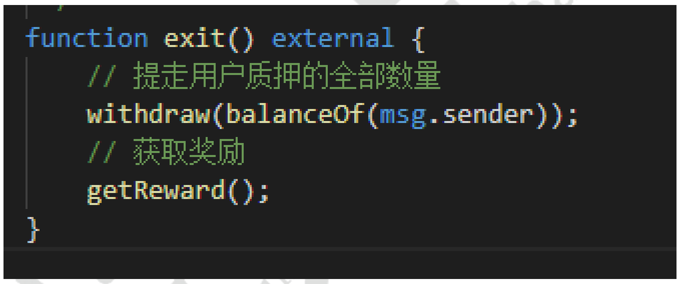
```
function getReward() public updateReward(msg.sender) checkStart {
    // 奖励数量 = 用户已奖励的数量
    uint256 reward = earned(msg.sender);
    // 如果奖励数量>0
    if (reward > 0) {
        // 用户未发放的奖励数量 = 0
        rewards[msg.sender] = 0;
        // 发送奖励
        token.safeTransfer(msg.sender, reward);
        // 触发支付奖励事件
        emit RewardPaid(msg.sender, reward);
    }
}
```

Figure 6 source code of *getReward*

- Related functions: *getReward*, *updateReward*, *safeTransfer*
- Result: Pass

(5) Exit function

- Description: The contract implements the *exit* function for the caller to withdraw from the stake reward participation, calls the *withdraw* function to extract all the staked tokens, calls the *getReward* function to get the caller's stake reward, and ends the "stake-reward" mode participation. At this time, the user address cannot get a new stake reward because the number of staked tokens is empty.



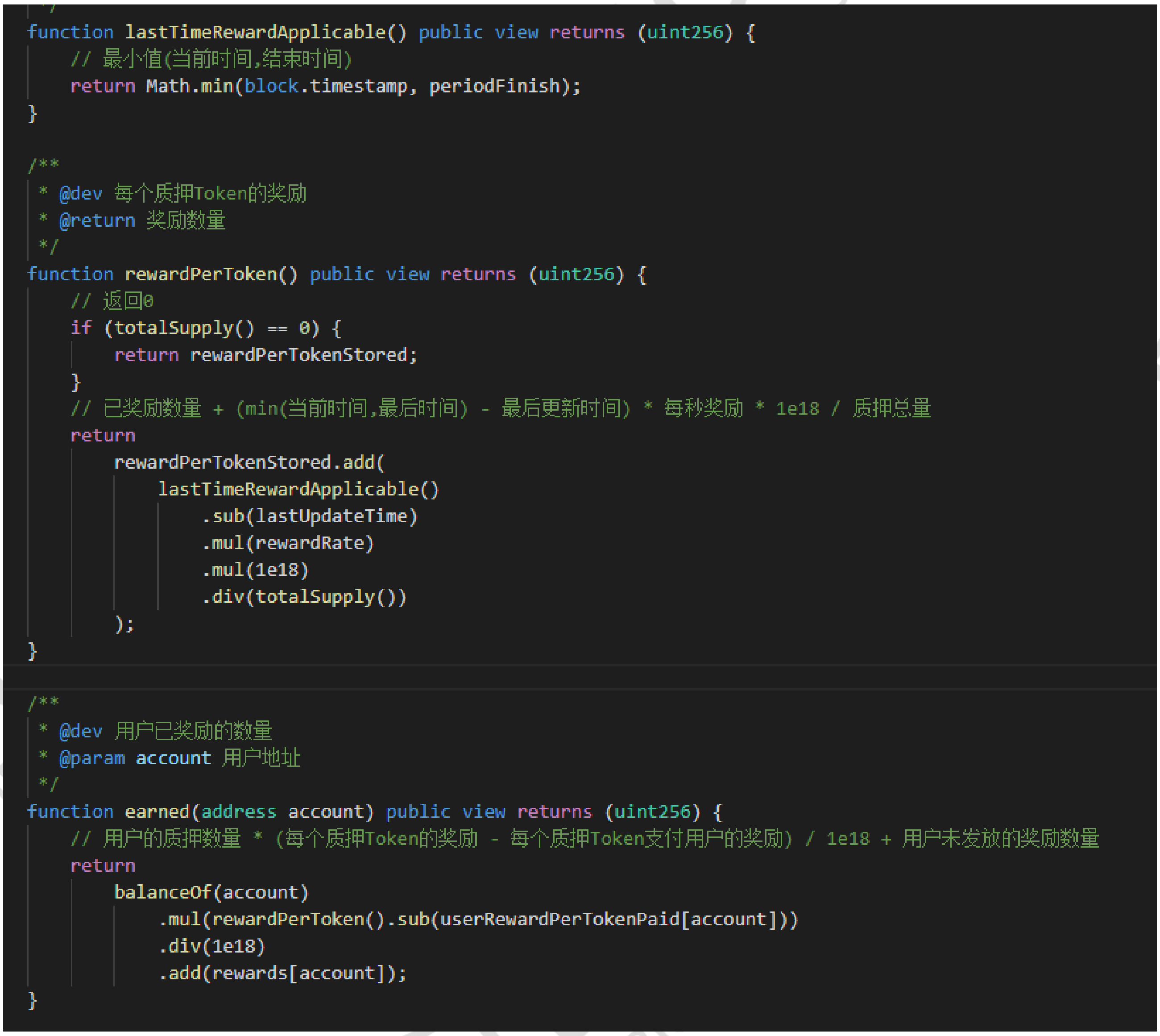
```
function exit() external {
    // 提走用户质押的全部数量
    withdraw(balanceOf(msg.sender));
    // 获取奖励
    getReward();
}
```

Figure 7 source code of *exit*

- Related functions: *exit*, *withdraw*, *getReward*
- Result: Pass

(6) Related parameter query function

- Description: Contract users can query the earliest timestamps in the current timestamp and phase completion time by calling the *lastTimeRewardApplicable* function; call the *rewardPerToken* function to query the stake rewards available for each stake token; and call the *earned* function to query the total stake awards obtained at the specified address.



```
function lastTimeRewardApplicable() public view returns (uint256) {
    // 最小值(当前时间,结束时间)
    return Math.min(block.timestamp, periodFinish);
}

/**
 * @dev 每个质押Token的奖励
 * @return 奖励数量
 */
function rewardPerToken() public view returns (uint256) {
    // 返回0
    if (totalSupply() == 0) {
        return rewardPerTokenStored;
    }
    // 已奖励数量 + (min(当前时间,最后时间) - 最后更新时间) * 每秒奖励 * 1e18 / 质押总量
    return
        rewardPerTokenStored.add(
            lastTimeRewardApplicable()
                .sub(lastUpdateTime)
                .mul(rewardRate)
                .mul(1e18)
                .div(totalSupply())
        );
}

/**
 * @dev 用户已奖励的数量
 * @param account 用户地址
 */
function earned(address account) public view returns (uint256) {
    // 用户的质押数量 * (每个质押Token的奖励 - 每个质押Token支付用户的奖励) / 1e18 + 用户未发放的奖励数量
    return
        balanceOf(account)
            .mul(rewardPerToken().sub(userRewardPerTokenPaid[account]))
            .div(1e18)
            .add(rewards[account]);
}
```

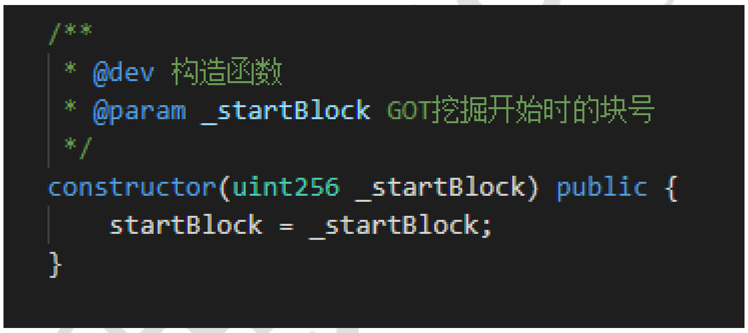
Figure 8 source code of related functions

- Related functions: *lastTimeRewardApplicable*, *rewardPerToken*, *earned*
- Result: Pass

3.2 MasterChef Contract audit

(1) constructor function

- Description: The *constructor* is used to set the block number of the reward token to start mining.



