成都链安
BEOSIN

# Smart contract security audit report

**Audit Number：202009131910**

**Report Query Name: dBalancerPool**

**Smart Contract Name：**

dBalancerPool

**Smart Contract Address：**

0x949414d01C913e71E3C141f5e607A4d8bBbb3b25

**Smart Contract Address Link：**

https://etherscan.io/address/0x949414d01c913e71e3c141f5e607a4d8bbbb3b25#code

**Start Date：2020.09.11**

**Completion Date：2020.09.13**

**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 dBalancerPool, including Coding Standards, Security, and Business Logic. **The dBalancerPool contract 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.
- 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:

  As shown in Figure 1, 2 below, the "invite-stake-reward" mode of the contract needs to initialize the relevant parameters (*rewardRate, lastUpdateTime, periodFinish, invLevel*), call the *notifyRewardAmount* function by the specified reward distribution manager address *rewardDistribution*, and enter the initial reward used to calculate the *rewardRate*, initialize the stake and reward related parameters.

```
IERC20 public syfi = IERC20(0xdc38a4846d811572452cB4CE747dc9F5F509820f);
IERC20 public inv = IERC20(0xfe26bb3644153BCDE31237E76eE337d39291420E);
uint256 public constant DURATION = 7 days;

uint256 public initreward = 10000*1e18;
uint256 public starttime = 1599408000; //utc+8 2020 09-07 00:00:00

uint256 public periodFinish = 0;
uint256 public rewardRate = 0;
uint256 public lastUpdateTime;
uint256 public rewardPerTokenStored;
mapping(address => uint256) public userRewardPerTokenPaid;
mapping(address => uint256) public rewards;

/**invitation**/
uint256 public oneFactor = 60;
uint256 public twoFactor = 30;
uint256 public threeFactor = 10;
uint256 public invLevel = 3;
mapping(address => uint256) public claimRewards;
mapping(address => uint256) public claimINVRewards;

mapping(address => bytes32) public invCodes;
mapping(bytes32 => address) public invRefs;
mapping(address => address[]) public invAddrs;
mapping(address => address) public isUseCode;
```

Figure 1 related parameter source code

```
function notifyRewardAmount(uint256 reward)
    external
    onlyRewardDistribution
    updateReward(address(0))
{
    if (block.timestamp >= periodFinish) {
        rewardRate = reward.div(DURATION);
    } else {
        uint256 remaining = periodFinish.sub(block.timestamp);
        uint256 leftover = remaining.mul(rewardRate);
        rewardRate = reward.add(leftover).div(DURATION);
    }
    syfi.mint(address(this),reward);
    lastUpdateTime = block.timestamp;
    if(block.timestamp < starttime){
        periodFinish = starttime.add(DURATION);
    }else {
        periodFinish = block.timestamp.add(DURATION);
    }
    emit RewardAdded(reward);
}
```

Figure 2 Source code of function *notifyRewardAmount*

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

3.2 Set Invitation Code

- Description:

  As shown in Figure 3 below, the contract implements the *setInvCode* function to set the invitation code. Users can set their own invitation code by calling this function and entering the invitation code of the superior inviter (the owner must set for the first time, and the owner cannot participate in the token stake).

```
function setInvCode(bytes32 code,bytes32 superCode) public {
    require(invCodes[msg.sender] == 0);
    require(invRefs[code] == address(0));
    require(isOwner() || invRefs[superCode] != address(0));
    invCodes[msg.sender] = code;
    invRefs[code] = msg.sender;
    isUseCode[msg.sender] = invRefs[superCode];
}
```

Figure 3 Source code of function setInvCode

- Related functions: *setInvCode*
- Result: Pass

3.3 Stake tokens

- Description:

  As shown in Figure 4 below, the contract implements the *stake* function to stake the BPT tokens. The user is required to call the *setInvCode* function to set their own invitation code with the invitation code of the superior inviter before they stake. The user approve the contract address in advance. By calling the *transferFrom* function in the BPT contract, the contract address transfers the specified amount of BPT tokens to the contract address on behalf of the user; This function restricts the user to call only after the "invite-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 stake(uint256 amount) public updateReward(msg.sender) checkhalve checkStart {
    require(amount > 0, "Cannot stake 0");
    /**invitation**/
    require(isUseCode[msg.sender] != address(0),"Must have a invite code");
    address superPerson = isUseCode[msg.sender];
    invAddrs[superPerson].push(msg.sender);
    /****/
    super.stake(amount);
    emit Staked(msg.sender, amount);
}
```

Figure 4 Source code of function *stake*

- Related functions: *stake, safeTransferFrom,rewardPerToken, lastTimeRewardApplicable, earned, balanceOf*
- Result: Pass

3.4 Withdraw invitation rewards

- Description:

  As shown in Figure 5&6 below, the contract implements the *getInvRewards* function to calculate the caller's invitation reward (INV token). The caller can call this function to calculate the invitation reward based on the invitation level set by the owner. The default is level 3 (the owner can call the *levelEmergencyChange* function to modify the invitation level, and modify the invitation level will affect the invitation reward of the inviter).

