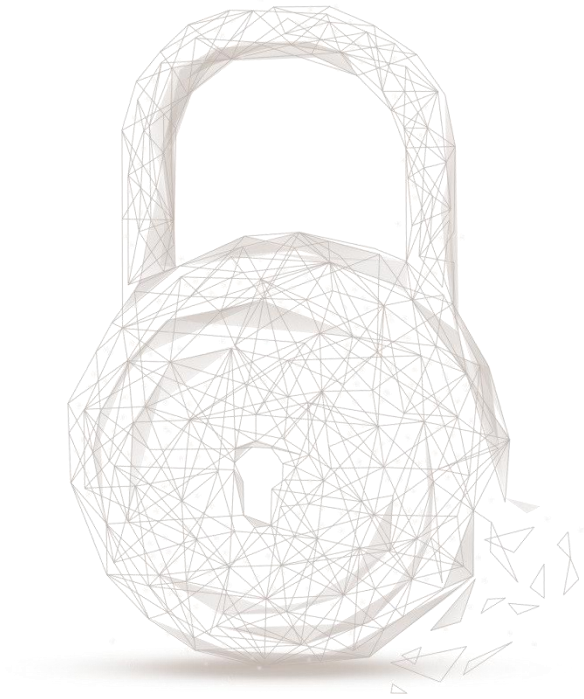
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202101221915

Report Query Name: DHM

Smart Contract Info：

Smart Contract Name	Smart Contract Address	Smart Contract Address Link
DHM	Fill in after deployment	Fill in after deployment
StakeDHM	Fill in after deployment	Fill in after deployment
StakingToken	Fill in after deployment	Fill in after deployment

Start Date：2021.01.20

Completion Date：2021.01.22

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 project DHM, including Coding Standards, Security, and Business Logic. **The DHM project 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 BNB.

- 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

Check whether the business is secure.

3.1 Business analysis of Contract DHM

(1) Basic Token Information

Token name	Fill in after deployment
Token symbol	Fill in after deployment
decimals	18
totalSupply	Fill in after deployment
Token type	BEP20

Table 1 Basic Token Information

(2) mint/burn function

- Description: As shown in figures below, the contract implements *mint*, *burn* and *burnFrom* functions to mint and burn tokens. An address with minter permission can call *mint* to mint a specified number of tokens to the DHM contract address (mintable with a cap, which is set by the project party during deployment). Any user can call the *burn/burnFrom* function to burn tokens.

```
121  function mint(uint256 amount_) public onlyMinter {
122      uint256 _cap = cap();
123      uint256 _supply = totalSupply();
124      if (paused() || _cap == _supply) {
125          return;
126      }
127      if (_cap.sub(_supply) < amount_) {
128          amount_ = _cap.sub(_supply);
129      }
130      _mint(address(this), amount_);
131      return;
132  }
```

Figure 1 Source Code of mint Function

```
180   function burn(uint256 amount) public {
181       _burn(_msgSender(), amount);
182   }
183
184   function burnFrom(address account, uint256 amount) public {
185       uint256 decreasedAllowance =
186           allowance(account, _msgSender()).sub(
187               amount,
188               "ERC20: burn amount exceeds allowance"
189           );
190
191       _approve(account, _msgSender(), decreasedAllowance);
192       _burn(account, amount);
193   }
```

Figure 2 Source Code of burn & burnFrom Function

- Related functions: *mint, burn, burnFrom, totalSupply*
- Result: Pass

(3) buy/recycle function

- Description: As shown in figures below, the contract implements the *buy* and *recycle* functions for users to buy and recycle DHM tokens. Any user can call the *buy* function after opening the sale(sell_price is greater than zero) to buy DHM tokens with USDT tokens or call the *recycle* function after opening the recycle(recycle_price is greater than zero) to recycle DHM tokens for USDT tokens (the selling price and the recycle price are arbitrarily set by the contract owner). When the user performs the recycle operation, half of the DHM tokens paid by the user will be burned.

```
135   function buy(uint256 amount_)
136       public
137       whenNotPaused
138       saleOpened
139       minBought(amount_)
140   {
141       // (usdt_amount / D6) / (amount_ / D18) == sell_price / price_decimals;
142       uint256 usdt_amount =
143           sell_price.mul(amount_).mul(D6).div(price_decimals).div(D18);
144       USDT.safeTransferFrom(msg.sender, address(this), usdt_amount);
145       IERC20(address(this)).safeTransfer(msg.sender, amount_);
146   }
```