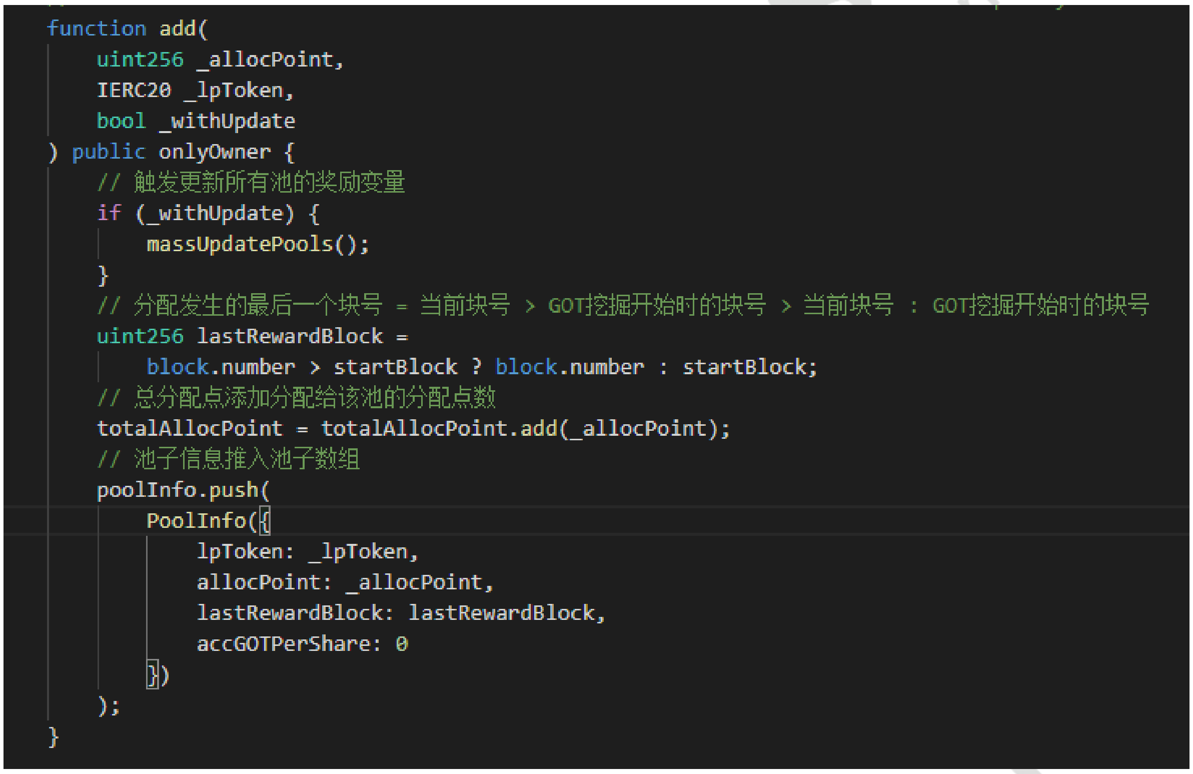
```
/**
 * @dev 构造函数
 * @param _startBlock GOT挖掘开始时的块号
 */
constructor(uint256 _startBlock) public {
    startBlock = _startBlock;
}
```

Figure 9 source code of *constructor*

- Related functions: *constructor*
- Result: Pass

(2) add function

- Description: As shown in the figure below, the contract implements the add function to add the stake pool of LP tokens, and only the owner of the contract can call it. When adding a new stake pool, you need to set the number of allocated points and the contract address of LP tokens. Choose whether to set the *massUpdatePools* function to update all stake pools. It should be noted here that do not repeatedly add the same lptoken as collateral, which will cause errors in the calculation of rewards.



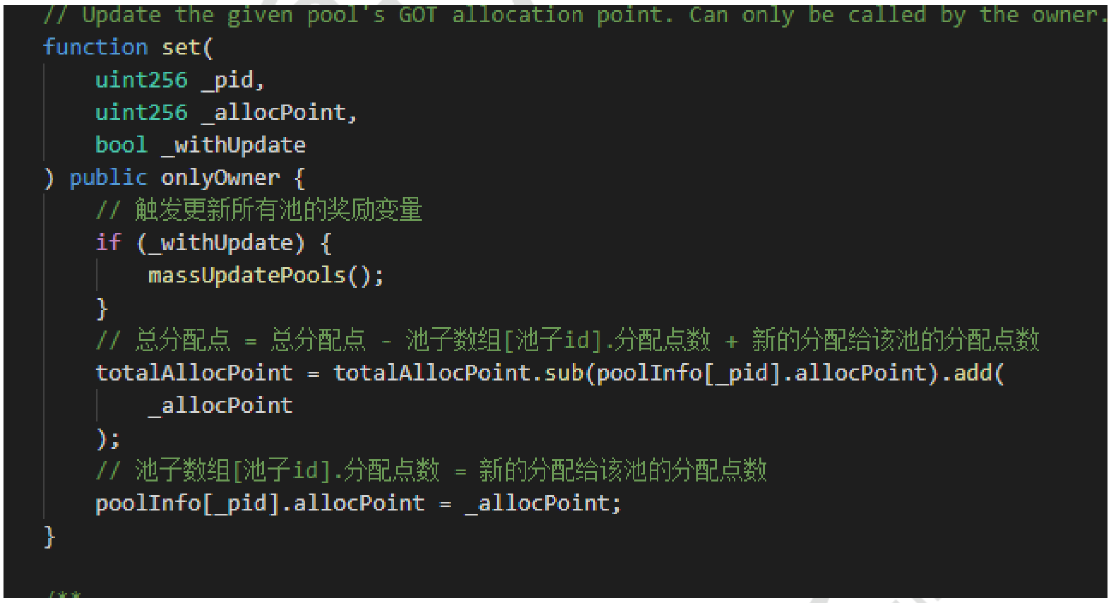
```
function add(
    uint256 _allocPoint,
    IERC20 _lpToken,
    bool _withUpdate
) public onlyOwner {
    // 触发更新所有池的奖励变量
    if (_withUpdate) {
        massUpdatePools();
    }
    // 分配发生的最后一个块号 = 当前块号 > GOT挖掘开始时的块号 > 当前块号 : GOT挖掘开始时的块号
    uint256 lastRewardBlock =
        block.number > startBlock ? block.number : startBlock;
    // 总分配点添加分配给该池的分配点数
    totalAllocPoint = totalAllocPoint.add(_allocPoint);
    // 池子信息推入池子数组
    poolInfo.push(
        PoolInfo({
            lpToken: _lpToken,
            allocPoint: _allocPoint,
            lastRewardBlock: lastRewardBlock,
            accGOTPerShare: 0
        })
    );
}
```

Figure 10 source code of *add*

- Related functions: *add*, *massUpdatePools*, *updatePool*, *balanceOf*, *getMultiplier*, *mint*, *safeApprove*
- Result: Pass

(3) set function

- Description: As shown in the figure below, the contract implements the *set* function to modify the specified stake pool, and only the contract owner can call it. You can modify the allocated points of the stake pool. You can choose whether to execute the *massUpdatePools* function to update all stake pools before modification.



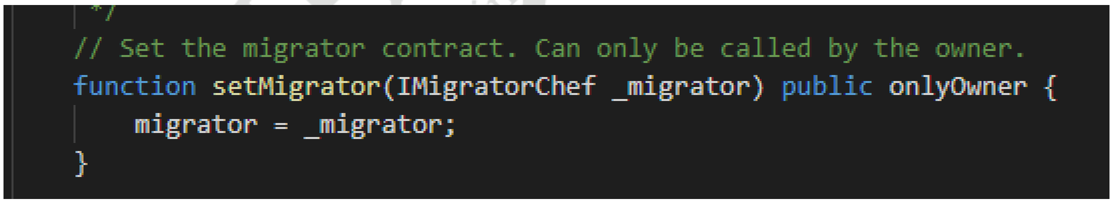
```
// Update the given pool's GOT allocation point. Can only be called by the owner.
function set(
    uint256 _pid,
    uint256 _allocPoint,
    bool _withUpdate
) public onlyOwner {
    // 触发更新所有池的奖励变量
    if (_withUpdate) {
        massUpdatePools();
    }
    // 总分配点 = 总分配点 - 池子数组[池子id].分配点数 + 新的分配给该池的分配点数
    totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(
        _allocPoint
    );
    // 池子数组[池子id].分配点数 = 新的分配给该池的分配点数
    poolInfo[_pid].allocPoint = _allocPoint;
}

/**
```

Figure 11 source code of *set*

- Related functions: set, massUpdatePools, updatePool, balanceOf, getMultiplier, mint, safeApprove
- Result: Pass

(4) setMigrator function

- Description: As shown in the figure below, the contract implements the *setMigrator* function to set the migration address, which can only be called by the contract owner.



