5th AUG 2021

SMART CONTRACT AUDIT REPORT

-

version v1.0

Smart Contract Security Audit and General Analysis

HAECHI AUDIT

Table of Contents

1 Issues (0 Critical, 0 Major, 1 Minor) Found

About HAECHI AUDIT

HAECHI AUDIT is a flagship service of HAECHI LABS, the leader of the global blockchain industry. HAECHI AUDIT provides specialized and professional smart contract security auditing and development services.

We are experts with years of experience in the research and development of blockchain technology. We are the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Cryptocurrency exchanges worldwide trust HAECHI AUDIT's security audit reports. Indeed, numerous clients have successfully listed on Huobi, OKEX, Upbit, and Bithumb, etc. after passing HAECHI AUDIT smart contract security audit.

Representative clients and partners include the Global Blockchain Project and Fortune Global 500 corporations, in addition to Ground X (Kakao Corp. subsidiary), Carry Protocol, Metadium, LG, Hanwha, and Shinhan Bank. Over 60 clients have benefited from our most reliable smart contract security audit and development services.

Inquiries : audit@haechi.io
Website : audit.haechi.io

01. Introduction

This report aims to audit the security of Decubate smart contract created by Decubate team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by Decubate team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the smart contract.

The issues discovered are categorized into CRITICAL, MAJOR, MINOR, TIPS according to their importance.

CRITICAL	Critical issues must be resolved as critical flaws that can harm a wide range of users.
MAJOR	Major issues require correction because they either have security problems or are implemented not as intended.
MINOR	Minor issues can potentially cause problems and therefore require correction.
TIPS	Tips issues can improve the code usability or efficiency when corrected.

HAECHI AUDIT recommends that Decubate team improves all issues discovered.

The following issue explanation uses the format of {file name}#{line number}, {contract name}#{function/variable name} to specify the code. For instance, Sample.sol:20 points to the 20th line of Sample.sol file, and Sample#fallback() means the fallback() function of the Sample contract.

Please refer to the Appendix to check all results of the tests conducted for this report.

02. Summary

The codes used in this Audit can be found on GitHub (https://github.com/Decubate-com/smart-contracts/tree/5fb73a3fb17a8c37e39ea8513c

4556ec2b1b64d6). The last commit for the code audited is "5fb73a3fb17a8c37e39ea8513c4556ec2b1b64d6".

Issues

HAECHI AUDIT identified 0 critical issue, 0 major issues, and 1 minor issues. There is 1 Tips issue explained for improving the code's usability and efficiency.

Severity	Issue	Status
MINOR	Unexpected revert may occur because equal sign is not included.	(Found - v1.0)
TIPS	There is a missing Event	(Found - v1.0)

03. Overview

Contracts Subject to Audit

- DCBToken
- DecubateCrowdFunding
- DecubateVesting

Key Features

Decubate team implemented ERC20 Smart contract that performs the following functions.

- Crowd Funding
- Token Vesting

Roles

Decubate Smart contract has the following functions.

- **Owner**

The specification for the control of each privilege is as follows.

Role	MAX	Addable	Deletable	Transferable	Renouncable
Owner	1	X	X	O	O

Each privilege can access the following functions.

Role	Functions
Owner	*DCBToken#lockTokens()* *DCBToken#burn()* *DecubateCrowdfunding#setRequiredVote()* *DecubateCrowdfunding#setDecubateToken()* *DecubateCrowdfunding#addDCBAgreement()* *DecubateCrowdfunding#addTokenToDCB()* *DecubateCrowdfunding#removeTokenFromDCB()* *DecubateCrowdfunding#transferToken()* *DecubateVesting#addInitialUnlock()* *DecubateVesting#claimInitialUnlock()* *DecubateVesting#setVestingInfo()* *DecubateVesting#addWhitelist()* *DecubateVesting#setDecubateToken()* *DecubateVesting#revoke()* *DecubateVesting#disable()* *DecubateVesting#transferDCB()*

04. Issues Found

MINOR : Unexpected revert may occur because equal sign is not included. (Found - v.1.0)

MINOR

```
427             /** make sure if project has enough vote */
428             require(
429                 dcbAgreement.vote > _requiredVoteNumber,
430                 "agreement should have enough vote"
431             );
```

[https://github.com/Decubate-com/smart-contracts/blob/5fb73a3fb17a8c37e39ea8513c4556ec2b1b64d6/contracts/DecubateCrowdfunding.sol#L429]

```
433         /** check if treasury have enough funds to withdraw to innovator */
434         require(
435             tokenPools[_token].token.balanceOf(address(this)) >
436                 dcbAgreement.totalInvestFund,
437             "need to have enough funds in treasury"
438         );
```