Figure 3 Source Code of buy Function

```
149   function recycle(uint256 amount_, uint256 at_price)
150     public
151     whenNotPaused
152     recycleOpened
153     minBought(amount_)
154   {
155     uint256 r_price = calculate_recycle_price();
156     require(r_price == at_price, "DHM: recycling price has changed");
157
158     uint256 usdt_amount =
159       r_price.mul(amount_).mul(D6).div(price_decimals).div(D18);
160     USDT.safeTransfer(msg.sender, usdt_amount);
161
162     uint256 to_burn = amount_.div(2);
163     // uint256 to_recycle = amount_.sub(to_burn);
164     _transfer(msg.sender, address(this), amount_);
165     _burn(address(this), to_burn);
166   }
```

Figure 4 Source Code of recycle Function

- Related functions: *buy, recycle, safeTransferFrom, safeTransfer, calculate_recycle_price*
- Result: Pass

(4) evacuate_usdt/evacuate_eth function

- Description: As shown in the figure below, the contract implements the *evacuate_usdt* and *evacuate_eth* functions for the contract owner to withdraw the BNB and USDT tokens in the contract. The contract owner can call the *evacuate_usdt* function at any time to transfer the specified amount of USDT tokens in the contract to the specified address, and at the same time, call the *evacuate_eth* function at any time to transfer the BNB in the contract to the contract owner's address. Note: If the USDT tokens in the contract have been withdrawn, the user cannot call the *recycle* function to recycle DHM tokens.

```
168   function evacuate_usdt(address recv, uint256 amount_) public onlyOwner {
169     USDT.safeTransfer(recv, amount_);
170   }
171
172   function evacuate_eth() public onlyOwner {
173     payable(owner()).transfer(address(this).balance);
174   }
```

Figure 5 Source Code of evacuate_usdt& evacuate_eth Function

- Related functions: *evacuate_usdt, evacuate_eth, safeTransfer, transfer*
- Result: Pass

(5) update_usdt/update_wbtc/update_minter function

- Description: As shown in the figure below, the contract implements *update_usdt, update_wbtc* and *update_minter* functions for the contract owner to update related parameters. The contract owner can call these functions to update the variables USDT, WBTC and minter.

```
90  function update_usdt(address usdt_) public onlyOwner {
91      USDT = IERC20(usdt_);
92  }
93
94  function update_wbtc(address wbtc_) public onlyOwner {
95      WBTC = IERC20(wbtc_);
96  }
```

Figure 6 Source Code of update_usdt& update_wbtc Function

```
116  function update_minter(address u) public onlyOwner {
117      minter = u;
118      emit MinterChanged(u);
119  }
```

Figure 7 Source Code of evacuate_usdt& evacuate_eth Function

- Related functions: *update_usdt, update_wbtc, update_minter*
- Result: Pass

(6) update_sell_price/update_recycle_price function

- Description: As shown in the figure below, the contract implements the *update_sell_price* and *update_recycle_price* functions for the contract owner to update the DHM token buy price and recycle price. Note: Since the variable price_feed is not initialized and cannot be changed, the contract owner can change the selling and recycling price arbitrarily. If the owner's private key is stolen or hacked, the price may be set maliciously, causing loss of user assets.

```
98   function update_sell_price(uint256 price_) public onlyOwner {
99       if (price_feed != address(0)) {
100          sell_price = IPriceFeed(price_feed).get_sell_price();
101      } else {
102          sell_price = price_;
103      }
104      emit SellPriceChanged(sell_price);
105  }
106
107  function update_recycle_price(uint256 price_) public onlyOwner {
108      if (price_feed != address(0)) {
109          recycle_price = IPriceFeed(price_feed).get_recycle_price();
110      } else {
111          recycle_price = price_;
112      }
113      emit RecyclePriceChanged(recycle_price);
114  }
```

Figure 8 Source Code of update_sell_price& update_recycle_price Function

- Related functions: *update_sell_price, update_recycle_price*
- Result: Pass

3.2 Business analysis of Contract StakingToken

(1) Basic Token Information

Token name	Fill in after deployment
Token symbol	Fill in after deployment
decimals	18
totalSupply	Fill in after deployment
Token type	BEP20

