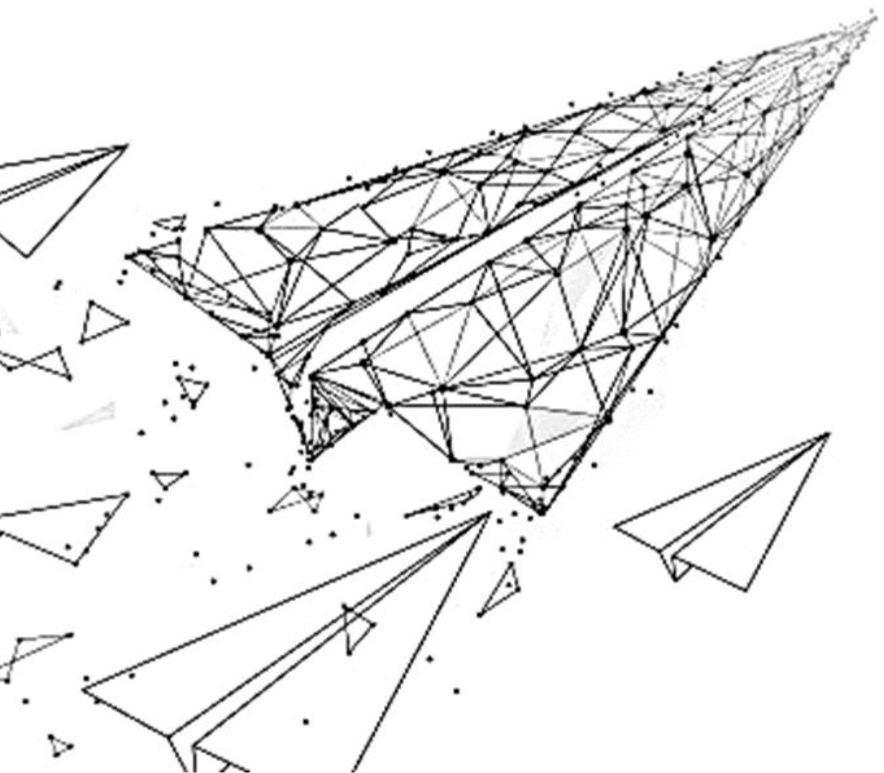
2020

YIFi Stake Contract Business Function Audit Report



File No.: DP- 20201204YIFi

YIFi Stake Contract Business Function Audit Repor

DATE: 04/12/2020

Contract Address Link:

https://etherscan.io/address/0xF0ca8e0B6Cc679D0D946b7470e26920601Ff1C37#code

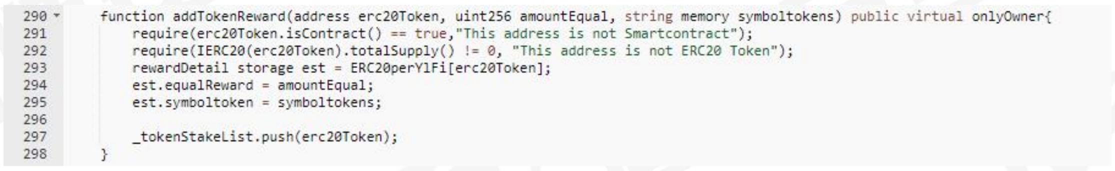
Note: the following tags represent different meanings of the test result, the customers who read this doc can refer to these tags to use this smart contract.

PASS	Means that the corresponding function has no issues in execution of business logic and design. Or means that the found issues which don' t affect the contract logic can be ignored.
FAIL	Means that the corresponding function has serious security issues which will cause the asset loss or other abnormal execution errors.
MISS	Means that the corresponding function is not implemented completely or that the business logic is not designed perfectly.

I、Function Analysis:

(1) Function AddTokenReward

➢ **Description:** As shown in the figure below, the contract implements the addTokenReward function for the owner to add the specified token as stake reward. The contract owner can call this function to add the rewardDetail of the specified reward token. When the user stakes, he can select the added reward token as the stake reward (selecting the unadded reward token for stake will not claim the reward).



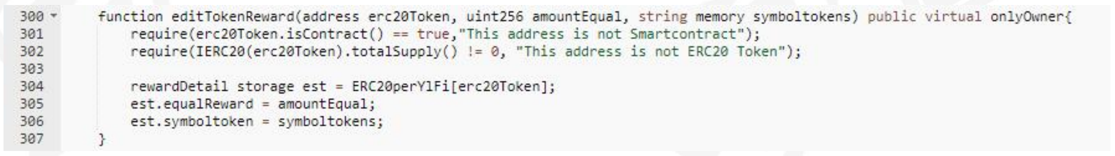
```
function addTokenReward(address erc20Token, uint256 amountEqual, string memory symboltokens) public virtual onlyOwner{
    require(erc20Token.isContract() == true,"This address is not Smartcontract");
    require(IERC20(erc20Token).totalSupply() != 0, "This address is not ERC20 Token");
    rewardDetail storage est = ERC20perYlFi[erc20Token];
    est.equalReward = amountEqual;
    est.symboltoken = symboltokens;

    _tokenStakeList.push(erc20Token);
}
```

