15.06.2021

GOLDPEGAS

SMART CONTRACT AUDIT REPORT

-

version v1.0

BEB20 Security Audit and General Analysis

HAECHI AUDIT

Table of Contents

2 Issues (0 Critical, 2 Major, 0 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.

01. Introduction

This report aims to audit the security of GOLDPEGASTOKEN smart contract created by GOLDPEGAS team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by GOLDPEGAS team is soundly implemented and designed as specified in the published materials as well as whether the smart contract is secure in terms of security.

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 thus require correction.

TIPS Tips issues are those that can improve the usability or efficiency of the code when corrected.

HAECHI AUDIT recommends GOLDPEGAS team to improve all issues discovered.

The following issue description 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 at Github (https://github.com/goldpegasio/contracts/blob/1f236ced18589ff865dff329dee53e2e3ec9c99d/GOLDPEGAS.sol). The last commit of the code used in this Audit is "1f236ced18589ff865dff329dee53e2e3ec9c99d".

Issues

HAECHI AUDIT found 0 critical issues, 2 major issues, and 0 minor issues. There was 1 issue categorized as Tips as those that can improve the code's usability and efficiency.

Severity	Issue	Status
MAJOR	Owner can restrict token transfers of other accounts	(Found - v1.0)
MAJOR	Token may be released to an address that is not a contract	(Found - v1.0)
TIPS	There are missing Events	(Found - v1.0)

03. Overview

Contracts Subject to Audit

- TokenAuth
- GOLDPEGASTOKEN
- SafeMath
- Context

Key Features

GOLDPEGAS team implemented the BEB20 Smart Contract that performs the following functions.

- Restricting account transfer (Lock)
- Token release (Release Allocation)

Roles

GOLDPEGASTOKEN Smart contract has the following authority.

- **Owner**

The details of the control of each authority are as follows.

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

Each authority can access the following functions.

Role	Functions
Owner	*TokenAuth#setFarmAddress()* *TokenAuth#_transferOwnership()* *GOLDPEGASTOKEN#releaseAirdropAllocation()* *GOLDPEGASTOKEN#releaseFarmAllocation()* *GOLDPEGASTOKEN#releaseLiquidityPoolAllocation()* *GOLDPEGASTOKEN#releasePrivateSaleAllocation()* *GOLDPEGASTOKEN#releaseStakingAllocation()* *GOLDPEGASTOKEN#updateLockStatus()*

04. Issues Found

MAJOR : Owner can restrict token transfers of other accounts (Found - v.1.0)

MAJOR

```
248   function updateLockStatus(address _address, bool locked) onlyOwner public {
249     lock[_address] = locked;
250   }
```

[https://github.com/goldpegasio/contracts/blob/1f236ced18589ff865dff329dee53e2e3ec9c99d/GOLDPEGAS.sol#L248-L250]

Owner can call the *GOLDPEGASTOKEN#updateLockStatus()* function to restrict token transfers of other accounts.

MAJOR : Token may be released to an address that is not a contract

(Found - v.1.0)

MAJOR

```
79  function releaseAirdropAllocation(address _contract) public onlyOwner {
80    require(!releaseAirdrop, 'Airdrop Allocation had released!!!');
81    releaseAirdrop = true;
82    _transfer(address(this), _contract, airdropAllocation);
83  }
84
85  function releaseFarmAllocation(address _farmerAddress, uint256 _amount) public onlyFarmContract {
86    require(farmingReleased.add(_amount) <= farmingAllocation, 'Max farming allocation had released!!!');
87    _transfer(address(this), _farmerAddress, _amount);
88    farmingReleased = farmingReleased.add(_amount);
89  }
90
91  function releaseLiquidityPoolAllocation(address _contract) public onlyOwner {
92    require(!releaseLiquidityPool, 'LiquidityPool Allocation had released!!!');
93    releaseLiquidityPool = true;
94    _transfer(address(this), _contract, liquidityPoolAllocation);
95  }
96
97  function releasePrivateSaleAllocation(address _contract) public onlyOwner {
98    require(!releasePrivateSale, 'Private sale Allocation had released!!!');
99    releasePrivateSale = true;
100   _transfer(address(this), _contract, privateSaleAllocation);
101 }
102
103 function releaseStakingAllocation(address _contract) public onlyOwner {
104   require(!releaseStaking, 'Staking Allocation had released!!!');
105   releaseStaking = true;
106   _transfer(address(this), _contract, stakingAllocation);
107 }
```

[https://github.com/goldpegasio/contracts/blob/1f236ced18589ff865dff329dee53e2e3ec9c99d/GOLDPEGAS.sol#L79-L107]

There is no guarantee in the _contract parameter in the *GOLDPEGASTOKEN#release[Airdrop|Farm|LiquidityPool|PrivateSale|Staking]Allocation()* function that allows release exclusively to an intended and specific contract. With the Owner's authority, Owner can release tokens to any address including Owner itself.