Table 2 Basic Token Information

(2) mint/burn function

- Description: As shown in figures below, the contract implements *mint*, *burn* and *burnFrom* functions to mint and burn tokens. The contract owner or _mint_account address can call *mint* function to mint a specified number of tokens to himself(The current totalsupply cannot exceed the values of the variables _softcap and _cap). The contract owner or _mint_account address can call the *mint_and_lock* function

to mint a specified number of tokens to specified address. All tokens mint by calling the *mint_and_lock* function will be locked and will be unlocked over time. Any user can call the *burn/burnFrom* function to burn tokens.

```
241    function mint(uint256 amount)
242        public
243        virtual
244        nonReentrant
245        mint_auth_required
246        returns (bool)
247    {
248        _mint(_msgSender(), amount);
249        return true;
250    }
```

Figure 9 Source Code of mint Function

```
560    function mint_and_lock(
561        address to_whom,
562        uint256 amount,
563        uint256 lockspan,
564        uint256 frozen_hell
565    ) public mint_auth_required nonReentrant {
566        require(llocks[to_whom].remains_in_lock == 0);
567
568        LinearLockWithFrozenHell storage lk = llocks[to_whom];
569
570        lk.total_amount = amount;
571        lk.lock_span = lockspan;
572        lk.frozen_hell = frozen_hell;
573        lk.created_timestamp = block.timestamp;
574        lk.remains_in_lock = amount;
575        lk.latest_claim = block.timestamp;
576
577        _mint(address(this), amount);
578        _transferToReserved(address(this), to_whom, amount);
579    }
```

Figure 10 Source Code of mint_and_lock Function

```
252  function burn(uint256 amount) public virtual nonReentrant returns (bool) {
253      _burn(_msgSender(), amount);
254      return true;
255  }
256
257  function burnFrom(address account, uint256 amount)
258      public
259      virtual
260      nonReentrant
261  {
262      uint256 decreasedAllowance =
263          allowance(account, _msgSender()).sub(
264              amount,
265              "ERC20: burn amount exceeds allowance"
266          );
267
268      _approve(account, _msgSender(), decreasedAllowance);
269      _burn(account, amount);
270  }
```

Figure 11 Source Code of burn & burnFrom Function

- Related functions: *mint, burn, burnFrom, totalSupply*
- Result: Pass

(3) claim function

- Description: As shown in the figure below, the contract implements the *claim* function for the user to withdraw the released tokens. The locked tokens will be gradually unlocked over time and users can query how many tokens they can unlock by calling the *can_claim* function.

```
606   function claim() public nonReentrant isAlive {
607       require(
608           _reserved_balances[_msgSender()] > 0,
609           "StakingToken::claim: sender has no reserved balance"
610       );
611       require(
612           llocks[_msgSender()].remains_in_lock > 0,
613           "StakingToken::claim: sender has no locks"
614       );
615
616       LinearLockWithFrozenHell storage llwf = llocks[_msgSender()];
617
618       uint256 begins = _releaseBegins(_msgSender());
619       require(
620           block.timestamp > begins,
621           "StakingToken::claim: release has not begin yet"
622       );
623
624       uint256 released_span = block.timestamp - begins;
625       if (llwf.latest_claim > begins) {
626           released_span = block.timestamp - llwf.latest_claim;
627       }
628       uint256 released_amount =
629           llwf.total_amount.div(llwf.lock_span).mul(released_span);
630
631       if (llwf.remains_in_lock <= released_amount) {
632           released_amount = llwf.remains_in_lock;
633           llwf.remains_in_lock = 0;
634       } else {
635           llwf.remains_in_lock = llwf.remains_in_lock.sub(released_amount);
636       }
637
638       llwf.latest_claim = block.timestamp;
639
640       _unreserve(_msgSender(), released_amount);
641   }
```

Figure 12 Source Code of claim Function

- Related functions: *claim, _releaseBegins*

- Result: Pass

(4) delegate function

- Description: .As shown in the figure below, the contract implements the *delegate* function to delegate. The user can call those functions to delegate. The function *delegate* updates the delegate information by calling internal functions *_delegate*, *_moveDelegates* and *_writeCheckpoint*.

```
ftrace | funcSig
490   function delegate(address delegatee) public nonReentrant {
491     return _delegate(msg.sender, delegatee);
492   }
```

Figure 13 Source Code of delegate Function

- Related functions: *delegate*
- Result: Pass

(5) reserve_from/unreserve_from function

- Description: As shown in the figure below, the contract implements *reserve_from* and *unreserve_from* functions for _mint_account and the contract owner to lock and unlock tokens. Note: The above address can be arbitrarily locked and unlocked for any user's tokens.

```
406   function reserve_from(address who, uint256 amount)
407     public
408     mint_auth_required
409     nonReentrant
410     returns (bool)
411   {
412     return _reserve(who, amount);
413   }
414
415   function unreserve_from(address who, uint256 amount)
416     public
417     mint_auth_required
418     nonReentrant
419     returns (bool)
420   {
421     return _unreserve(who, amount);
422   }
```