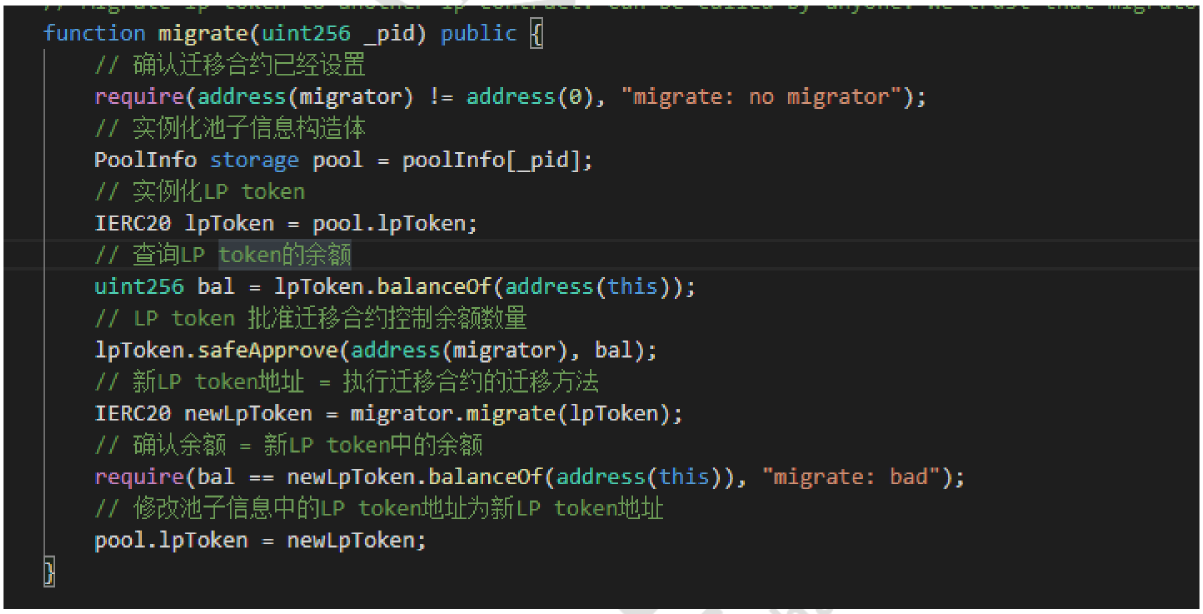
```
    */
    // Set the migrator contract. Can only be called by the owner.
    function setMigrator(IMigratorChef _migrator) public onlyOwner {
        migrator = _migrator;
    }
```

Figure 12 source code of setMigrator

- Related functions: *setMigrator*
- Result: Pass

(5) migrate function

- Description: As shown in the figure below, the contract implements the *migrate* function to migrate all stake lp tokens in the specified pool to the migrator address. The contract owner calls the *setMigrator* function to set the migrator address. Any user can call this function to migrate the specified pool. All staked tokens to the migrator address (requires that the migrator address is not equal to the zero address), the contract will authorize all the staked lp tokens in the specified pool to the migrator address, and call the *migrate* function of the migrator contract to migrate all the staked tokens (default migrator is Zero address). (Note: If migrator is a malicious contract address, there is a risk of losing the user's stake tokens.)



```
// Migrate lp token to another lp contract. Can be called by anyone. We trust that migrato
function migrate(uint256 _pid) public {
    // 确认迁移合约已经设置
    require(address(migrator) != address(0), "migrate: no migrator");
    // 实例化池子信息构造体
    PoolInfo storage pool = poolInfo[_pid];
    // 实例化LP token
    IERC20 lpToken = pool.lpToken;
    // 查询LP token的余额
    uint256 bal = lpToken.balanceOf(address(this));
    // LP token 批准迁移合约控制余额数量
    lpToken.safeApprove(address(migrator), bal);
    // 新LP token地址 = 执行迁移合约的迁移方法
    IERC20 newLpToken = migrator.migrate(lpToken);
    // 确认余额 = 新LP token中的余额
    require(bal == newLpToken.balanceOf(address(this)), "migrate: bad");
    // 修改池子信息中的LP token地址为新LP token地址
    pool.lpToken = newLpToken;
}
```

Figure 13 source code of *migrate*

- Related functions: *migrate, safeApprove, balanceOf*

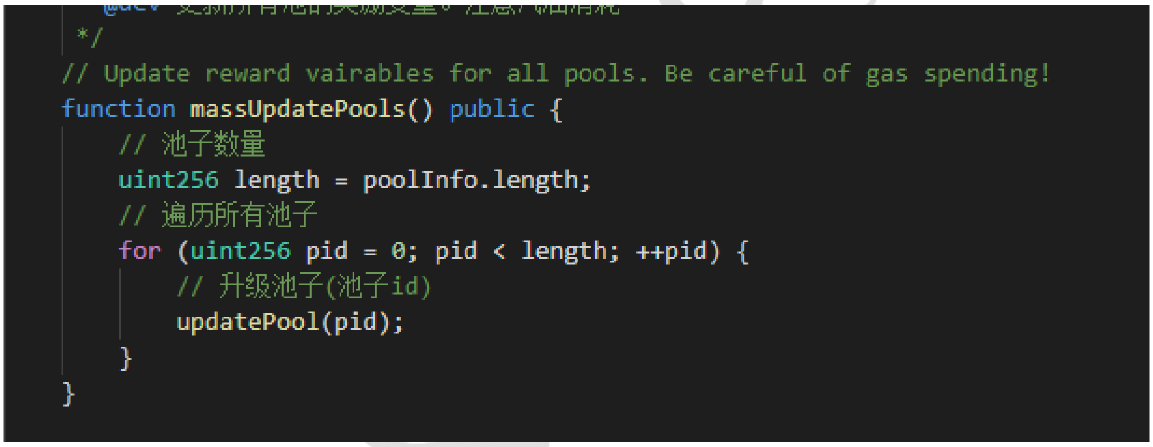
- Safety advice: It is recommended to delete this function.

- Modification results: Ignored (the project party declares: The value of the migrator variable can only be set by the administrator. The purpose of setting this is similar to the purpose of sushiswap. The corresponding assets of lptoken from other exchanges are taken out, and then deposited in our own exchange. For the purpose of generating our own lptoken, to accomplish this, we must first set the migrator variable. The corresponding contract can only be designed by ourselves, because this method has a relatively large authority, so we can give the authority to set the migrator to time Lock contract or multi-sign contract)

- Result: Pass

(6) massUpdatePools function

- Description: As shown in the figure below, the contract implements the *massUpdatePools* function to update all stake pool information by traversing all stake pools and calling the *updatePool* function.



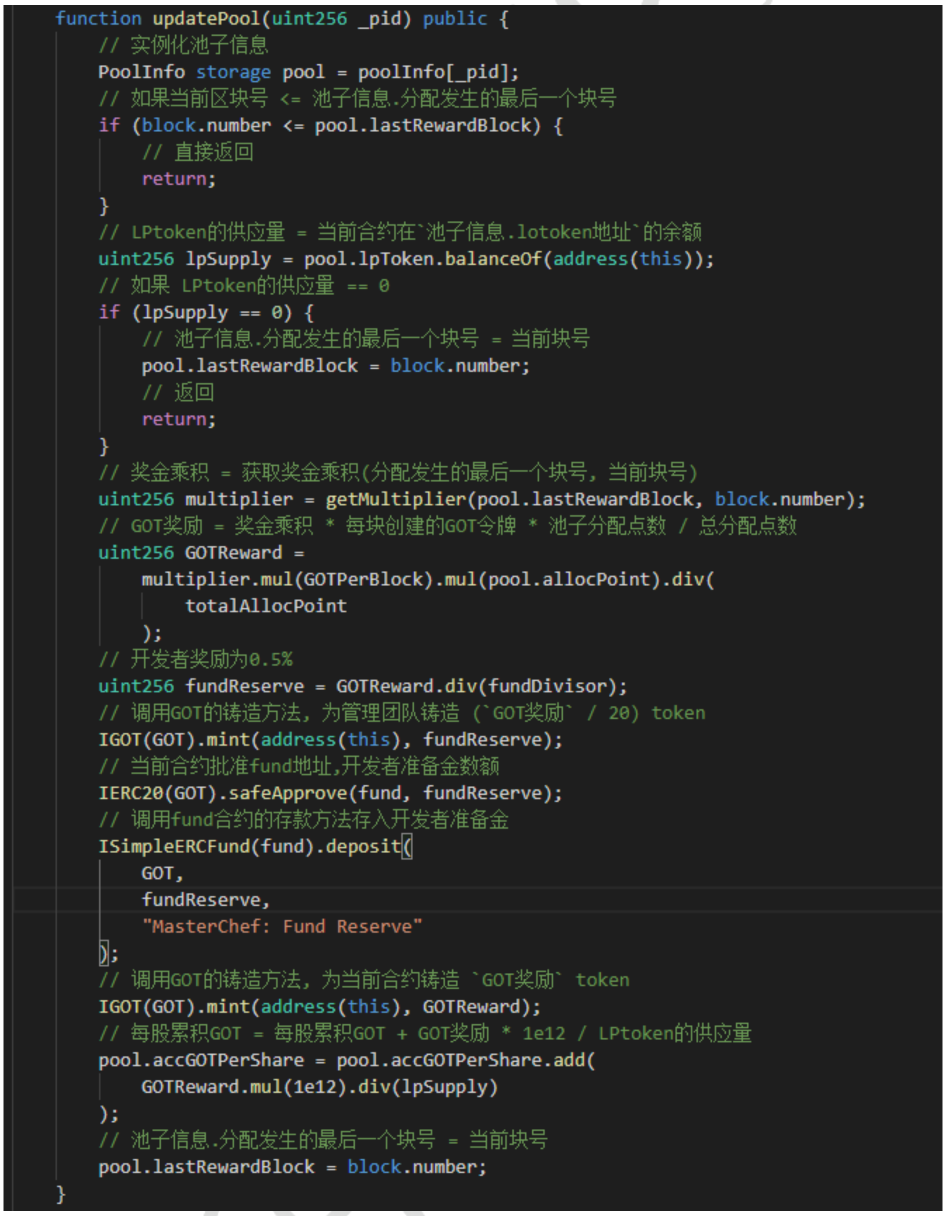
```
 */
// Update reward vairables for all pools. Be careful of gas spending!
function massUpdatePools() public {
    // 池子数量
    uint256 length = poolInfo.length;
    // 遍历所有池子
    for (uint256 pid = 0; pid < length; ++pid) {
        // 升级池子(池子id)
        updatePool(pid);
    }
}
```