[https://github.com/Decubate-com/smart-contracts/blob/5fb73a3fb17a8c37e39ea8513c4556ec2b1b64d6/contracts/DecubateCrowdfunding.sol#L435]

```
462     function transferToken(
463         address _token,
464         uint256 _amount,
465         address _to
466     ) external override onlyOwner requireToken(_token) returns (bool) {
467         /** check if treasury have enough funds  */
468         require(
469             tokenPools[_token].token.balanceOf(address(this)) > _amount,
470             "need to have enough funds in treasury"
471         );
472         tokenPools[_token].token.transfer(_to, _amount);
473 
474         emit TransferFund(_token, _amount, _to);
475         return true;
476     }
```

[https://github.com/Decubate-com/smart-contracts/blob/5fb73a3fb17a8c37e39ea8513c4556ec2b1b64d6/contracts/DecubateCrowdfunding.sol#L469]

DecubateCrowdfunding#claimInnovatorFund(), *DecubateCrowdfunding#transferToken()* does not include equal signs in its require() functions. However, the function should be executed even when two arguments are equal.

Recommendation

We recommend adding an equal sign in the function.

TIPS : There is a missing Event. (Found - v.1.0)

TIPS

The following list shows functions with a missing Event.

Function	Expected Event	Emitted Event	Omitted Event
burn	Transfer, Burn	Transfer	Burn

When an Event is missing, it is difficult to identify in real-time whether an accurate value is recorded on the blockchain. In this case, it is difficult to determine whether the corresponding value has been changed and whether the corresponding function has been called in the application.

Thus, we recommend adding an Event that corresponds to the change occurring in the pertinent function.

05. Disclaimer

This report does not guarantee investment advice, suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects in Ethereum and Solidity. In order to write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix A. Test Results

