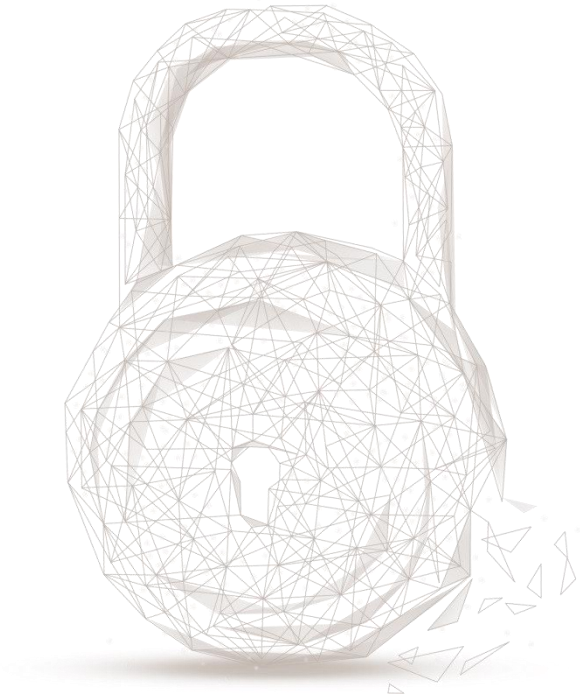
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202102021743

Smart Contract Info：

Smart Contract Name	Smart Contract Address	Smart Contract Address Link
DotLavaRewards	0x090fc08fd93b4ddeab5522d6c7d2b7a91ba562cb	https://scan.hecochain.com/address/0x090fc08fd93b4ddeab5522d6c7d2b7a91ba562cb#contracts
ETHLavaRewards	0x41e49031680cf425322232deebe9a7732963438a	https://scan.hecochain.com/address/0x41e49031680cf425322232deebe9a7732963438a#contracts
LavaLavaRewards	0x7e7766fcf7e6e7c361735fef786b5da6b8f65c93	https://scan.hecochain.com/address/0x7e7766fcf7e6e7c361735fef786b5da6b8f65c93#contracts

Start Date：2021.02.02

Completion Date：2021.02.02

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

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 DotLavaRewards/ETHLavaRewards/LavaLavaRewards, including Coding Standards, Security, and Business Logic. **The DotLavaRewards/ETHLavaRewards/LavaLavaRewards contracts 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 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. In this project, the contract
- Result: Pass

2.10 Replay Attack

- Description: Check whether 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

In this project, three "stake rewards" smart contracts were implemented based on the same code structure, namely DotLavaRewards,LavaLavaRewards and ETHLavaRewards. The code logic of DotLavaRewards and ETHLavaRewards is the same, but the logic of LavaLavaRewards is somewhat different. The following screenshots are based on DotLavaRewards and LavaLavaRewards.

3.1 Business analysis of Contract DotLavaRewards

(1) Stake Initialization

- Description: The "stake-reward" mode of the contract needs to initialize the relevant parameters (*rewardRate*, *lastUpdateTime*, *periodFinish*), 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. This function can be called by the specified address *rewardDistribution* at any time to control the reward rate and the key time judgment condition, even if the *rewardRate* is updated when the *checkhalve* modifier executes the logic, it can still be modified by entering the specified value reward in this function. If the value is too small, the user's reward will not match expectations.

```
    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);
        }
        lastUpdateTime = block.timestamp;
        periodFinish = block.timestamp.add(DURATION);
        emit RewardAdded(reward);
    }
}
```

Figure 1 source code of notifyRewardAmount(DotLavaRewards contract)

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

(2) Stake DOT tokens

- Description: The contract implements the *stake* function to stake the DOT tokens. The user need to *approve* the contract address in advance. By calling the *transferFrom* function in the DOT contract, the contract address transfers the specified amount of DOT tokens to the contract address on behalf of 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 through the modifier *checkhalve* to check whether the specified *periodFinish* is reached (a total of 4 weeks), and the *halveTimes* and the *periodFinish* are updated.

```
// stake visibility is public as overriding LPTokenWrapper's stake() function
function stake(uint256 amount) public updateReward(msg.sender) checkhalve checkStart{
    require(amount > 0, "Cannot stake 0");
    super.stake(amount);
    emit Staked(msg.sender, amount);
}
```

Figure 2 source code of stake function(1/2)(DotLavaRewards contract)

```
function stake(uint256 amount) public {
    _totalSupply = _totalSupply.add(amount);
    _balances[msg.sender] = _balances[msg.sender].add(amount);
    y.safeTransferFrom(msg.sender, address(this), amount);
}
```

