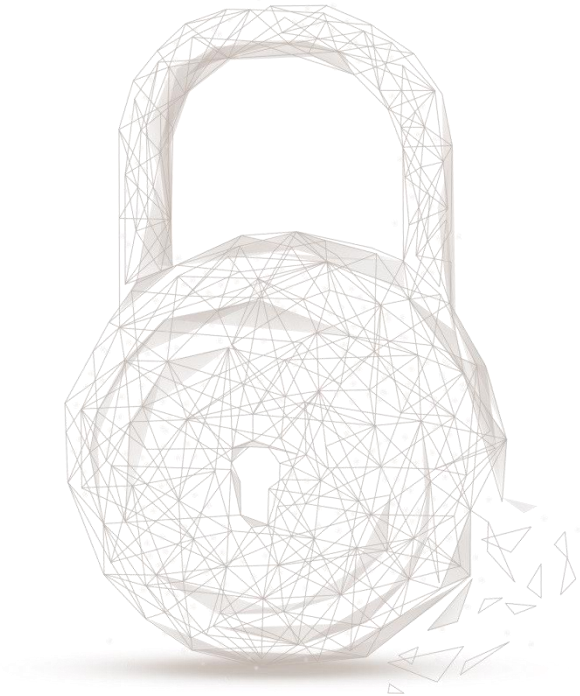
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202103081631

Smart Contract Info：

Smart Contract Name	Smart Contract Address	Smart Contract Address Link
HUSDLavaRewards2	0x55A718374aab282c43C46517a04bed17827dcf9b	https://hecoinfo.com/address/0x55A718374aab282c43C46517a04bed17827dcf9b#code
LavaFompRewards	0xEcD6967a0F523c335dd61A10558f10ACa0c590e0	https://hecoinfo.com/address/0xEcD6967a0F523c335dd61A10558f10ACa0c590e0#code
MXLavaRewards	0x8058d7f817e8701856686B139f6574E1A6D73992	https://hecoinfo.com/address/0x8058d7f817e8701856686B139f6574E1A6D73992#code
TPTFompRewards	0xFe1F1fA12135f5685CC72D09276cb78CC0ed3B70	https://hecoinfo.com/address/0xFe1F1fA12135f5685CC72D09276cb78CC0ed3B70#code

Start Date：2021.03.08

Completion Date：2021.03.08

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 HUSDLavaRewards2/LavaFompRewards/MXLavaRewards/TPTFompRewards, including Coding Standards, Security, and Business Logic. **The HUSDLavaRewards2/LavaFompRewards/MXLavaRewards/ TPTFompRewards 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, Four "stake rewards" smart contracts were implemented based on the same code structure, namely HUSDLavaRewards2, LavaFompRewards, MXLavaRewards and TPTFompRewards. except for the name and address, The HUSDLavaRewards2 contract is somewhat different from other contract logic. The following screenshots are based on LavaFompRewards and HUSDLavaRewards2.

3.1 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(LavaFompRewards)

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

- Result: Pass

3.2 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 (Greater than starttime); each time this function is called to *stake* tokens, the reward related data is updated through the modifier *updateReward*.

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

Figure 2 source code of stake function(1/2) (LavaFompRewards)

```
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) (LavaFompRewards)

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

3.3 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 (Greater than starttime); each time this function is called to withdraw tokens, the reward related data is updated through the modifier *updateReward*.

```
function withdraw(uint256 amount) public updateReward(msg.sender) checkStart{
    require(amount > 0, "Cannot withdraw 0");
    super.withdraw(amount);
    emit Withdrawn(msg.sender, amount);
}
```

Figure 4 source code of withdraw function(1/2) (LavaFompRewards)

```
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) (LavaFompRewards)

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

3.4 Withdraw rewards (fomp)

- Description: The contract implements the *getReward* function to withdraw the rewards (fomp). By calling the *transfer* function in the fomp contract, the contract address transfers the specified amount (all rewards of caller) of fomp to the user; This function restricts the user to call only after the "stake-reward" mode is turned on (Greater than starttime); each time this function is called to getreward, the reward related data is updated through the modifier *updateReward*.

```
function getReward() public updateReward(msg.sender) checkStart{
    uint256 reward = earned(msg.sender);
    if (reward > 0) {
        rewards[msg.sender] = 0;
        gene.safeTransfer(msg.sender, reward);
        emit RewardPaid(msg.sender, reward);
    }
}
```

Figure 6 source code of function getReward(LavaFompRewards)

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

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

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

Figure 7 source code of function exit(LavaFompRewards)

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

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

3.7 The logic is different

- Description: Add some logic to the HUSDLavaRewards2 contract, as shown in the figure. When a function caller calls a checkhalve-modified function, the first cycle will get all the rewards, the second cycle will be halved, and there will be no rewards later.

```
IERC20 public lava = IERC20(0x56f95662E71f30b333b456439248c6dE589082a4);

uint256 public constant DURATION = 1 weeks;
uint256 public halveTimes = 1;
uint256 public initreward = 25200*1e18;
uint256 public starttime = 1615208400; //utc+8 2021 03-08 21:00:00
uint256 public periodFinish = 0;
uint256 public rewardRate = 0;
uint256 public lastUpdateTime;
uint256 public rewardPerTokenStored;
mapping(address => uint256) public userRewardPerTokenPaid;
mapping(address => uint256) public rewards;
```

Figure 8 Code screenshot(HUSDLavaRewards2)

```
modifier checkhalve(){
    if (block.timestamp >= periodFinish) {
        if(halveTimes > 0){
            initreward = initreward.mul(50).div(100);
            rewardRate = initreward.div(DURATION);
            periodFinish = block.timestamp.add(DURATION);
            halveTimes = halveTimes.sub(1);
            emit RewardAdded(initreward);
        }
        else{
            rewardRate = 0;
        }
    }
    _;
}
```

Figure 9 modifier checkhalve screenshot(HUSDLavaRewards2)

- Result: Pass

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts HUSDLavaRewards2/LavaFompRewards/MXLavaRewards/TPTFompRewards. The HUSDLavaRewards2/LavaFompRewards/MXLavaRewards/TPTFompRewards 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