Figure 14 source code of *massUpdatePools*

- Related functions: *updatePool*
- Result: Pass

(7) updatePool function

- Description: As shown in the figure below, the contract implements the *updatePool* function to update the specified stake pool information. Calling this function requires that the current block number is greater than the last reward block number, and the LP token balance of the designated stake pool in the contract is greater than 0. Calculate the GOT rewards obtained from the last block to the current block, and mint the rewards to this contract address (requires this contract address to be the GOT token contract owner), where the GOT contract will mint an additional 5% of the coins to this contract address, and authorize it to The authorization value corresponding to the fund address is deposited in the developer reserve after calling the *deposit* function of the fund contract, and finally the relevant parameters of the stake pool are updated.



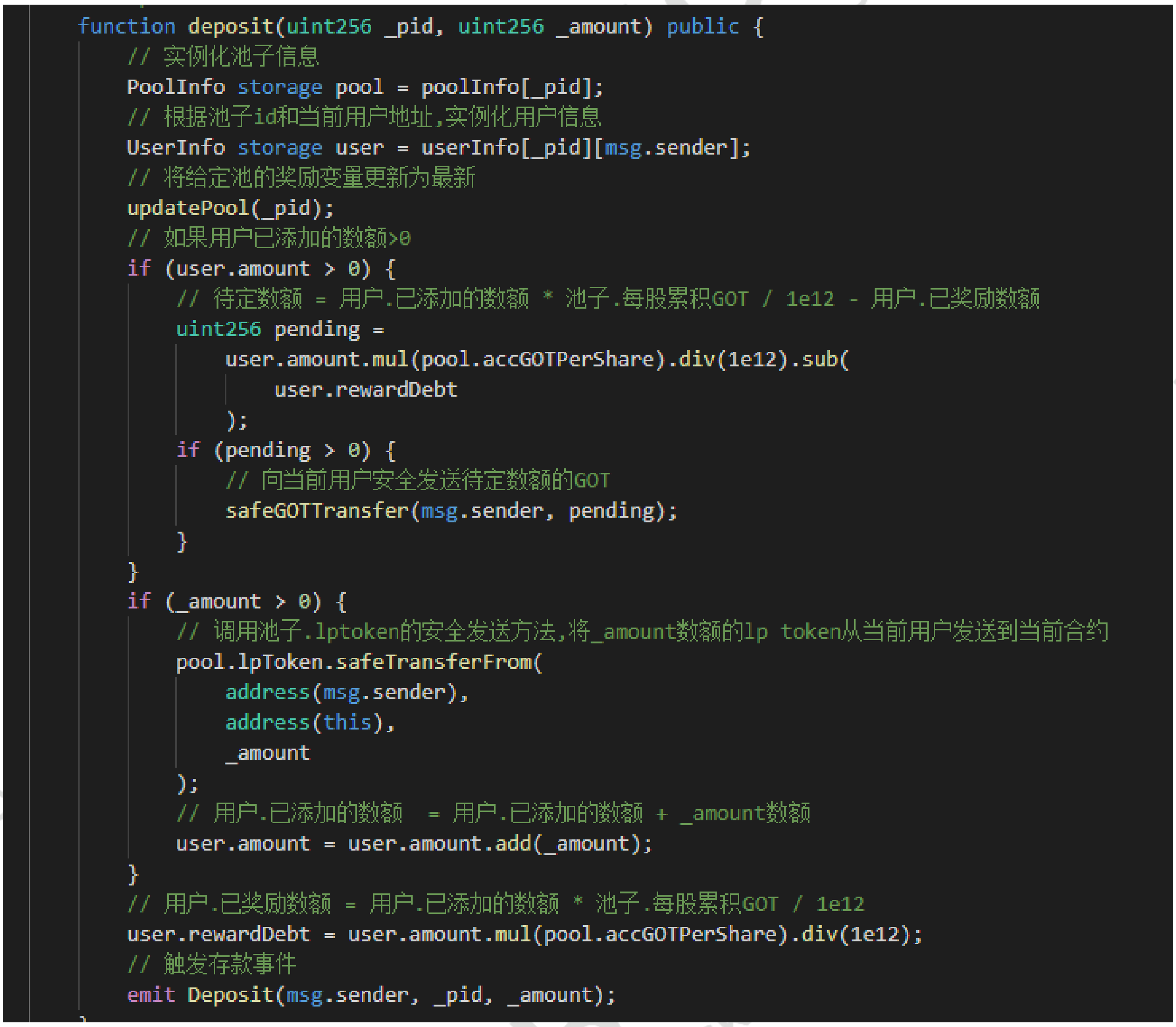
```
function updatePool(uint256 _pid) public {
    // 实例化池子信息
    PoolInfo storage pool = poolInfo[_pid];
    // 如果当前区块号 <= 池子信息.分配发生的最后一个块号
    if (block.number <= pool.lastRewardBlock) {
        // 直接返回
        return;
    }
    // LPtoken的供应量 = 当前合约在`池子信息.lotoken地址`的余额
    uint256 lpSupply = pool.lpToken.balanceOf(address(this));
    // 如果 LPtoken的供应量 == 0
    if (lpSupply == 0) {
        // 池子信息.分配发生的最后一个块号 = 当前块号
        pool.lastRewardBlock = block.number;
        // 返回
        return;
    }
    // 奖金乘积 = 获取奖金乘积(分配发生的最后一个块号, 当前块号)
    uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
    // GOT奖励 = 奖金乘积 * 每块创建的GOT令牌 * 池子分配点数 / 总分配点数
    uint256 GOTReward =
        multiplier.mul(GOTPerBlock).mul(pool.allocPoint).div(
            totalAllocPoint
        );
    // 开发者奖励为0.5%
    uint256 fundReserve = GOTReward.div(fundDivisor);
    // 调用GOT的铸造方法, 为管理团队铸造 (`GOT奖励` / 20) token
    IGOT(GOT).mint(address(this), fundReserve);
    // 当前合约批准fund地址,开发者准备金数额
    IERC20(GOT).safeApprove(fund, fundReserve);
    // 调用fund合约的存款方法存入开发者准备金
    ISimpleERCFund(fund).deposit(
        GOT,
        fundReserve,
        "MasterChef: Fund Reserve"
    );
    // 调用GOT的铸造方法, 为当前合约铸造 `GOT奖励` token
    IGOT(GOT).mint(address(this), GOTReward);
    // 每股累积GOT = 每股累积GOT + GOT奖励 * 1e12 / LPtoken的供应量
    pool.accGOTPerShare = pool.accGOTPerShare.add(
        GOTReward.mul(1e12).div(lpSupply)
    );
    // 池子信息.分配发生的最后一个块号 = 当前块号
    pool.lastRewardBlock = block.number;
}
```

Figure 15 source code of *updatePool*

- Related functions: *updatePool*, *balanceOf*, *getMultiplier*, *mint*, *safeApprove*
- Result: Pass