Figure 14 Source Code of reserve_from& unreserve_from Function

- Related functions: *reserve_from, unreserve_from function*
- Result: Pass

3.2 Business analysis of Contract StakeDHM

(1) stake/stake_for function

- Description: As shown in the figure below, the contract implements the *stake* and *stake_for* functions for users to stake. Any user calls the *stake* or *stake_for* function (the beneficiary may not be the caller) to stake DHM tokens to obtain WBTC token rewards. The DHM tokens stake in this epoch

will start to calculate rewards at the beginning of the next epoch. If the user stakes for the second time, the reward generated by the last stake will be automatically received.

```
222    // a fresh stake will make all former unclaimed reward claimed automatically
223    function stake(uint256 amount_) public whenNotPaused {
224        _stake_(msg.sender, msg.sender, amount_);
225        emit Stake(msg.sender, amount_);
226    }
227
228    // allow some delegates to *STAKE* for others
229    // the tokens which will be staked will come from msg.sender's account
230    function stake_for(address u_, uint256 amount_) public whenNotPaused {
231        _stake_(msg.sender, u_, amount_);
232        emit Stake(u_, amount_);
233    }
```

Figure 15 Source Code of stake& stake_for Function

- Related functions: *stake, stake_for, safeTransferFrom, is_epoch_reported, safeTransfer, totalSupply, mint, cap, balanceOf, cached_epoch_reward*

- Result: Pass

(2) withdraw function

- Description: .As shown in the figure below, the contract implements the *withdraw* function for users to withdraw stake tokens. Any user calls the *withdraw* function to withdraw the stake DHM tokens and receive WBTC token rewards.

```
279    function withdraw() public whenNotPaused {
280        (uint256 amount, uint256 rewards) = _withdraw_(msg.sender);
281        emit Withdraw(msg.sender, amount, rewards);
282    }
```

Figure 16 Source Code of withdraw Function

- Related functions: *withdraw, safeTransferFrom, is_epoch_reported, safeTransfer, totalSupply, mint, cap, balanceOf, cached_epoch_reward*

- Result: Pass

(3) claim function

- Description: As shown in the figure below, the contract implements the *claim* function for the withdrawal of user stake reward tokens. Any user can call this function to receive rewards for the specified epoch. If the epoch entered by the user is 0, all the rewards so far will be received.

```
309  function claim(uint256 epochs) public whenNotPaused {
310      uint256 current_epoch = _to_epoch(block.timestamp);
311      _snapshot(current_epoch);
312
313      uint256 collected;
314      uint256 cached;
315      if (epochs == 0) {
316          (cached, collected) = _claim(msg.sender, current_epoch);
317      } else {
318          uint256 last_claim_epoch =
319              _user_last_claim_epoch(msg.sender, current_epoch);
320
321          // claim from [last_claim_epoch +1, last_claim_epoch + epochs]
322          (cached, collected) = _claim(msg.sender, last_claim_epoch + epochs);
323      }
324
325      _update_seen_epoch(current_epoch);
326
327      if (cached + collected > 0) {
328          emit Claim(msg.sender, cached + collected);
329      }
330  }
```

Figure 17 Source Code of claim Function

- Related functions: *claim, cached_epoch_reward, is_epoch_reported, safeTransfer*

- Result: Pass

(4) evacuate_reward/evacuate_stake/evacuate_eth function

- Description: As shown in the figure below, the contract implements the *evacuate_reward, evacuate_eth*, and *evacuate_stake* functions for the contract owner to withdraw the BNB, DHM and WBTC tokens in the contract. The contract owner can call the *evacuate_eth* function to transfer the BNB in the contract to himself, call the *evacuate_reward* function to transfer WBTC tokens to the specified address, and call the *evacuate_stake* function to return the stake tokens that the user cannot withdraw to the user address.

```
156  function evacuate_reward(address u, uint256 amount) public onlyOwner {
157      IERC20(reward_token).safeTransfer(u, amount);
158  }
159
160  function evacuate_eth() public onlyOwner {
161      payable(owner()).transfer(address(this).balance);
162  }
163
164  // NOTE: this only serves as a last resort for rescuing user's stakes,
165  // and only works when paused
166  function evacuate_stake(address u) public onlyOwner whenPaused {
167      uint256 amt = stakes[u];
168      delete (stakes[u]);
169      IERC20(stake_token).safeTransfer(u, amt);
170  }
```