The following results are unit test results that cover the key logic of the smart contract subject to the security audit. Parts marked in red are test cases that failed to pass the test due to having issues.

```
DecubateCrowdfunding
  #constructor()
    ✓ required vote number set to 100 (58ms)
    ✓ decubate token is not set yet
    ✓ no one participates
  #addDCBAgreement()
    ✓ should fail if msg.sender is not owner (70ms)
    ✓ should fail if hardcap is zero
    ✓ should fail if tr to set agreement end time to past
    ✓ should fail if agreement already exist (58ms)
   valid case
     ✓ generate the new agreement
     ✓ should emit CreateAgreement event
  #addTokenToDCB()
    ✓ should fail if msg.sender is not owner
    ✓ should fail if token already exist
   valid case
     ✓ add new token to the token pool
     ✓ should emit AddToken event
  #removeTokenFromDCB()
    ✓ should fail if token is not exist in the pool
    ✓ should fail if msg.sender is not owner
   valid case
     ✓ pool status set to false
     ✓ should emit RemoveToken event
  #joinDCBAgreement()
    ✓ should fail if msg.sender does not have DCB
    ✓ should fail if token is not in the token pool
    ✓ should fail if user already joined agreement pool
    ✓ should fail if try to invest 0 (96ms)
    ✓ should fail if msg.sender does not have enough token to invest (129ms)
   valid case
     ✓ contract earn tokens
     ✓ should emit InvestorJoin event
  #voteDCBAgreement()
```

```
    ✓ should fail if msg.sender does not have DCB
    ✓ should fail if already voted (51ms)
    valid case
      ✓ vote count should update
  #claimInnovatorFund()
    ✓ should fail if token is not active
    ✓ should fail if msg.sender does not have DCB
    ✓ should fail if agreement does not have enough vote
    ✓ should fail if contract does not have enough token (72ms)
    valid case
      ✓ innovator waller earn token
      ✓ should emit ClaimFund event
  #fallback()
    ✓ revert fallback function

DecubateVesting
  #addInitialUnlock()
    ✓ should fail if msg.sender is not owner
    ✓ should fail if VIP already exist
    ✓ vipPools should be set properly
  #claimInitialUnlock()
    ✓ should fail if msg.sender is not owner
    ✓ should fail if wallet does not have an initial unlock to claim
    ✓ should fail if unlockAmount is zero (70ms)
    valid case
      ✓ unlockAmount should be zero
      ✓ wallet's decubateToken balance should increase
  #setVestingInfo()
    ✓ should fail if msg.sender is not owner
    ✓ should fail if _strategy is zero
    ✓ should fail if vesting option is already existing
    ✓ vesting info should be set properly
  #getVestingInfo()
    ✓ should fail if vesting option is not existing
    ✓ should return vesting info properly
  #addWhitelist()
    ✓ should fail if msg.sender is not owner
    ✓ should fail if vesting option is not existing
    ✓ should fail if whitelist is already available (47ms)
    valid case
      ✓ whitelist should be set properly (46ms)
      ✓ should emit AddWhitelist event (38ms)
```

```
  #getWhitelist()
    ✓ should fail if vesting option is not existing
    ✓ should fail if whitelist is not existing
    ✓ should return whitelist properly (44ms)
  #setDecubateToken()
    ✓ should fail if msg.sender is not owner
    ✓ _decubateToken should be set properly
  #calculateVestAmount()
    ✓ should fail if vesting option is not existing
    ✓ should fail if user in not in whitelist
    valid case
      ✓ should return zero if vesting option's cliff is greater than now (51ms)
      ✓ should return dcbAmount if wallet is revoked (67ms)
      ✓ should return dcbAmount if now is greater than vesting option's start + duration (42ms)
      ✓ else, should return properly (47ms)
  #calculateReleasableAmount()
    ✓ should fail if vesting option is not existing
    ✓ should fail if user in not in whitelist
    ✓ should return the amount obtained by subtracting distributedAmount from calculatedVestAmount
(108ms)
  #claimDistribution()
    ✓ should fail if vesting option is not existing
    ✓ should fail if user is not in whitelist (39ms)
    ✓ should fail if user is disabled from claiming token (72ms)
    ✓ should fail if releaseAmount is zero (57ms)
    valid case
      ✓ wallet's decubateToken balance should increase
      ✓ wallet's distributedAmount should increase
  #revoke()
    ✓ should fail if msg.sender is not owner
    ✓ should fail if vesting option is not existing
    ✓ should fail if user is not in whitelist
    ✓ should fail if vesting option is not revocable (43ms)
    ✓ should fail if user is already revoked (56ms)
    valid case
      ✓ Wallet should be revoked (64ms)
      ✓ should emit Disabled event (105ms)
  #disable()
    ✓ should fail if msg.sender is not owner
    ✓ should fail if vesting option does not exist
    ✓ should fail if user is not in whitelist
    ✓ should fail if user is already disabled (65ms)
```

```
    valid case
      ✓ Wallet should be disabled (62ms)
      ✓ should emit Disabled event (62ms)
  #transferDCB()
    ✓ should fail if msg.sender is not owner
    ✓ should increase owner's DCB balance
  #getTotalToken()
    ✓ should return totalToken properly
  #hasWhitelist()
    ✓ should fail if vesting option is not existing
    ✓ should retrun properly (45ms)

DCBToken
  #constructor()
    ✓ should set name properly
    ✓ should set symbol properly
    ✓ should set decimals properly
    ✓ should set initial supply properly
  ERC20 Spec
    #transfer()
      ✓ should fail if recipient is ZERO_ADDRESS
      ✓ should fail if sender's amount is lower than balance
      when succeeded
        ✓ sender's balance should decrease
        ✓ recipient's balance should increase
        ✓ should emit Transfer event
    #transferFrom()
      ✓ should fail if sender is ZERO_ADDRESS
      ✓ should fail if recipient is ZERO_ADDRESS
      ✓ should fail if sender's amount is lower than transfer amount
      ✓ should fail if allowance is lower than transfer amount
      ✓ should fail even if try to transfer sender's token without approve process
      when succeeded
        ✓ sender's balance should decrease
        ✓ recipient's balance should increase
        ✓ should emit Transfer event
        ✓ allowance should decrease
        ✓ should emit Approval event
    #approve()
      ✓ should fail if spender is ZERO_ADDRESS
      valid case
        ✓ allowance should set appropriately
```

```
    ✓ should emit Approval event
ERC20Burnable spec
  #burn()
    ✓ should fail if try to burn more than burner's balance
    valid case
      ✓ totalSupply should decrease
      ✓ account's balance should decrease
      ✓ should emit Transfer event
      2) should emit Burn event
DCBToken spec
  #lockTokens()
    ✓ should fail if msg.sender is not owner
    ✓ should fail if try to lock more than owner's balance
    valid case
      ✓ owner's balance should decrease
      ✓ should emit Transfer event
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/					
DCBToken.sol	**100**	**100**	**100**	**100**	
DecubateCrowdFunding.sol	**100**	**100**	**100**	**100**	
DecubateVesting.sol	**100**	**100**	**100**	**100**	

[Table 1] Test Case Coverage