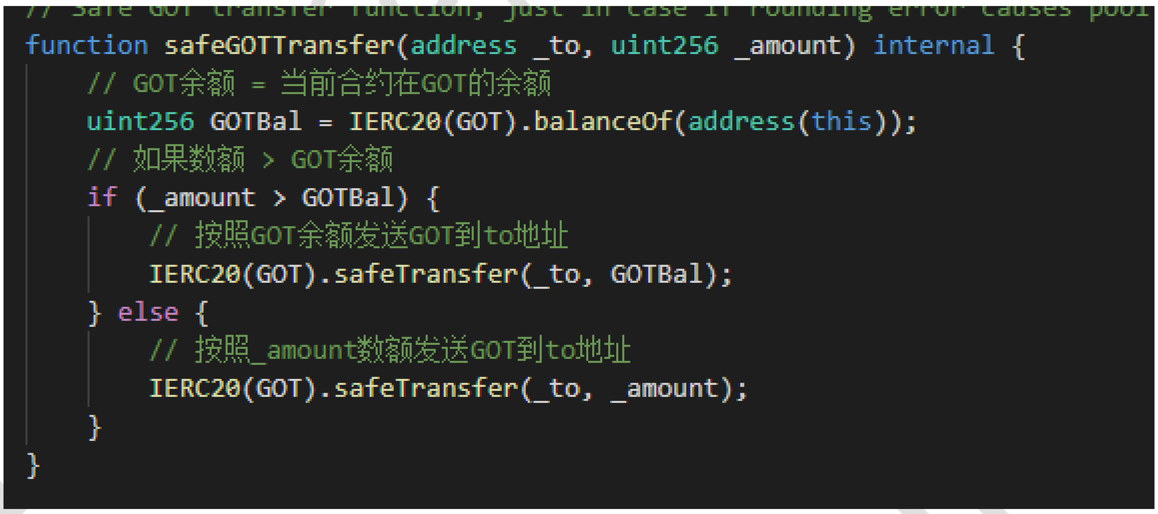
(8) deposit function

- Description: As shown in the figure below, the contract implements the *deposit* function for users to pledge LP tokens to obtain rewards. The user stake tokens to the designated stake pool through this function, and updates the stake pool information. If the user has previously staked in the current stake pool, the reward will be calculated and received. After the tokens are staked, the relevant parameters of the user are updated.



```
function deposit(uint256 _pid, uint256 _amount) public {
    // 实例化池子信息
    PoolInfo storage pool = poolInfo[_pid];
    // 根据池子id和当前用户地址,实例化用户信息
    UserInfo storage user = userInfo[_pid][msg.sender];
    // 将给定池的奖励变量更新为最新
    updatePool(_pid);
    // 如果用户已添加的数额>0
    if (user.amount > 0) {
        // 待定数额 = 用户.已添加的数额 * 池子.每股累积GOT / 1e12 - 用户.已奖励数额
        uint256 pending =
            user.amount.mul(pool.accGOTPerShare).div(1e12).sub(
                user.rewardDebt
            );
        if (pending > 0) {
            // 向当前用户安全发送待定数额的GOT
            safeGOTTransfer(msg.sender, pending);
        }
    }
    if (_amount > 0) {
        // 调用池子.lptoken的安全发送方法,将_amount数额的lp token从当前用户发送到当前合约
        pool.lpToken.safeTransferFrom(
            address(msg.sender),
            address(this),
            _amount
        );
        // 用户.已添加的数额  = 用户.已添加的数额 + _amount数额
        user.amount = user.amount.add(_amount);
    }
    // 用户.已奖励数额 = 用户.已添加的数额 * 池子.每股累积GOT / 1e12
    user.rewardDebt = user.amount.mul(pool.accGOTPerShare).div(1e12);
    // 触发存款事件
    emit Deposit(msg.sender, _pid, _amount);
```

Figure 16 source code of *deposit*



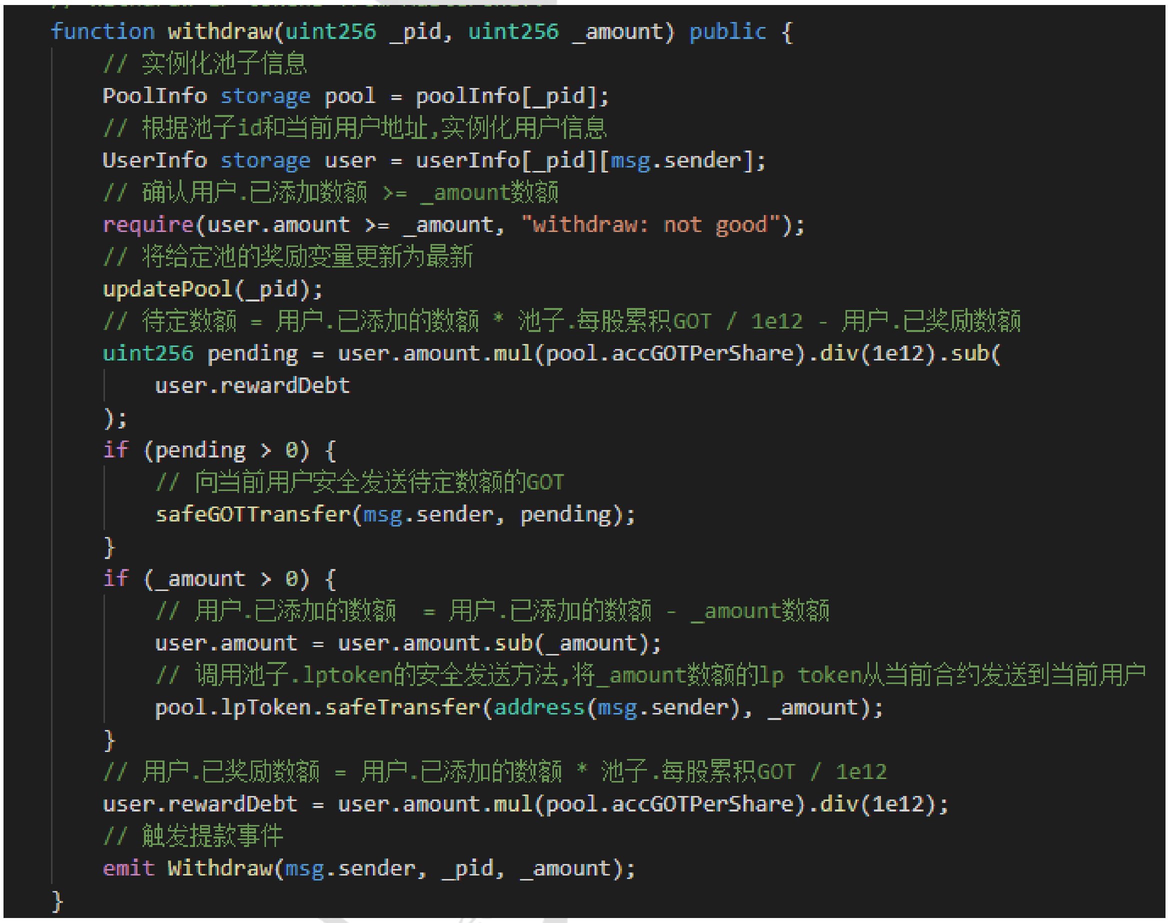
```
// Safe GOT transfer function, just in case if rounding error causes pool
function safeGOTTransfer(address _to, uint256 _amount) internal {
    // GOT余额 = 当前合约在GOT的余额
    uint256 GOTBal = IERC20(GOT).balanceOf(address(this));
    // 如果数额 > GOT余额
    if (_amount > GOTBal) {
        // 按照GOT余额发送GOT到to地址
        IERC20(GOT).safeTransfer(_to, GOTBal);
    } else {
        // 按照_amount数额发送GOT到to地址
        IERC20(GOT).safeTransfer(_to, _amount);
    }
}
```

Figure 17 source code of *safeGOTTransfer*

- Related functions: *updatePool*, *balanceOf*, *getMultiplier*, *mint*, *safeApprove*, *safeGOTTransfer*, *safeTransferFrom*
- Result: Pass

(9) withdraw function

- Description: As shown in the figure below, the contract implements the *withdraw* function to enable users to withdraw the staked LP tokens from the designated stake pool. When the user calls this function, the current stake pool information will be updated first, the rewards should be calculated and received, and the relevant parameters of the user will be updated after the tokens are withdrawn.