Recommendation

We recommend creating and distributing contracts in advance, assigning each address as a variable in *GOLDPEGASTOKEN#constructor()*, then releasing tokens by using each variable in the *GOLDPEGASTOKEN#release[Airdrop|Farm|LiquidityPool|PrivateSale|Staking]Allocation()* function.

TIPS : There are missing Events (Found - v.1.0)

TIPS

The following list shows functions with missing Events.

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

Without Event, it is difficult to identify in real-time whether correct values are recorded on the blockchain. In this case, it becomes problematic to determine whether the corresponding value has been changed in the application and whether the corresponding function has been called.

Thus, we recommended adding Events corresponding to the change occurring in the function.

05. Disclaimer

This report does not guarantee investment advice, the 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 on Klaytn. In order to write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix A. Test Results

The following results show the unit test results covering 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 existing issues.

```
GOLDPEGAS
 #constructor()
   ✓ should set name properly
   ✓ should set symbol properly
   ✓ should set decimals properly
   ✓ should set initial supply properly
 BEB20 Spec
  #transfer()
    ✓ should fail if sender's amount is lower than balance
    ✓ should fail if msg.sender is locked
    ✓ totalSupply should decrease if recipient is Zero Address
   when succeeded
     ✓ sender's balance should decrease
     ✓ recipient's balance should increase
     ✓ should emit Transfer event
  #transferFrom()
    ✓ 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
    ✓ should fail if from account is locked
   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 sender is ZERO_ADDRESS
    ✓ should fail if spender is ZERO_ADDRESS
    valid case
      ✓ allowance should set appropriately
      ✓ should emit Approval event
  #increaseAllowance()
    ✓ should fail if spender is ZERO_ADDRESS
    ✓ should fail if overflows
    valid case
      ✓ allowance should set appropriately
      ✓ should emit Approval event
  #decreaseAllowance()
    ✓ should fail if spender is ZERO_ADDRESS
    ✓ should fail if underflows
    valid case
      ✓ allowance should set appropriately
      ✓ should emit Approval event
BEB20Burnable 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
      1) should emit Burn event
#updateLockStatus()
  ✓ should fail if msg.sender is not owner
  valid case
    ✓ lockStatus should update properly
#transferOwnership()
  ✓ should fail if msg.sender is not owner
  ✓ should fail if try to transfer ownership to AddressZero
```

valid case

✓ should change owner to newOwner

✓ should emit OwnershipTransferred event

#releaseAirdropAllocation()

✓ should fail if already released

✓ should fail if msg.sender is not owner

valid case

✓ releaseAirdrop should be set true

✓ should transfer contract's token properly

#releaseLiquidityPoolAllocation()

✓ should fail if already released

✓ should fail if msg.sender is not owner

valid case

✓ releaseAirdrop should be set true

✓ should transfer contract's token properly

#releasePrivateSaleAllocation()

✓ should fail if already released

✓ should fail if msg.sender is not owner

valid case

✓ releaseAirdrop should be set true

✓ should transfer contract's token properly

#releaseStakingAllocation()

✓ should fail if already released

✓ should fail if msg.sender is not owner

valid case

✓ releaseAirdrop should be set true

✓ should transfer contract's token properly

#releaseFarmAllocation()

✓ should fail if msg.sender is nether owner nor farmContract

✓ should fail if sum of released amount and amount to be released exceeds farmingAllocation

valid case

✓ should transfer contract's token properly

#setFarmAddress()

```
    ✓ should fail if msg.sender is not owner
    ✓ should fail if address is Zero address
   valid case
     ✓ farmAddresses should be set properly


SafeMath
  add
    ✓ adds correctly
    ✓ reverts on addition overflow
  sub
    ✓ subtracts correctly
    ✓ reverts if subtraction result would be negative
  mul
    ✓ multiplies correctly
    ✓ multiplies by zero correctly
    ✓ reverts on multiplication overflow
  div
    ✓ divides correctly
    ✓ divides zero correctly
    ✓ returns complete number result on non-even division
    ✓ reverts on division by zero
  mod
    ✓ reverts with a 0 divisor
   modulos correctly
     ✓ when the dividend is smaller than the divisor
     ✓ when the dividend is equal to the divisor
     ✓ when the dividend is larger than the divisor
     ✓ when the dividend is a multiple of the divisor
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/					

GOLDPEGAS.sol	**100**	**100**	**100**	**100**	
contracts/libs/goldpegas/					
Context.sol	**100**	**100**	**100**	**100**	
contracts/libs/zeppelin/math/					
SafeMath.sol	**100**	**100**	**100**	**100**	
contracts/libs/zeppelin/token/BEP20/					
IBEP20.sol	**100**	**100**	**100**	**100**	

[Table 1] Test Case Coverage