21th AUGUST 2020

SMART CONTRACT AUDIT REPORT

version 1.0

KCT Security Audit and General Analysis

HAECHI AUDIT

Table of Contents

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

About HAECHI AUDIT

HAECHI AUDIT is a global leading smart contract security audit and development firm operated by HAECHI LABS. HAECHI AUDIT consists of professionals with years of experience in blockchain R&D and provides the most reliable smart contract security audit and development services.

So far, based on the HAECHI AUDIT's security audit report, our clients have been successfully listed on the global cryptocurrency exchanges such as Huobi, Upbit, OKEX, and others.

Our notable portfolios include SK Telecom, Ground X by Kakao, and Carry Protocol while HAECHI AUDIT has conducted security audits for the world's top projects and enterprises.

Trusted by the industry leaders, we have been incubated by Samsung Electronics and awarded the Ethereum Foundation Grants and Ethereum Community Fund.

Contact : audit@haechi.io
Website : audit.haechi.io

01. Introduction

This report was written to provide a security audit for the Pibble smart contract. HAECHI AUDIT conducted the audit focusing on whether Pibble smart contract is designed and implemented in accordance with publicly released information and whether it has any security vulnerabilities.

The issues found are classified as CRITICAL, MAJOR, MINOR or TIPS according to their severity.

CRITICAL	Critical issues are security vulnerabilities that MUST be addressed in order to prevent widespread and massive damage.
MAJOR	Major issues contain security vulnerabilities or have faulty implementation issues and need to be fixed.
MINOR	Minor issues are some potential risks that require some degree of modification.
TIPS	Tips could help improve the code's usability and efficiency

HAECHI AUDIT advises addressing all the issues found in this report.

02. Summary

The code used for the audit can be found at BitBucket(https://bitbucket.org/dooba202/pibble-klaytn/src/master/src/SmartContract/Klaytn/PIBwFreeze.sol). The last commit for the code audited is at "8ad3dea8b8d189c03362d8b2c44220934840ee07".

Issues

HAECHI AUDIT has 0 Critical Issues, 0 Major Issues, and 0 Minor Issue; also, we included 1 Tip category that would improve the usability and/or efficiency of the code.

Severity	Issue	Status
TIPS	We recommend you to add an Event	**(Found - v1.0)**

03. Overview

Contracts Subject to Audit

- PibbleToken
- MintableToken
- StandardToken
- BasicToken
- Owanble

Key Features

Team Pibble has implemented an KCT Smart contract with the following features:

- MInt
- Freeze

Roles

The Pibble Smart contract has the following authorizations:

- **Owner**

Each level of authorization has these respective characteristics.

Role	MAX	Addable	Deletable	Transferable	Renouncable
Owner	1	X	X	by **Owner**	X

The features accessible by each level of authorization is as follows:

Role	Functions
Owner	*Ownable#transferOwnership()* *MintableToken#mint()* *MintableToken#finishMinting()* *PibbleToken#freezeAccount()*

04. Issues Found

TIPS : We recommend you to add an Event (Found - v.1.0)

TIPS

When running *PibbleToken#burn()*, the balance will be updated but a Transfer Event will not occur. A change in balance due to the function may not be detected if balance updates are detected upon Transfer Events. Thus, it is recommended that you add the Transfer Event to your function.

05. Disclaimer

This report is not an advice on investment, nor does it guarantee adequacy of a business model and/or a bug-free code. This report should be used only to discuss known technical problems. The code may include problems on Ethereum and/or Solidity that are not included in this report. It will be necessary to resolve addressed issues and conduct thorough tests to ensure the safety of the smart contract.

Appendix A. Test Results

The following are the results of a unit test that covers the major logics of the smart contract under audit. The parts in red contain issues and therefore have failed the test.

```
Contract: Ownable
  #constructor()
    ✓ should set owner to contract initializer (41ms)
  #owner()
    ✓ should return appropriate owner (38ms)
  #transferOwnership()
    ✓ should fail if msg.sender is not owner (92ms)
    ✓ should fail if newOwner is ZERO_ADDRESS (89ms)
    valid case
      ✓ should emit OwnershipTransferred event
      ✓ should set owner to newOwner

Contract: SafeMath
  #add()
    ✓ adds correctly (121ms)
    ✓ reverts on addition overflow (70ms)
  #sub()
    ✓ subtracts correctly
    ✓ reverts if subtraction result would be negative
  #mul()
    ✓ multiplies correctly (65ms)
    ✓ multiplies by zero correctly (71ms)
    ✓ reverts on multiplication overflow (67ms)
  #div()
    ✓ divides correctly
    ✓ divides zero correctly
    ✓ returns complete number result on non-even division
    ✓ reverts on divison by zero

Contract: PibbleToken
  #constructor()
    ✓ contract caller set to owner
    ✓ contract initializer's balance set to initial supply
    ✓ name, symbol, decimals set properly (83ms)
  ERC20 Spec
    #transfer()
```

```
    ✓ should fail if recipient is ZERO_ADDRESS (75ms)
    ✓ should fail if sender's amount is lower than balance (271ms)
    ✓ should fail if sender's account is frozen (148ms)
    ✓ should fail if recipient's account is frozen (284ms)
    when succeeded
      ✓ sender's balance should decrease (38ms)
      ✓ recipient's balance should increase (88ms)
      ✓ should emit Transfer event
  #transferFrom()
    ✓ should fail if sender is ZERO_ADDRESS (80ms)
    ✓ should fail if recipient is ZERO_ADDRESS (208ms)
    ✓ should fail if sender's amount is lower than transfer amount (169ms)
    ✓ should fail if sender's account is frozen (195ms)
    ✓ should fail if recipient's account is frozen (193ms)
    ✓ should fail if allowance is lower than transfer amount (123ms)
    ✓ should fail even if try to transfer sender's token without approve process (88ms)
    when succeeded
      ✓ sender's balance should decrease
      ✓ recipient's balance should increase
      ✓ should emit Transfer event
      ✓ allowance should decrease
  #approve()
    valid case
      ✓ allowance should set appropriately
      ✓ should emit Approval event
  #increaseApproval()
    ✓ should fail if overflows (322ms)
    valid case
      ✓ allowance should set appropriately (38ms)
      ✓ should emit Approval event
  #decreaseApproval()
    ✓ should set allowance to zero if overflows (93ms)
    valid case
      ✓ allowance should set appropriately
      ✓ should emit Approval event
ERC20Mintable Spec
  #mint()
    ✓ should fail if msg.sender is not owner (47ms)
    ✓ should fail if overflows (44ms)
    ✓ should fail if mint is finished (100ms)
    valid case
      ✓ receiver's amount should increase
```

```
    ✓ totalSupply should increase
    ✓ should emit Transfer event
    ✓ should emit Mint event
  #finishMinting()
    ✓ should fail if msg.sender is not owner (49ms)
    ✓ should fail if already finished (102ms)
    ✓ should set _mintingFinished to true (72ms)
  ERC20Burnable Spec
  #burn()
    ✓ should fail if overflows (47ms)
    valid case
    ✓ totalSupply should decrease
    ✓ account's balance should decrease
    1) should emit Transfer event
    ✓ should emit Burn event
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/					
PibbleToken.sol	**100**	**100**	**100**	**100**	

[Table 1] Test Case Coverage