Figure 1 source code of function addTokenReward

- **Related Functions:** addTokenReward、isContract、totalSupply
- **Test Result:** PASS

(2) Functions EditTokenReward

- **Description：** As shown in the figure below, the contract implements the editTokenReward function for the owner to modify the rewardDetail of the specified reward token. The contract owner can call this function to modify the rewardDetail of the specified reward token. When the user has a stake, modifying the rewardDetail of the specified reward token will affect the user to claim the normal stake reward.



```
function editTokenReward(address erc20Token, uint256 amountEqual, string memory symboltokens) public virtual onlyOwner{
    require(erc20Token.isContract() == true,"This address is not Smartcontract");
    require(IERC20(erc20Token).totalSupply() != 0, "This address is not ERC20 Token");

    rewardDetail storage est = ERC20perYlFi[erc20Token];
    est.equalReward = amountEqual;
    est.symboltoken = symboltokens;
}
```

Figure 2 source code of function editTokenReward

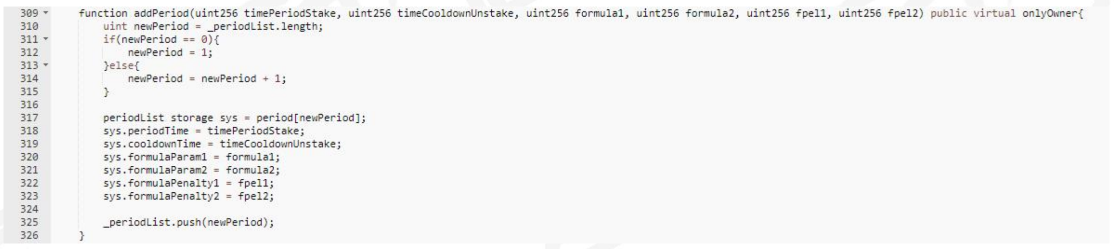
- **Related Functions:** editTokenReward、isContract、totalSupply
- **Security Suggestion:** It is recommended to delete this function.

- **Fix Result:** Ignored.

- **Test Result:** PASS

(3) Function addPeriod

- **Description:** As shown in the figure below, the contract implements addPeriod function for the owner to add a specified stake period. The contract owner can call this function to add a specified stake period. When the user stakes, he can choose the added stake period mode for staking (selecting the unadded stake period mode for staking will not claim rewards).



```
    function addPeriod(uint256 timePeriodStake, uint256 timeCooldownUnstake, uint256 formula1, uint256 formula2, uint256 fpel1, uint256 fpel2) public virtual onlyOwner{
        uint newPeriod = _periodList.length;
        if(newPeriod == 0){
            newPeriod = 1;
        }else{
            newPeriod = newPeriod + 1;
        }

        periodList storage sys = period[newPeriod];
        sys.periodTime = timePeriodStake;
        sys.cooldownTime = timeCooldownUnstake;
        sys.formulaParam1 = formula1;
        sys.formulaParam2 = formula2;
        sys.formulaPenalty1 = fpel1;
        sys.formulaPenalty2 = fpel2;

        _periodList.push(newPeriod);
    }
```

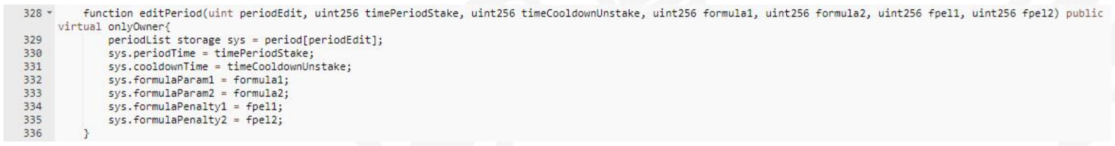
Figure 3 source code of function addPeriod

- **Related Functions:** addPeriod

- **Test Result:** PASS

(4) Function editPeriod

- **Description:** As shown in the figure below, the contract implements the editPeriod function for the owner to modify the specified stake period mode. The contract owner can call this function to modify the specified stake period mode. When the user has a stake, the contract owner's modification of the specified stake period mode may affect the amount of staked tokens withdrawn by the user and the amount of stake rewards claimed.



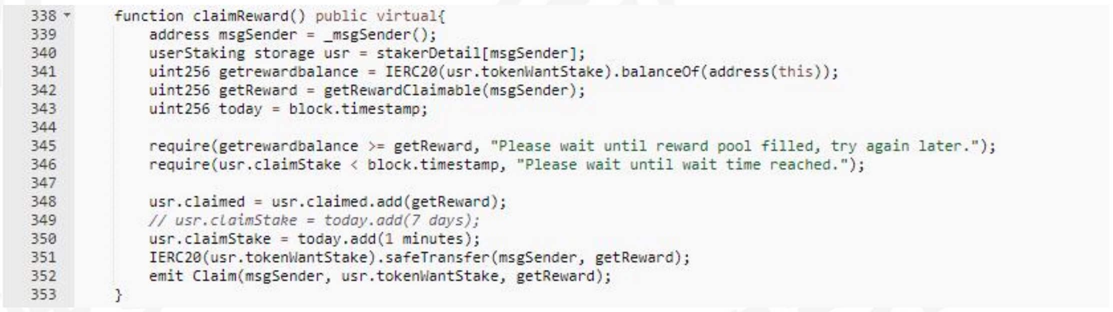
```
    function editPeriod(uint periodEdit, uint256 timePeriodStake, uint256 timeCooldownUnstake, uint256 formula1, uint256 formula2, uint256 fpel1, uint256 fpel2) public
virtual onlyOwner{
        periodList storage sys = period[periodEdit];
        sys.periodTime = timePeriodStake;
        sys.cooldownTime = timeCooldownUnstake;
        sys.formulaParam1 = formula1;
        sys.formulaParam2 = formula2;
        sys.formulaPenalty1 = fpel1;
        sys.formulaPenalty2 = fpel2;
    }
```