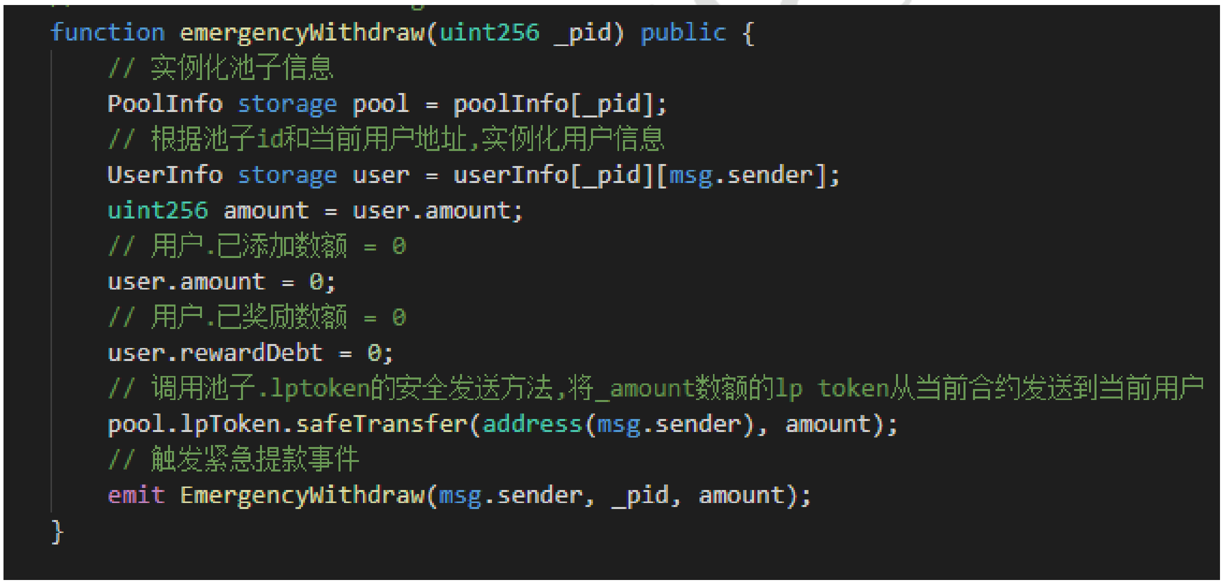
```
function withdraw(uint256 _pid, uint256 _amount) public {
    // 实例化池子信息
    PoolInfo storage pool = poolInfo[_pid];
    // 根据池子id和当前用户地址,实例化用户信息
    UserInfo storage user = userInfo[_pid][msg.sender];
    // 确认用户.已添加数额 >= _amount数额
    require(user.amount >= _amount, "withdraw: not good");
    // 将给定池的奖励变量更新为最新
    updatePool(_pid);
    // 待定数额 = 用户.已添加的数额 * 池子.每股累积GOT / 1e12 - 用户.已奖励数额
    uint256 pending = user.amount.mul(pool.accGOTPerShare).div(1e12).sub(
        user.rewardDebt
    );
    if (pending > 0) {
        // 向当前用户安全发送待定数额的GOT
        safeGOTTransfer(msg.sender, pending);
    }
    if (_amount > 0) {
        // 用户.已添加的数额  = 用户.已添加的数额 - _amount数额
        user.amount = user.amount.sub(_amount);
        // 调用池子.lptoken的安全发送方法,将_amount数额的lp token从当前合约发送到当前用户
        pool.lpToken.safeTransfer(address(msg.sender), _amount);
    }
    // 用户.已奖励数额 = 用户.已添加的数额 * 池子.每股累积GOT / 1e12
    user.rewardDebt = user.amount.mul(pool.accGOTPerShare).div(1e12);
    // 触发提款事件
    emit Withdraw(msg.sender, _pid, _amount);
}
```

Figure 18 source code of *withdraw*

- Related functions: *updatePool*, *balanceOf*, *getMultiplier*, *mint*, *safeApprove*, *safeGOTTransfer*, *safeTransfer*
- Result: Pass

(10) *emergencyWithdraw* function

- Description: As shown in the figure below, the contract implements the *emergencyWithdraw* function for emergency withdrawal and exit. Users calling this function will withdraw all LP tokens staked in the designated stake pool, and there is no reward calculation.



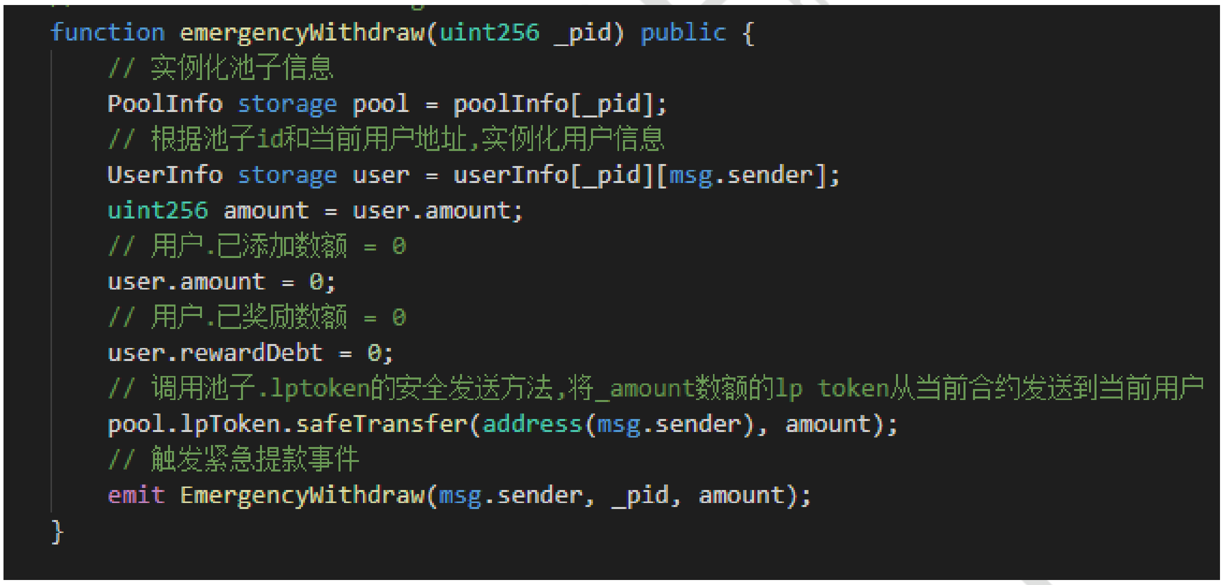
```
function emergencyWithdraw(uint256 _pid) public {
    // 实例化池子信息
    PoolInfo storage pool = poolInfo[_pid];
    // 根据池子id和当前用户地址,实例化用户信息
    UserInfo storage user = userInfo[_pid][msg.sender];
    uint256 amount = user.amount;
    // 用户.已添加数额 = 0
    user.amount = 0;
    // 用户.已奖励数额 = 0
    user.rewardDebt = 0;
    // 调用池子.lptoken的安全发送方法,将_amount数额的lp token从当前合约发送到当前用户
    pool.lpToken.safeTransfer(address(msg.sender), amount);
    // 触发紧急提款事件
    emit EmergencyWithdraw(msg.sender, _pid, amount);
}
```

Figure 19 source code of *emergencyWithdraw*

- Related functions: *emergencyWithdraw*
- Result: Pass

(11) *harvest* function

- Description: As shown in the figure below, the contract implements the *harvest* function for receiving rewards from the designated stake pool. Users calling this function will receive reward tokens in the designated stake pool. User-related parameters will be updated after receiving the reward.



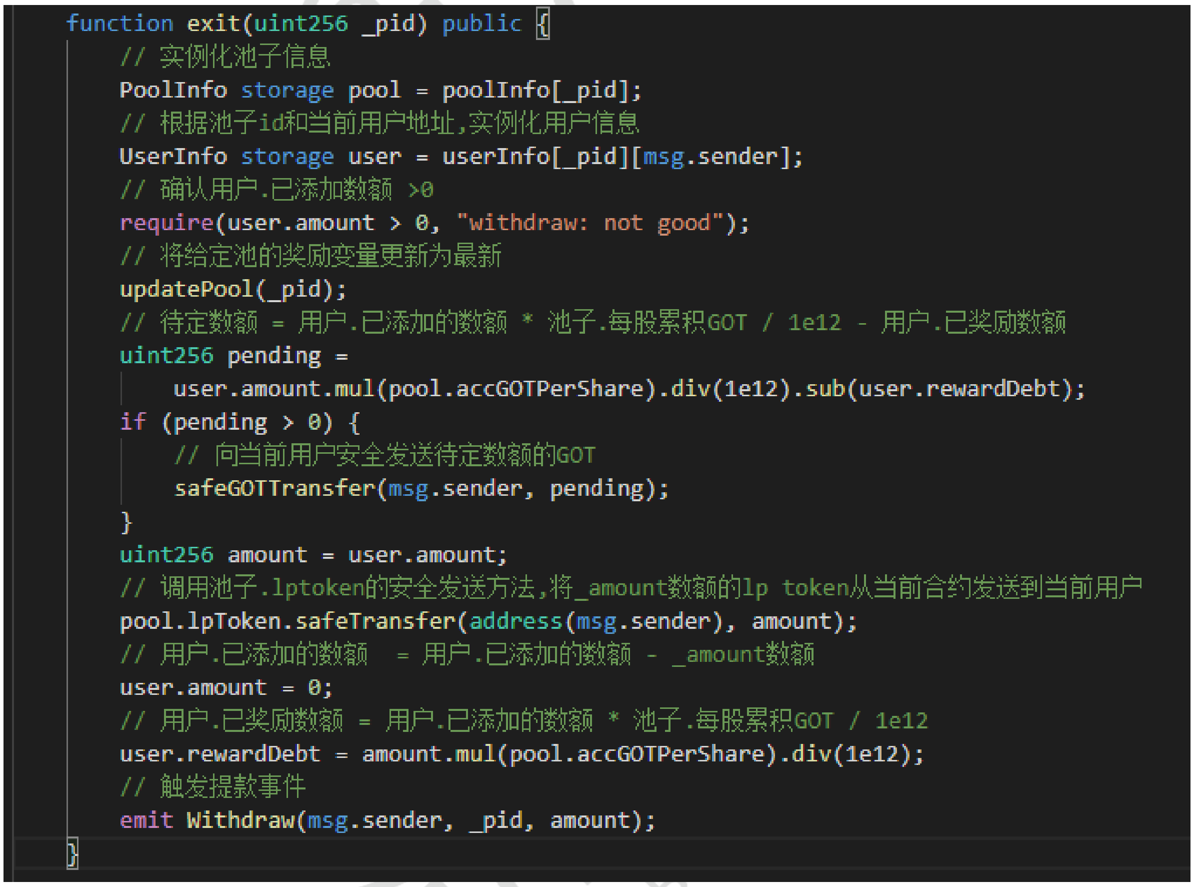
```
function emergencyWithdraw(uint256 _pid) public {
    // 实例化池子信息
    PoolInfo storage pool = poolInfo[_pid];
    // 根据池子id和当前用户地址,实例化用户信息
    UserInfo storage user = userInfo[_pid][msg.sender];
    uint256 amount = user.amount;
    // 用户.已添加数额 = 0
    user.amount = 0;
    // 用户.已奖励数额 = 0
    user.rewardDebt = 0;
    // 调用池子.lptoken的安全发送方法,将_amount数额的lp token从当前合约发送到当前用户
    pool.lpToken.safeTransfer(address(msg.sender), amount);
    // 触发紧急提款事件
    emit EmergencyWithdraw(msg.sender, _pid, amount);
}
```

