This is a peer-review audit of aiVaultUSDT.sol.

Peer-review participants: Tien-Thao Nguyen, Le-Duc Pham, The-Quan Phung, Thanh-Chung Dao Ph.D., Binh-Minh Nguyen Ph.D., and Ba-Lam Do Ph.D.
Editor: Le-Duc Pham

Date: October 08, 2020 at 14:00
Place: BKC Lab, SOICT, Hanoi University of Science and Technology
Website: https://bkc-group.github.io

A. General overview and comments

- This is Oraichain's FIRST product using AI services on Oraichain to perform yield farming. **ORAI token is used as the governance token of yAI.Finance.**
- Users can deposit **USDT** into the yAI.Finance vault mining contract to start farming. They also receive ORAI tokens as rewards weekly based on the shares.
- The deployer used SafeERC20 as well as SafeMath for calculation and token transfer. Most of the aiVaultUSDT functions are identical to those of yVault. Hence, the current issues of yVault have to be acknowledged when using aiVaultUSDT such as "Poisoning Baby Vault" .
- There are in total two new variables and five new functions in the contract. They are responsible for rewarding ORAI tokens for users that deposit USDT into the contract. Thus, we looked into all of them carefully, as this is the unique feature of aiVaultUSDT that stands out from the yVault.

B. Functions similar to YFI

1. **constructor()**
2. **balance()**
3. **setMin(uint256)**
4. **setGovernance(address)**
5. **setController(address)**
6. **available()**
7. **earn()**
8. **depositAll()**
9. **deposit(uint256)**
10. **withdrawAll()**
11. **harvest(address, uint256)**
12. **withdraw(uint256)**
13. **getPricePerFullShare()**

The above functions are mostly identical to those in the yVault smart contract. Only the data types are changed from uint to uint256.

C. New variables and functions added to the smart contract

- **rewardAddress** - This is the address of the ORAI reward contract. The address seems legit, and it can be adjusted by the governance address. LGTM
- **rewardAmount** = 60200. This is the initial value of the reward amount. LGTM
- **setRewardAmount(uint256)**: This is the function only for the *governance*. A setter method to update the reward amount

```
function setRewardAmount(uint _amount) external {
        require(msg.sender == governance, "!governance");
        rewardAmount = _amount;
    }
```

- **setRewardAddress(address)**: This is the function used only by the *governance*. A setter method to update the reward contract address. LGTM

```
function setRewardAddress(address _address) external {
        require(msg.sender == governance, "!governance");
        rewardAddress = address(_address);
    }
```

- **checkReward(address)**: This is a view function that calculates the reward amount of a user that can be received based on his aiUSDT balance.

```
function checkReward(address _reciver) public view
returns(uint256){
    uint256 _balance = balanceOf(_reciver);
    uint256 amount =
(_balance.mul(rewardAmount)).div(totalSupply()).mul(10**18);
    return amount;
}
```

Issues:

- There is a typo in the parameter “reciver”. It should be “receiver”. Severity: Low

- checkRewardBalance(): This is a view function that views the total reward balance stored in the reward address of a user. LGTM.

```
function checkRewardBalance() public view returns(uint256){
    uint256 _balance =
ERC20(rewardAddress).balanceOf(msg.sender);
    return _balance;
}
```

- reward(address[]): This is the main function that allows the sender to transfer rewards to a list of addresses.

```
function reward(address[] memory _addressList) public{
        // require(msg.sender == governance, "!governance");
        for (uint i=0; i<_addressList.length; i++) {
            address _address = _addressList[i];
            uint256 _balance = balanceOf(_address);
            if(_balance > 0){
                uint256 amount =
(_balance.mul(10**18).mul(rewardAmount)).div(totalSupply());

ERC20(rewardAddress).transferFrom(msg.sender,_address, amount);
            }
        }
    }
```

Issues:

- The address list does not identify duplicate elements. As a result, one account can receive rewards multiple times, and those that deserve to receive the rewards may not get what they want. This possible issue depends on the sender (the invoker of this function), so the sender should double check the address list before calling the function. Severity: Low to Medium
- There is no assurance that the sender has enough balance to reward all addresses. This could result in a situation where the function returns an error before finishing the loop, and the sender cannot keep track of which address that has been rewarded, and which has not. Severity: Low to Medium

References

1. aiVaultUSDT Contract:
https://etherscan.io/address/0x50410884462a075f27fd1c9030b955f2abe798d7#code