Figure 4 source code of function editPeriod

- **Related Functions:** editTokenReward
- **Security Suggestion:** It is recommended to delete this function.
- **Fix Result:** Ignored.
- **Test Result:** PASS

(5) Function ClaimReward

- **Description:** As shown in the figure below, the contract implements the claimReward function for users to withdraw stake rewards. Any user can call this function to withdraw stake rewards. The amount of reward tokens under this contract address is greater than the amount claimed by the user, the current time is greater than claimStake, and the user can continue to claim 7 days after claimed it, and transfer the tokens of the amount of the user stake reward to the user address.



```
function claimReward() public virtual{
    address msgSender = _msgSender();
    userStaking storage usr = stakerDetail[msgSender];
    uint256 getrewardbalance = IERC20(usr.tokenWantStake).balanceOf(address(this));
    uint256 getReward = getRewardClaimable(msgSender);
    uint256 today = block.timestamp;

    require(getrewardbalance >= getReward, "Please wait until reward pool filled, try again later.");
    require(usr.claimStake < block.timestamp, "Please wait until wait time reached.");

    usr.claimed = usr.claimed.add(getReward);
    // usr.claimStake = today.add(7 days);
    usr.claimStake = today.add(1 minutes);
    IERC20(usr.tokenWantStake).safeTransfer(msgSender, getReward);
    emit Claim(msgSender, usr.tokenWantStake, getReward);
}
```

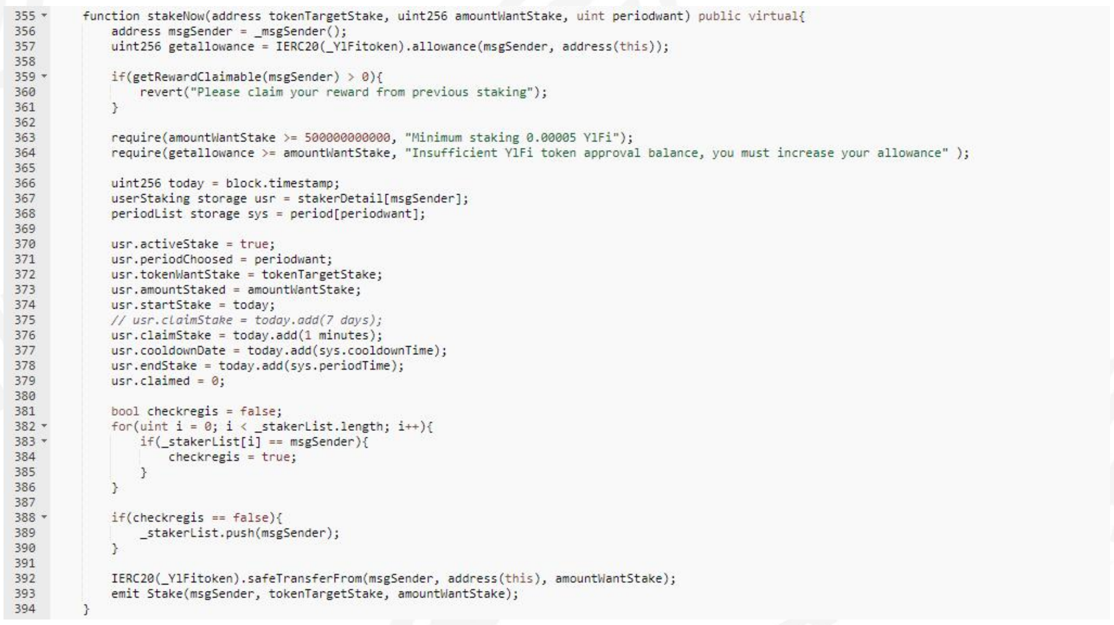
Figure 5 source code of function editPeriod

- **Related Functions:** claimReward、_msgSender、balanceOf、getRewardClaimable、safeTransfer