```
function getInvRewards() public {
    require(isOpenWithdraw == true,"Not open inv claim");
    uint256 reward = earnedInv(msg.sender);
    if (reward > 0) {
        claimINVRewards[msg.sender] = claimINVRewards[msg.sender].add(reward);
        inv.safeTransfer(msg.sender, reward);
        emit RewardINVPaid(msg.sender, reward);
    }
}
```

Figure 5 Source code of function *getInvRewards*

```
function levelEmergencyChange(uint256 l) public onlyOwner {
    invLevel = l;
}
```

Figure 6 Source code of function *levelEmergencyChange*

- Related functions: *getInvRewards, earnedInv, earned, calcInv, levelEmergencyChange*
- Result: Pass

3.5 Withdraw tokens

- Description:

  As shown in Figure 7 below, the contract implements the *withdraw* function to withdraw the BPT tokens. By calling the *transfer* function in the BPT contract, the contract address transfers the specified amount of BPT tokens to the user; This function restricts the user 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 withdraw(uint256 amount) public updateReward(msg.sender) checkhalve checkStart {
    require(amount > 0, "Cannot withdraw 0");
    super.withdraw(amount);
    emit Withdrawn(msg.sender, amount);
}
```

Figure 7 Source code of function *withdraw*

- Related functions: *withdraw, safeTransfer, rewardPerToken, lastTimeRewardApplicable, earned, balanceOf*
- Result: Pass

3.6 Withdraw rewards (SYFI token)

- Description:

As shown in Figure 8 below, the contract implements the *getReward* function to withdraw the rewards (SYFI token). By calling the *transfer* function in the SYFI contract, the contract address transfers the specified amount (all rewards of caller) of SYFI tokens to the user; This function restricts the user 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;
        claimRewards[msg.sender] = claimRewards[msg.sender].add(reward);
        syfi.safeTransfer(msg.sender, reward);
        emit RewardPaid(msg.sender, reward);
    }
}
```

Figure 8 Source code of function *getReward*

- Related functions: *getReward, safeTransfer, rewardPerToken, lastTimeRewardApplicable, earned, balanceOf*

- Result: Pass

3.7 Exit the stake participation

- Description:

As shown in Figure 9 below, the contract implements the *exit* function to close the participation of "invite-stake-reward" mode. Call the *withdraw* function to withdraw all stake BPT tokens, call the *getReward* function to receive all rewards. If the "invite-stake-reward" mode is turned on, call the *getInvRewards* function to withdraw the caller's invitation reward, and end the participation of "invite-stake-reward" mode. The user address cannot get new rewards because the balance of USDT tokens already staked is empty.

```
function exit() external {
    withdraw(balanceOf(msg.sender));
    getReward();
    if(isOpenWithdraw == true){
        getInvRewards();
    }
}
```

Figure 9 Source code of function *exit*

- Related functions: *exit, withdraw, getReward, rewardPerToken, lastTimeRewardApplicable, earned, balanceOf, getInvRewards, earnedInv, calcInv, levelEmergencyChange*

- Result: Pass

3.8 Reward related data query function

- Description:

As shown in Figure 10&11 below, the user can query the caller's current invitation token reward by calling the *earnedInv* function; contract users 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 USDT token; calling the *earned* function can query the total gettable stake rewards of the specified address.

```
function earnedInv(address account) public view returns(uint256) {
    address[] memory list_1 = invAddrs[account];
    uint256 total = 0;
    uint256 curEarn = earned(account).add(claimRewards[account]);
    for(uint32 i = 0; i<list_1.length; i++){
        total = total.add(calcInv(list_1[i],oneFactor,curEarn));
        if(invLevel == 1 || invAddrs[list_1[i]].length == 0){
            continue;
        }
        address[] memory list_2 = invAddrs[list_1[i]];
        for(uint32 j = 0; j<list_2.length; j++) {
            total = total.add(calcInv(list_2[j],twoFactor,curEarn));
            if(invLevel == 2 || invAddrs[list_2[j]].length == 0){
                continue;
            }
            address[] memory list_3 = invAddrs[list_2[j]];
            for(uint32 k = 0; k<list_3.length; k++){
                total = total.add(calcInv(list_3[k],threeFactor,curEarn));
            }
        }
    }
    return total.sub(claimINVRewards[account]);
}
```

Figure 10 Source code of function earnedInv

```
function lastTimeRewardApplicable() public view returns (uint256) {
    return Math.min(block.timestamp, periodFinish);
}

function rewardPerToken() public view returns (uint256) {
    if (totalSupply() == 0) {
        return rewardPerTokenStored;
    }
    return
        rewardPerTokenStored.add(
            lastTimeRewardApplicable()
                .sub(lastUpdateTime)
                .mul(rewardRate)
                .mul(1e18)
                .div(totalSupply())
        );
}

function earned(address account) public view returns (uint256) {
    return
        balanceOf(account)
            .mul(rewardPerToken().sub(userRewardPerTokenPaid[account]))
            .div(1e18)
            .add(rewards[account]);
}
```

Figure 11 source code of related functions

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

## 4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the dBalancerPool project. The SYFI contract implements the mint function, and the mint function can issue tokens without a certain cap. The governance management address (initially is deployer address) can add minter, and the minter address can call the mint function to issue tokens without limitation, affecting the

token swap in the specified swap pool. Cautiously using mint function and adding minter are recommended.
The overall audit result of dBalancerPoolproject is **Pass**.


成都链安

BEOSIN

**Official Website**

https://lianantech.com

**E-mail**

vaas@lianantech.com

**Twitter**

https://twitter.com/Beosin_com