Figure 20 source code of *harvest*

- Related functions: *harvest*
- Result: Pass

(12) *exit* function

- Description: As shown in the figure below, the contract implements an *exit* function to enable users to withdraw all LP tokens that have been staked in the designated stake pool. When the user calls this function, the current stake pool information will be updated first, the rewards should be calculated and received, and the relevant parameters of the user will be updated after the tokens are withdrawn.



```
function exit(uint256 _pid) public {
    // 实例化池子信息
    PoolInfo storage pool = poolInfo[_pid];
    // 根据池子id和当前用户地址,实例化用户信息
    UserInfo storage user = userInfo[_pid][msg.sender];
    // 确认用户.已添加数额 >0
    require(user.amount > 0, "withdraw: not good");
    // 将给定池的奖励变量更新为最新
    updatePool(_pid);
    // 待定数额 = 用户.已添加的数额 * 池子.每股累积GOT / 1e12 - 用户.已奖励数额
    uint256 pending =
        user.amount.mul(pool.accGOTPerShare).div(1e12).sub(user.rewardDebt);
    if (pending > 0) {
        // 向当前用户安全发送待定数额的GOT
        safeGOTTransfer(msg.sender, pending);
    }
    uint256 amount = user.amount;
    // 调用池子.lptoken的安全发送方法,将_amount数额的lp token从当前合约发送到当前用户
    pool.lpToken.safeTransfer(address(msg.sender), amount);
    // 用户.已添加的数额  = 用户.已添加的数额 - _amount数额
    user.amount = 0;
    // 用户.已奖励数额 = 用户.已添加的数额 * 池子.每股累积GOT / 1e12
    user.rewardDebt = amount.mul(pool.accGOTPerShare).div(1e12);
    // 触发提款事件
    emit Withdraw(msg.sender, _pid, amount);
}
```

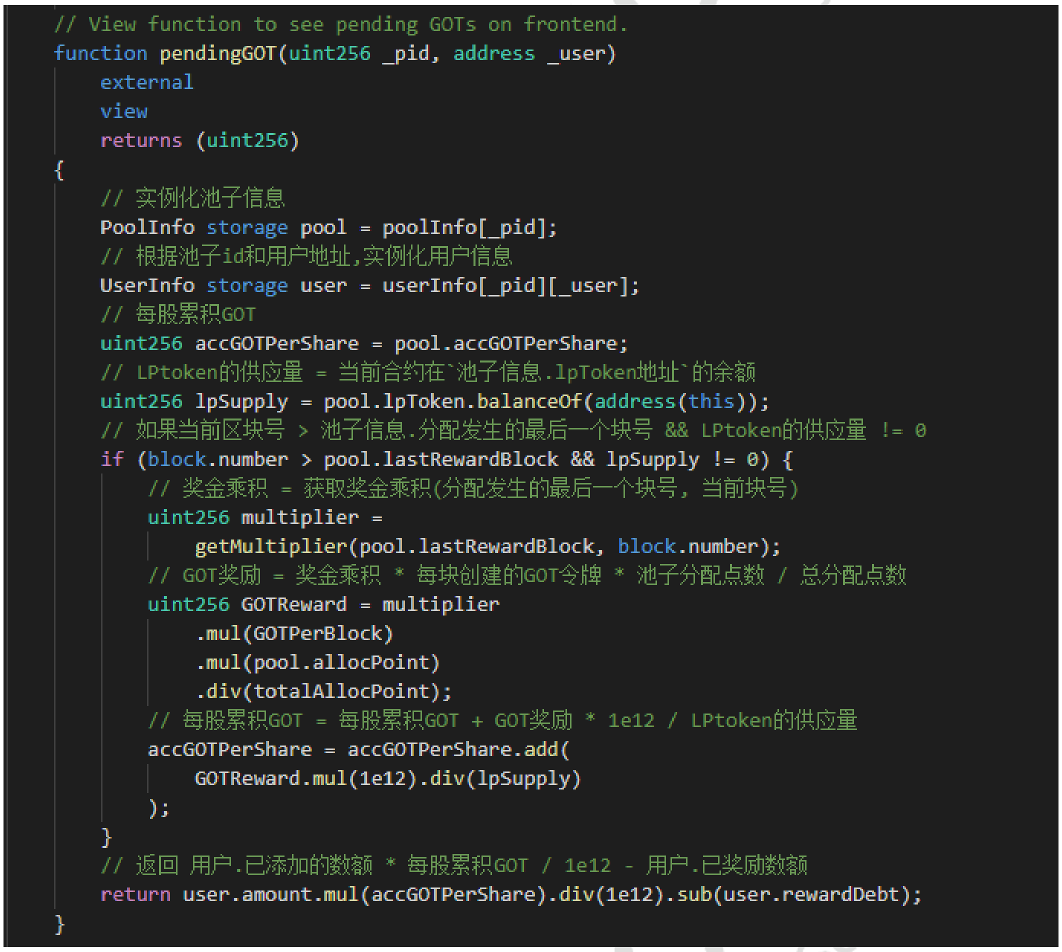
Figure 21 source code of *exit*

- Related functions: *updatePool*, *balanceOf*, *getMultiplier*, *mint*,*safeApprove*, *safeGOTTransfer*, *safeTransfer*

- Result: Pass

(13) *pendingGOT* function

- Description: As shown in the figure below, the contract implements the *pendingGOT* function to enable users to query the unclaimed GOT tokens in the designated stake pool.



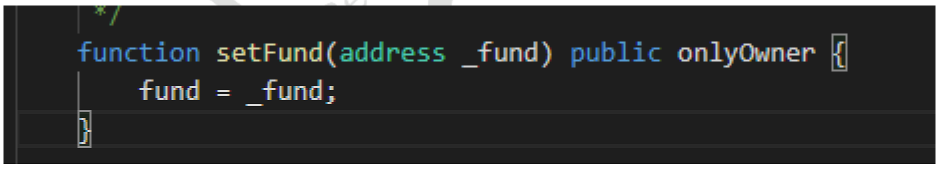
```
// View function to see pending GOTs on frontend.
function pendingGOT(uint256 _pid, address _user)
    external
    view
    returns (uint256)
{
    // 实例化池子信息
    PoolInfo storage pool = poolInfo[_pid];
    // 根据池子id和用户地址,实例化用户信息
    UserInfo storage user = userInfo[_pid][_user];
    // 每股累积GOT
    uint256 accGOTPerShare = pool.accGOTPerShare;
    // LPtoken的供应量 = 当前合约在`池子信息.lpToken地址`的余额
    uint256 lpSupply = pool.lpToken.balanceOf(address(this));
    // 如果当前区块号 > 池子信息.分配发生的最后一个块号 && LPtoken的供应量 != 0
    if (block.number > pool.lastRewardBlock && lpSupply != 0) {
        // 奖金乘积 = 获取奖金乘积(分配发生的最后一个块号, 当前块号)
        uint256 multiplier =
            getMultiplier(pool.lastRewardBlock, block.number);
        // GOT奖励 = 奖金乘积 * 每块创建的GOT令牌 * 池子分配点数 / 总分配点数
        uint256 GOTReward = multiplier
            .mul(GOTPerBlock)
            .mul(pool.allocPoint)
            .div(totalAllocPoint);
        // 每股累积GOT = 每股累积GOT + GOT奖励 * 1e12 / LPtoken的供应量
        accGOTPerShare = accGOTPerShare.add(
            GOTReward.mul(1e12).div(lpSupply)
        );
    }
    // 返回 用户.已添加的数额 * 每股累积GOT / 1e12 - 用户.已奖励数额
    return user.amount.mul(accGOTPerShare).div(1e12).sub(user.rewardDebt);
}
```

Figure 22 source code of *pendingGOT*

- Related functions: *balanceOf*, *getMultiplier*
- Result: Pass

(14) *setFund* function

- Description: As shown in the figure below, the contract implements the *setFund* function to modify the fund address, which can only be called by the owner.