- **Test Result:** PASS

(6) Function StakeNow

- **Description:** As shown in the figure below, the contract implements the stakeNow function for users to stake tokens. Any user pre-approves this contract address, and then calls this function to stake tokens. The user is required to have no stake rewards and the stake amount is not less than 0.0000005 YLFi, the allowance is greater than the stake amount. After the stake, record the user's stake information and add the address to the list of stake user and transfer the user's staked tokens to this contract address (note: if the user claims the current stake reward and simultaneously stake again, it will cover the user's previous stake information, resulting in the failure to withdraw the last staked tokens).



```solidity
    function stakeNow(address tokenTargetStake, uint256 amountWantStake, uint periodwant) public virtual{
        address msgSender = _msgSender();
        uint256 getallowance = IERC20(_Y1Fitoken).allowance(msgSender, address(this));

        if(getRewardClaimable(msgSender) > 0){
            revert("Please claim your reward from previous staking");
        }

        require(amountWantStake >= 500000000000, "Minimum staking 0.00005 Y1Fi");
        require(getallowance >= amountWantStake, "Insufficient Y1Fi token approval balance, you must increase your allowance" );

        uint256 today = block.timestamp;
        userStaking storage usr = stakerDetail[msgSender];
        periodList storage sys = period[periodwant];

        usr.activeStake = true;
        usr.periodChoosed = periodwant;
        usr.tokenWantStake = tokenTargetStake;
        usr.amountStaked = amountWantStake;
        usr.startStake = today;
        // usr.claimStake = today.add(7 days);
        usr.claimStake = today.add(1 minutes);
        usr.cooldownDate = today.add(sys.cooldownTime);
        usr.endStake = today.add(sys.periodTime);
        usr.claimed = 0;

        bool checkregis = false;
        for(uint i = 0; i < _stakerList.length; i++){
            if(_stakerList[i] == msgSender){
                checkregis = true;
            }
        }

        if(checkregis == false){
            _stakerList.push(msgSender);
        }

        IERC20(_Y1Fitoken).safeTransferFrom(msgSender, address(this), amountWantStake);
        emit Stake(msgSender, tokenTargetStake, amountWantStake);
    }
```

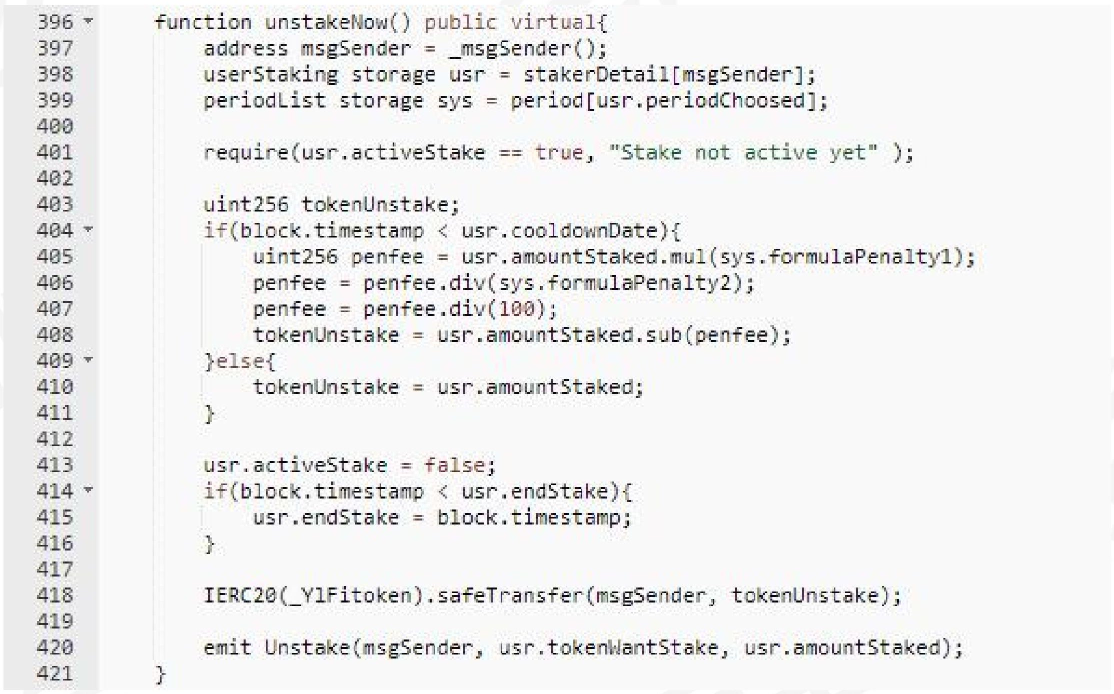
Figure 6 source code of function stakeNow

- **Related Functions:** stakeNow、safeTransfer、getRewardClaimable

- **Test Result:** PASS

(7) Function UnstakeNow

- **Description:** As shown in the figure below, the contract implements the unstakeNow function for users to claim stake tokens. The staked user can call this function to withdraw staked tokens, requiring the user's stake status to be true. If the current time is less than cooldownDate, a certain amount of staked tokens will be deducted as punishment. If the current time is greater than cooldownDate, the user can withdraw all staked tokens, modify the user's stake status to false, and transfer the user's staked tokens to the user address.