Figure 3 source code of stake function(2/2)(DotLavaRewards contract)

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

(3) Withdraw DOT tokens

- Description: The contract implements the *withdraw* function to withdraw the DOT tokens. By calling the *transfer* function in the DOT token contract, the contract address transfers the specified amount of DOT 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*(a total of 4 weeks) is reached by the modifier *checkhalve*, and the *halveTimes* 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 4 source code of withdraw function(1/2) (DotLavaRewards contract)

```
function withdraw(uint256 amount) public {
    _totalSupply = _totalSupply.sub(amount);
    _balances[msg.sender] = _balances[msg.sender].sub(amount);
    y.safeTransfer(msg.sender, amount);
}
```

Figure 5 source code of withdraw function(2/2)(DotLavaRewards contract)

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

(4) Withdraw rewards (Lava token)

- Description: The contract implements the *getReward* function to withdraw the rewards (Lava token). By calling the *transfer* function in the Lava contract, the contract address transfers the specified amount (all rewards of caller) of Lava 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*(a total of 4 weeks) is reached by the modifier *checkhalve*, and the *halveTimes* 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;
        lava.safeTransfer(msg.sender, reward);
        emit RewardPaid(msg.sender, reward);
    }
}
```

Figure 6 source code of function getReward(DotLavaRewards contract)

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

(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 DOT, call the *getReward* function to receive all rewards. The user address cannot get new rewards because the balance of DOT tokens already staked is empty.

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

Figure 7 source code of function exit

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

(6) Reward related data query function

- Description: 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 Lava token; calling the *earned* function can query the total claimable stake rewards of the specified address.
- Related functions: *lastTimeRewardApplicable, rewardPerToken, earned*
- Result: Pass

3.2 Business analysis of Contract LavaLavaRewards

(1) Stake Lava tokens

- Description: The contract implements the *stake* function to stake the Lava tokens. The user need to *approve* the contract address in advance. By calling the *transferFrom* function in the Lava contract, the contract address transfers the specified amount of Lava tokens to the contract address on behalf of 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 through the modifier *checkhalve* to check whether the specified *periodFinish* is reached (a total of 4 weeks), Here, with the increase of the period, the *initreward* will increase, so the *rewardRate* will become larger. and the *rewardRate*, the *halveTimes* and the *periodFinish* are updated.

```
// stake visibility is public as overriding LPTokenWrapper's stake() function
function stake(uint256 amount) public updateReward(msg.sender) checkhalve checkStart{
    require(amount > 0, "Cannot stake 0");
    super.stake(amount);
    emit Staked(msg.sender, amount);
}
```

Figure 8 source code of stake function(1/2)(LavaLavaRewards)

```
function stake(uint256 amount) public {
    _totalSupply = _totalSupply.add(amount);
    _balances[msg.sender] = _balances[msg.sender].add(amount);
    y.safeTransferFrom(msg.sender, address(this), amount);
}
```

Figure 10 source code of stake function(2/2)(LavaLavaRewards)

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

(2) Withdraw Lava tokens

- Description: The contract implements the *withdraw* function to withdraw the Lava tokens. By calling the *transfer* function in the Lava token contract, the contract address transfers the specified amount of Lava 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*(a total of 4 weeks) is reached by the modifier *checkhalve*, Here, with the increase of the period, the initreward will increase, so the *rewardRate* will become larger. and the *rewardRate*, the *halveTimes* 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 11 source code of withdraw function(1/2) (LavaLavaRewards)

```
function withdraw(uint256 amount) public {
    _totalSupply = _totalSupply.sub(amount);
    _balances[msg.sender] = _balances[msg.sender].sub(amount);
    y.safeTransfer(msg.sender, amount);
}
```

Figure 12 source code of withdraw function(2/2)(LavaLavaRewards)

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

(3) Withdraw rewards (Lava token)

- Description: The contract implements the *getReward* function to withdraw the rewards (Lava token). By calling the *transfer* function in the Lava contract, the contract address transfers the specified amount (all rewards of caller) of Lava 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*(a total of 4 weeks) is reached by the modifier *checkhalve*, Here, with the increase of the period, the *initreward* will increase, so the *rewardRate* will become larger. and the *rewardRate*, the *halveTimes* 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;
        lava.safeTransfer(msg.sender, reward);
        emit RewardPaid(msg.sender, reward);
    }
}
```

Figure 9 source code of function getReward(LavaLavaRewards)

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

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts DotLavaRewards/LavaLavaRewards/ETHLavaRewards. The DotLavaRewards/LavaLavaRewards/ETHLavaRewards contracts passed all audit items, The overall audit result is **Pass.**

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com