Figure 18 Source Code of evacuate_reward, evacuate_eth& evacuate_stake Function

- Related functions: *evacuate_reward, evacuate_eth, evacuate_stake, safeTransfer, transfer*
- Modify Recommendation: It is recommended to update the stake information when returning the user's token.
- Fixed Result: Not fixed. The project team believes that after using this function, no other functions will be used again.
- Result: Pass

(5) set_reward_token/set_reward_reporter function

- Description: As shown in the figure below, the contract implements the *set_reward_token* and *set_reward_reporter* functions for the contract owner to change the reward token address and reward_reporter.

```
188  function set_reward_token(address token) public onlyOwner {
189      reward_token = token;
190  }
191
192  function set_reward_reporter(address r) public onlyOwner {
193      reward_reporter = r;
194  }
```

Figure 19 Source Code of set_reward_token& set_reward_reporter Function

- Related functions: *set_reward_token, set_reward_reporter*
- Result: Pass

(6) set_epoch_length/reset_epochs function

- Description: As shown in the figure below, the contract implements the *set_epoch_length* and *reset_epochs* functions for the contract owner to change epoch related information.

```
196  function set_epoch_length(uint256 l) public onlyOwner whenPaused {
197      delete (stakes_epoch_snapshots[default_epoch]);
198
199      epoch_length = l;
200      // renew default epoch
201      default_epoch = _to_epoch(block.timestamp);
202
203      stakes_epoch_snapshots[default_epoch] = StakesSnapshot({
204          valid: true,
205          amount: 0
206      });
207  }
208
209  function reset_epochs() public onlyOwner whenPaused {
210      default_epoch = _to_epoch(block.timestamp);
211      // safe zone
212      delete (stakes_epoch_snapshots[default_epoch - 1]);
213      delete (stakes_epoch_snapshots[default_epoch - 2]);
214      delete (stakes_epoch_snapshots[default_epoch - 3]);
215      // safe zone
216      stakes_epoch_snapshots[default_epoch] = StakesSnapshot({
217          valid: true,
218          amount: 0
219      });
220  }
```

Figure 20 Source Code of set_epoch_length& reset_epochs Function

- Related functions: *set_epoch_length, reset_epochs*
- Result: Pass

(7) snapshot function

- Description: As shown in the figure below, the contract implements the *snapshot* function for the contract owner to repair the lost epoch snapshot information.

```
347  function snapshot(uint256 epoch_) public onlyOwner {
348      uint256 epoch_to_snapshot = epoch_;
349      if (epoch_ == 0) {
350          epoch_to_snapshot = _to_epoch(block.timestamp);
351      }
352      _snapshot(epoch_to_snapshot);
353  }
```

Figure 21 Source Code of snapshot Function

- Related functions: *snapshot*
- Result: Pass

(8) report_timestamp_reward function

- Description: As shown in the figure below, the contract implements the *report_timestamp_reward* function for reward_reporter to set or modify epoch rewards. According to the project party's explanation: the revenue is calculated off-chain, and the publicly displayed quantity is the actual output of the explosion block, not the reward that the user will eventually get. The actual reward will also deduct the electricity cost of the day. In order to reduce the two sides' loss and the project cannot be corrected, the *report_timestamp_reward* function also provides a modified reward.

```
377  function report_timestamp_reward(
378      uint256 timestamp_,
379      uint256 data,
380      bool override_
381  ) public onlyReporter {
382      uint256 epc = _to_epoch(timestamp_);
383      require(
384          epc >= default_epoch,
385          "StakeDHM: epoch should be greater than default"
386      );
387
388      // TODO: we might want to disable the future telling
389
390      if (!rewards_each_epoch[epc].valid) {
391          rewards_each_epoch[epc] = EpochReward({valid: true, reward: data});
392          total_rewarded = total_rewarded.add(data);
393      } else {
394          require(override_, "StakeDHM: report existed");
395
396          // TODO: check if new data is valid,
397          // replace with the new data
398          rewards_each_epoch[epc] = EpochReward({valid: true, reward: data});
399      }
400
401      if (epc > latest_reported_epoch) {
402          latest_reported_epoch = epc;
403      }
404
405      _update_seen_epoch(epc);
406  }
```

Figure 22 Source Code of report_timestamp_reward Function

- Related functions: *report_timestamp_reward*
- Result: Pass

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts project DHM. All issues have been notified to the project party, and the project party has ignored some of the amendments. The overall audit result of the smart contract project DHM is Pass.

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com