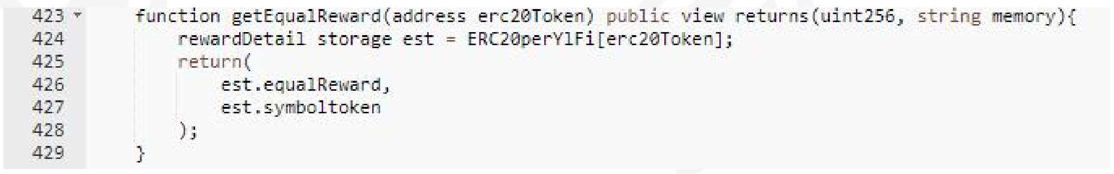
```
function unstakeNow() public virtual{
    address msgSender = _msgSender();
    userStaking storage usr = stakerDetail[msgSender];
    periodList storage sys = period[usr.periodChoosed];

    require(usr.activeStake == true, "Stake not active yet" );

    uint256 tokenUnstake;
    if(block.timestamp < usr.cooldownDate){
        uint256 penfee = usr.amountStaked.mul(sys.formulaPenalty1);
        penfee = penfee.div(sys.formulaPenalty2);
        penfee = penfee.div(100);
        tokenUnstake = usr.amountStaked.sub(penfee);
    }else{
        tokenUnstake = usr.amountStaked;
    }

    usr.activeStake = false;
    if(block.timestamp < usr.endStake){
        usr.endStake = block.timestamp;
    }

    IERC20(_Y1Fitoken).safeTransfer(msgSender, tokenUnstake);

    emit Unstake(msgSender, usr.tokenWantStake, usr.amountStaked);
}
```

Figure 7 source code of function unstakeNow

- **Related Functions:** UnstakeNow、SafeTransfer
- **Test Result:** PASS

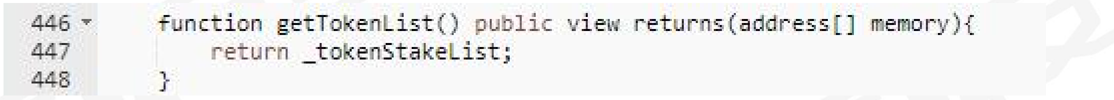
(8) Functions GetEqualReward & GetTokenList

- **Description:** As shown in the figure below, the contract implements the getEqualReward & sgetTokenList functions for users to query the rewardDetail of the specified reward token address and reward token list.



```
function getEqualReward(address erc20Token) public view returns(uint256, string memory){
    rewardDetail storage est = ERC20perY1Fi[erc20Token];
    return(
        est.equalReward,
        est.symboltoken
    );
}
```

Figure 8 source code of function getEqualReward



```
function getTokenList() public view returns(address[] memory){
    return _tokenStakeList;
}
```

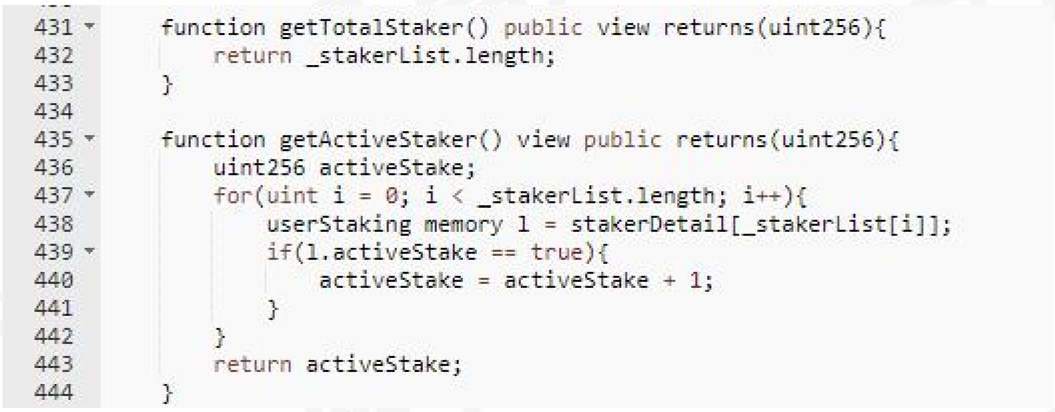
Figure 9 source code of function getTokenList

➢ **Related Functions:** GetEqualReward、GetTokenList

➢ **Test Result:** PASS

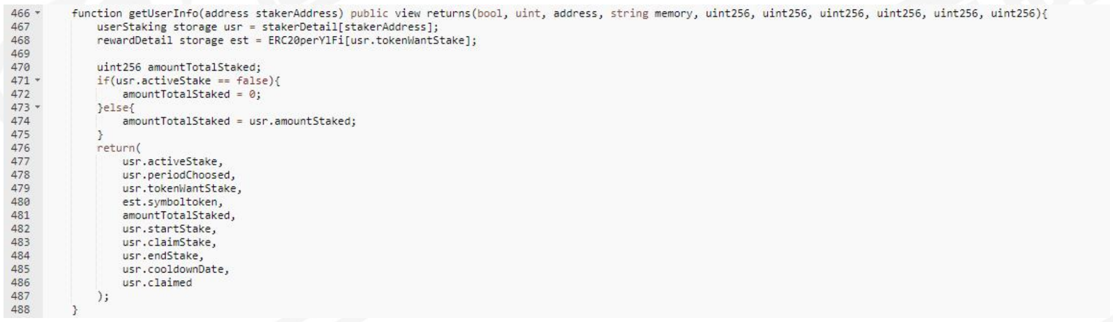
(9) Functions GetTotalStaker & GetActiveStaker & GetUserInfo