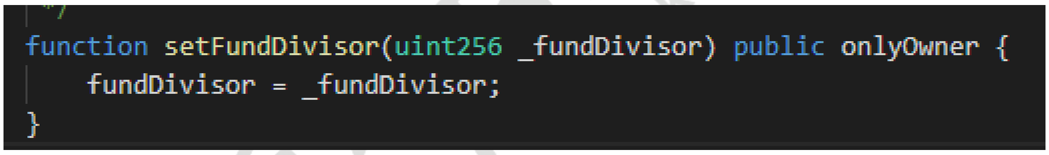
```
    */
    function setFund(address _fund) public onlyOwner {
        fund = _fund;
    }
```

Figure 23 source code of *setFund*

- Related functions: *setFund*
- Result: Pass

(15) *setFundDivisor* function

- Description: As shown in the figure below, the contract implements the *setFundDivisor* function to modify the developer reward fund ratio (the default value is 20), and only the owner can call it.



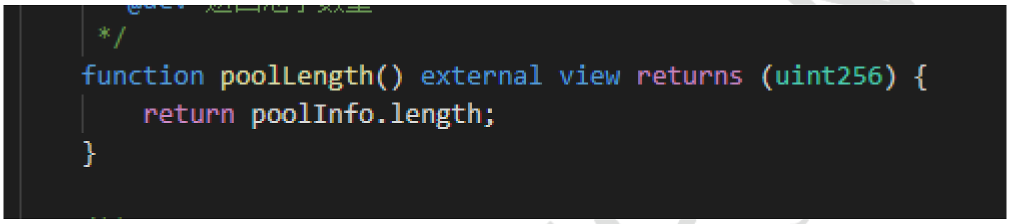
```
function setFundDivisor(uint256 _fundDivisor) public onlyOwner {
    fundDivisor = _fundDivisor;
}
```

Figure 24 source code of *setFundDivisor*

- Related functions: *setFundDivisor*
- Result: Pass

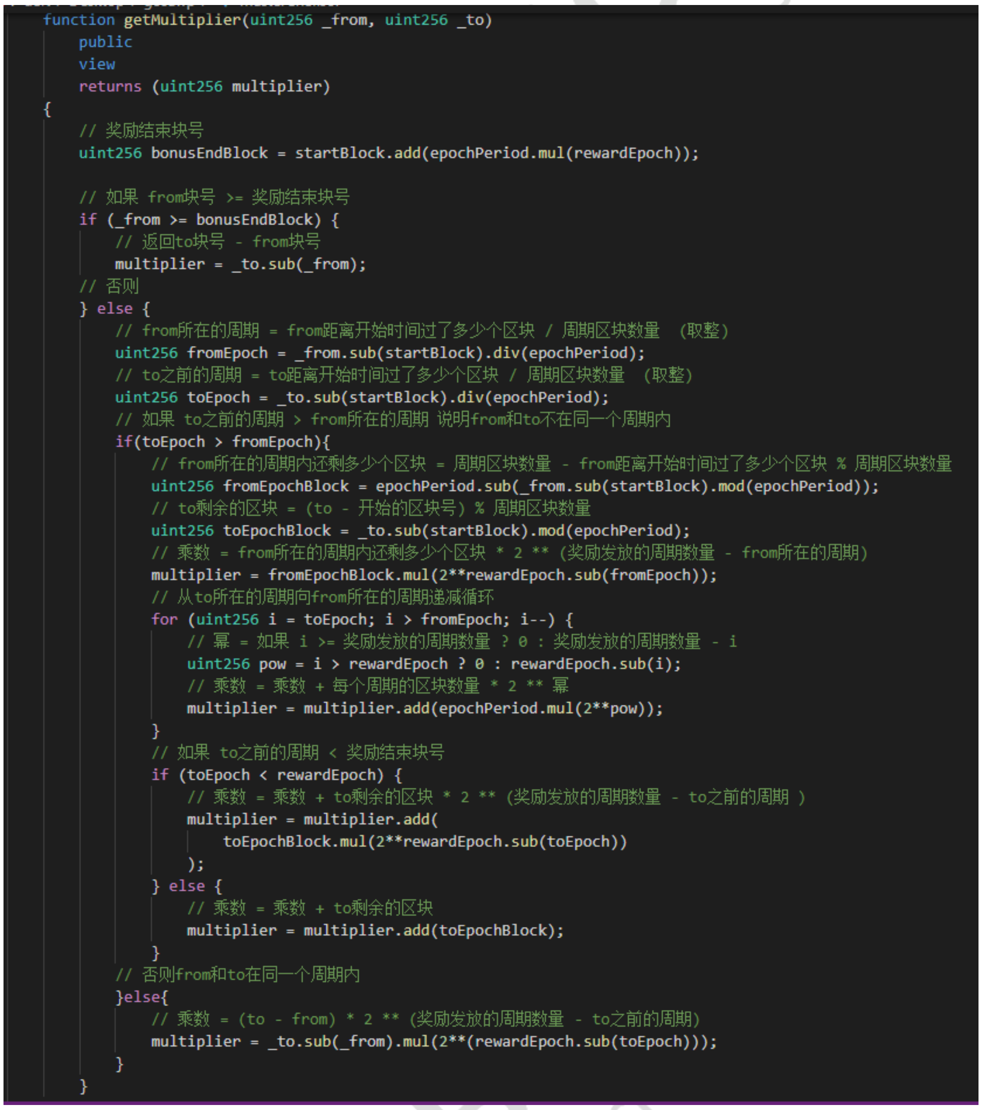
(16) Other related query functions

- Description: As shown in the figure below, the contract implements the *poolLength* function to query the number of stake pools, and the *getMultiplier* function to query the parameters of the reward calculation. There are 12 cycles here, and the reward will decrease as the block increases.



```
 */
function poolLength() external view returns (uint256) {
    return poolInfo.length;
}
```

Figure 25 source code of *poolLength*



```
function getMultiplier(uint256 _from, uint256 _to)
    public
    view
    returns (uint256 multiplier)
{
    // 奖励结束块号
    uint256 bonusEndBlock = startBlock.add(epochPeriod.mul(rewardEpoch));

    // 如果 from块号 >= 奖励结束块号
    if (_from >= bonusEndBlock) {
        // 返回to块号 - from块号
        multiplier = _to.sub(_from);
    // 否则
    } else {
        // from所在的周期 = from距离开始时间过了多少个区块 / 周期区块数量  (取整)
        uint256 fromEpoch = _from.sub(startBlock).div(epochPeriod);
        // to之前的周期 = to距离开始时间过了多少个区块 / 周期区块数量  (取整)
        uint256 toEpoch = _to.sub(startBlock).div(epochPeriod);
        // 如果 to之前的周期 > from所在的周期 说明from和to不在同一个周期内
        if(toEpoch > fromEpoch){
            // from所在的周期内还剩多少个区块 = 周期区块数量 - from距离开始时间过了多少个区块 % 周期区块数量
            uint256 fromEpochBlock = epochPeriod.sub(_from.sub(startBlock).mod(epochPeriod));
            // to剩余的区块 = (to - 开始的区块号) % 周期区块数量
            uint256 toEpochBlock = _to.sub(startBlock).mod(epochPeriod);
            // 乘数 = from所在的周期内还剩多少个区块 * 2 ** (奖励发放的周期数量 - from所在的周期)
            multiplier = fromEpochBlock.mul(2**rewardEpoch.sub(fromEpoch));
            // 从to所在的周期向from所在的周期递减循环
            for (uint256 i = toEpoch; i > fromEpoch; i--) {
                // 幂 = 如果 i >= 奖励发放的周期数量 ? 0 : 奖励发放的周期数量 - i
                uint256 pow = i > rewardEpoch ? 0 : rewardEpoch.sub(i);
                // 乘数 = 乘数 + 每个周期的区块数量 * 2 ** 幂
                multiplier = multiplier.add(epochPeriod.mul(2**pow));
            }
            // 如果 to之前的周期 < 奖励结束块号
            if (toEpoch < rewardEpoch) {
                // 乘数 = 乘数 + to剩余的区块 * 2 ** (奖励发放的周期数量 - to之前的周期 )
                multiplier = multiplier.add(
                    toEpochBlock.mul(2**rewardEpoch.sub(toEpoch))
                );
            } else {
                // 乘数 = 乘数 + to剩余的区块
                multiplier = multiplier.add(toEpochBlock);
            }
        // 否则from和to在同一个周期内
        }else{
            // 乘数 = (to - from) * 2 ** (奖励发放的周期数量 - to之前的周期)
            multiplier = _to.sub(_from).mul(2**(rewardEpoch.sub(toEpoch)));
        }
    }
```

Figure 26 source code of *getMultiplier*

- Related functions: *poolLength*, *getMultiplier*
- Result: Pass

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts project Goswap. The problems found by the audit team during the audit process have been notified to the project party and reached an agreement on the repair results, the overall audit result of the Goswap project's smart contract is **Pass**.

BEOSIN
Blockchain Security

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com