➢ **Description:** As shown in the figure below, the contract implements the getTotalStaker & getActiveStaker & getUserInfo functions to query the current total number of stak users & the number of users currently staked & the stake information of the current stake users.



```
    function getTotalStaker() public view returns(uint256){
        return _stakerList.length;
    }

    function getActiveStaker() view public returns(uint256){
        uint256 activeStake;
        for(uint i = 0; i < _stakerList.length; i++){
            userStaking memory l = stakerDetail[_stakerList[i]];
            if(l.activeStake == true){
                activeStake = activeStake + 1;
            }
        }
        return activeStake;
    }
```

Figure 10 source code of functions getTotalStaker & getActiveStaker



```
    function getUserInfo(address stakerAddress) public view returns(bool, uint, address, string memory, uint256, uint256, uint256, uint256, uint256, uint256){
        userStaking storage usr = stakerDetail[stakerAddress];
        rewardDetail storage est = ERC20perYlFi[usr.tokenWantStake];

        uint256 amountTotalStaked;
        if(usr.activeStake == false){
            amountTotalStaked = 0;
        }else{
            amountTotalStaked = usr.amountStaked;
        }
        return(
            usr.activeStake,
            usr.periodChoosed,
            usr.tokenWantStake,
            est.symboltoken,
            amountTotalStaked,
            usr.startStake,
            usr.claimStake,
            usr.endStake,
            usr.cooldownDate,
            usr.claimed
        );
    }
```

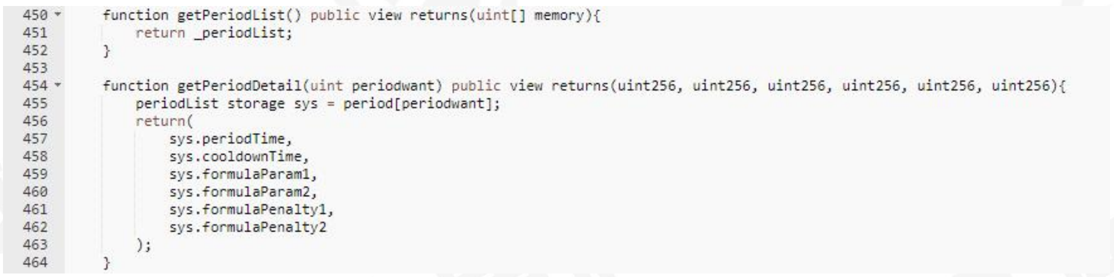
Figure 11 source code of function getUserInfo

➢ **Related Functions:** GetTotalStaker、GetActiveStaker、GetUserInfo

➢ **Test Result:** PASS

(10) Functions GetPeriodList & GetPeriodDetail

➢ **Description:** As shown in the figure below, the contract implements the functions getPeriodList & getPeriodDetail to query the current stake period list and specified stake periodDetail.



```
function getPeriodList() public view returns(uint[] memory){
    return _periodList;
}

function getPeriodDetail(uint periodwant) public view returns(uint256, uint256, uint256, uint256, uint256, uint256){
    periodList storage sys = period[periodwant];
    return(
        sys.periodTime,
        sys.cooldownTime,
        sys.formulaParam1,
        sys.formulaParam2,
        sys.formulaPenalty1,
        sys.formulaPenalty2
    );
}
```

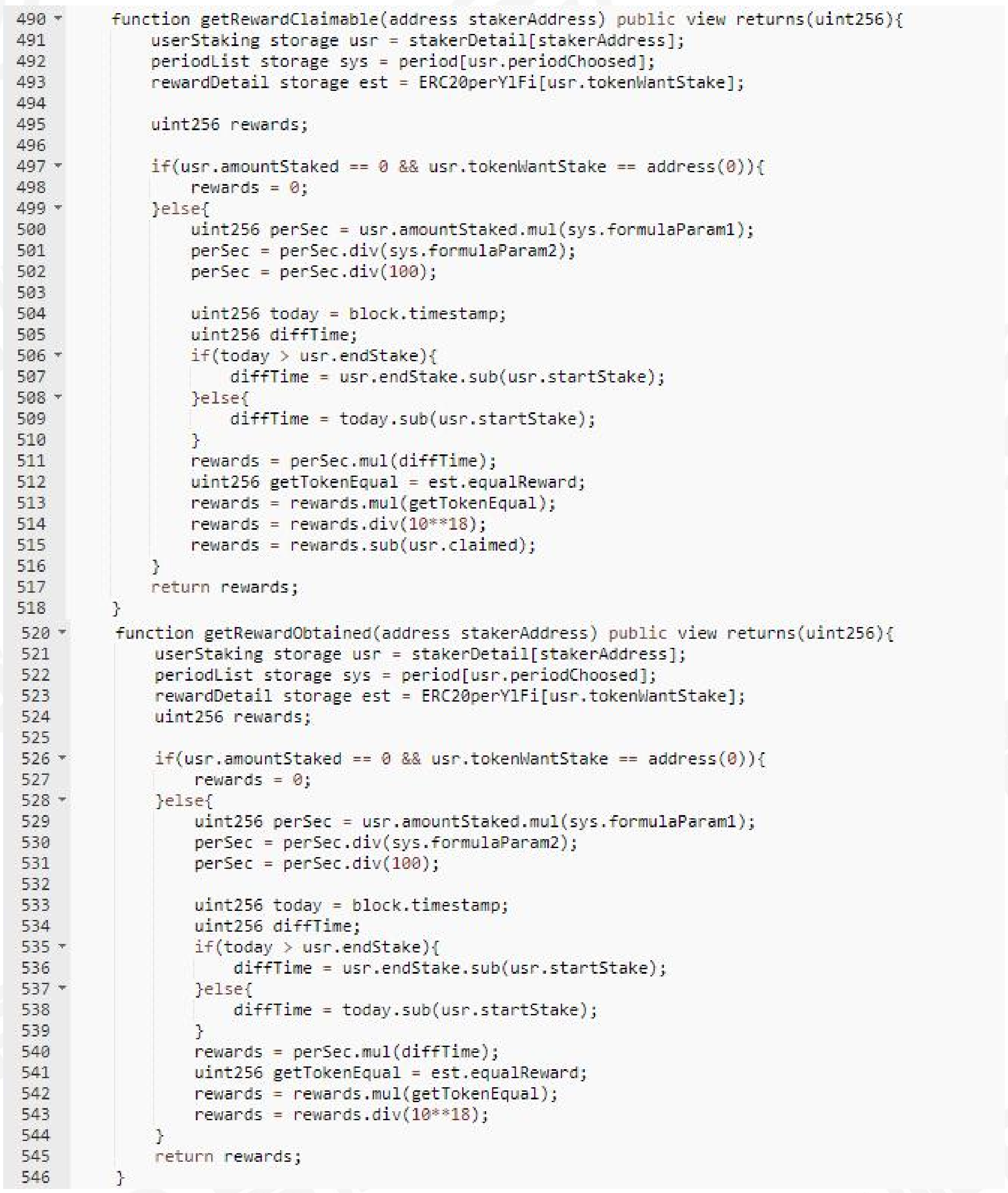
Figure 12 source code of functions getPeriodList & getPeriodDetail

➢ **Related functions:** GetPeriodList、GetPeriodDetail

➢ **Test Result:** PASS

(11) Functions GetRewardClaimable & GetRewardObtained

➢ **Description:** As shown in the figure below, the contract implements the function getRewardClaimable & getRewardObtained to query the current stake users’ available reward and the generated rewards amount stake users.



```
function getRewardClaimable(address stakerAddress) public view returns(uint256){
    userStaking storage usr = stakerDetail[stakerAddress];
    periodList storage sys = period[usr.periodChoosed];
    rewardDetail storage est = ERC20perYlFi[usr.tokenWantStake];

    uint256 rewards;

    if(usr.amountStaked == 0 && usr.tokenWantStake == address(0)){
        rewards = 0;
    }else{
        uint256 perSec = usr.amountStaked.mul(sys.formulaParam1);
        perSec = perSec.div(sys.formulaParam2);
        perSec = perSec.div(100);

        uint256 today = block.timestamp;
        uint256 diffTime;
        if(today > usr.endStake){
            diffTime = usr.endStake.sub(usr.startStake);
        }else{
            diffTime = today.sub(usr.startStake);
        }
        rewards = perSec.mul(diffTime);
        uint256 getTokenEqual = est.equalReward;
        rewards = rewards.mul(getTokenEqual);
        rewards = rewards.div(10**18);
        rewards = rewards.sub(usr.claimed);
    }
    return rewards;
}
function getRewardObtained(address stakerAddress) public view returns(uint256){
    userStaking storage usr = stakerDetail[stakerAddress];
    periodList storage sys = period[usr.periodChoosed];
    rewardDetail storage est = ERC20perYlFi[usr.tokenWantStake];
    uint256 rewards;

    if(usr.amountStaked == 0 && usr.tokenWantStake == address(0)){
        rewards = 0;
    }else{
        uint256 perSec = usr.amountStaked.mul(sys.formulaParam1);
        perSec = perSec.div(sys.formulaParam2);
        perSec = perSec.div(100);

        uint256 today = block.timestamp;
        uint256 diffTime;
        if(today > usr.endStake){
            diffTime = usr.endStake.sub(usr.startStake);
        }else{
            diffTime = today.sub(usr.startStake);
        }
        rewards = perSec.mul(diffTime);
        uint256 getTokenEqual = est.equalReward;
        rewards = rewards.mul(getTokenEqual);
        rewards = rewards.div(10**18);
    }
    return rewards;
}
```

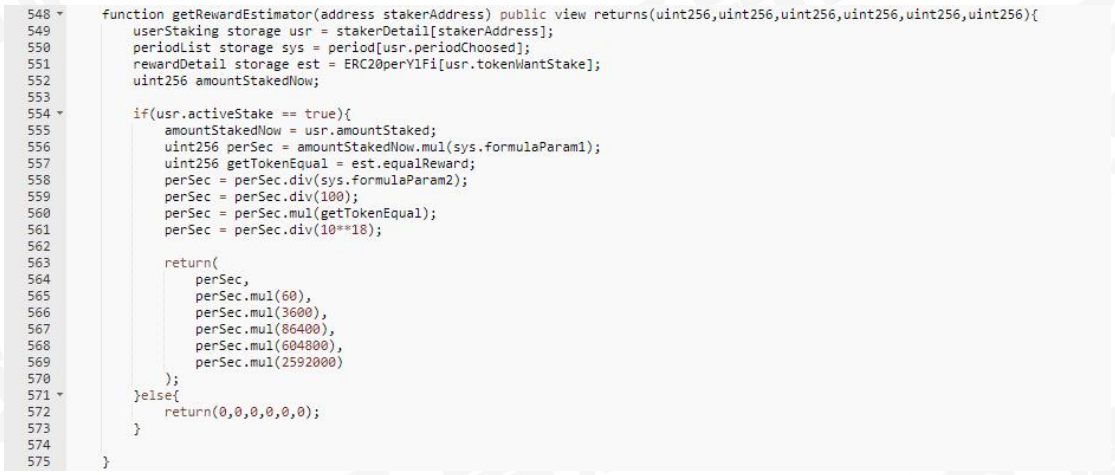
Figure 13 source code of functions getRewardClaimable & getRewardObtained

- **Related functions:** GetRewardClaimable、GetRewardObtained

- **Test Result:** PASS

(12) Function GetRewardEstimator

- **Description:** As shown in the figure below, the contract implements the getRewardEstimator function to query the stake rewards that can be claimed in 1 minute/1 hour/1 week/1 month if there is a stake at the specified address. If the user does not have a stake, return 0.



```
function getRewardEstimator(address stakerAddress) public view returns(uint256,uint256,uint256,uint256,uint256,uint256){
    userStaking storage usr = stakerDetail[stakerAddress];
    periodList storage sys = period[usr.periodChoosed];
    rewardDetail storage est = ERC20perYlFi[usr.tokenWantStake];
    uint256 amountStakedNow;

    if(usr.activeStake == true){
        amountStakedNow = usr.amountStaked;
        uint256 perSec = amountStakedNow.mul(sys.formulaParam1);
        uint256 getTokenEqual = est.equalReward;
        perSec = perSec.div(sys.formulaParam2);
        perSec = perSec.div(100);
        perSec = perSec.mul(getTokenEqual);
        perSec = perSec.div(10**18);

        return(
            perSec,
            perSec.mul(60),
            perSec.mul(3600),
            perSec.mul(86400),
            perSec.mul(604800),
            perSec.mul(2592000)
        );
    }else{
        return(0,0,0,0,0,0);
    }

}
```

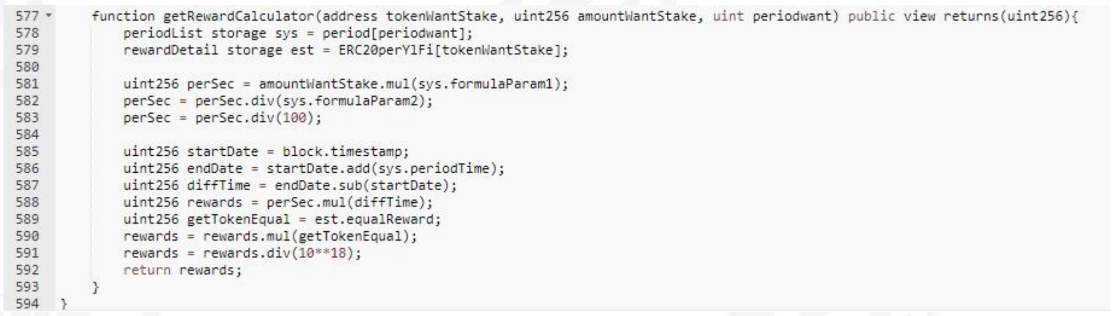
Figure 14 source code of function getRewardEstimator

- **Related Functions:** getRewardEstimator

- **Test Result:** PASS

(13) Function GetRewardCalculator

- **Description:** As shown in the figure below, the contract implements the getRewardCalculator function to query specified address’ s the selected specified period, the specified reward token, and the specified the number of rewards that can be claimed after meets the conditions of the period mode.



```
    function getRewardCalculator(address tokenWantStake, uint256 amountWantStake, uint periodwant) public view returns(uint256){
        periodList storage sys = period[periodwant];
        rewardDetail storage est = ERC20perYlFi[tokenWantStake];

        uint256 perSec = amountWantStake.mul(sys.formulaParam1);
        perSec = perSec.div(sys.formulaParam2);
        perSec = perSec.div(100);

        uint256 startDate = block.timestamp;
        uint256 endDate = startDate.add(sys.periodTime);
        uint256 diffTime = endDate.sub(startDate);
        uint256 rewards = perSec.mul(diffTime);
        uint256 getTokenEqual = est.equalReward;
        rewards = rewards.mul(getTokenEqual);
        rewards = rewards.div(10**18);
        return rewards;
    }
}
```

Figure 15 source code of function getRewardEstimator

- **Related Functions:** getRewardCalculator

- **Test Result:** PASS

II、Conclusion

The design & implementation of the YLFiStake contract are tested and reviewed detailly, all the issues found during the test process have been informed to the project party, and the feedback result is to be ignored. Among these issues, the issue of modifying stake period and rewardDetail by owner will affect the user’ s claiming stake rewards and withdrawing staked tokens, cautiously using it is recommended.

Note: the stake token is YLFi, and the owner of YLFi contract can destroy any holder’ s token.

The overall test & review result of YLFiStake contract is PASS.

盾牌安全

YlFi Stake Contract Business Function Audit Report

Official website：http://www.dpanquan.com

Official email：service@dpanquan.com