20th JULY 2021

ELYSIA

SMART CONTRACT AUDIT REPORT

-

version v1.0

ERC20 Security Audit and General Analysis

HAECHI AUDIT

Table of Contents

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

Table of Contents

About HAECHI AUDIT

01. Introduction

02. Summary

- Issues

03. Overview

- Contracts Subject to Audit
- Key Features
- Roles

04. Issues Found

- MAJOR : A code is implemented differently than intended. (Found - v.1.0)
- TIPS : There is a missing Event. (Found - v.1.0)

05. Disclaimer

Appendix A. Test Results

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 ElyfiToken smart contract created by Elyfi team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by ElyfiToken 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 are those that can improve the usability or efficiency of the code when corrected.

HAECHI AUDIT recommends that Elyfi team improves 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/elysia-dev/elyfi-token/tree/d89e336cbf86d229b6b47d301d22a33cb5287ccd/contracts). The last commit of the code used for this Audit is "d89e336cbf86d229b6b47d301d22a33cb5287ccd".

Issues

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

Severity	Issue	Status
MAJOR	A code is implemented differently than intended	(Found - v1.0)
TIPS	There is a missing Event	(Found - v1.0)

03. Overview

Contracts Subject to Audit

- ElyfiToken
- ElyfiAccessControl

Key Features

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

- Token burn (Burn)
- Token migration (Migrate)

Roles

ElyfiToken Smart contract has the following functions.

- **Admin**
- **SnapshotMaker**

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

Role	MAX	Addable	Deletable	Transferable	Renouncable
Admin	∞	O	O	X	O
SnapshotMaker	∞	O	O	X	O

Each authority can access the following functions.

Role	Functions
Admin	*ElyfiToken#initMigration()*
SnapshotMaker	*ElyfiToken#snapshot()*

04. Issues Found

MAJOR : A code is implemented differently than intended. (Found - v.1.0)

MAJOR

```
32    function initMigration(address newElyfiToken_) public onlyAdmin {
33      require(_newElyfiToken != address(0), "Already Initialized");
34
35      _newElyfiToken = newElyfiToken_;
36    }
```

[https://github.com/elysia-dev/elyfi-token/blob/d89e336cbf86d229b6b47d301d22a33cb5287ccd/contracts/token/ElyfiToken.sol#L32-L36]

In order to check whether migration is already initialized in the *ElyfiToken#initMigration()* function, it seems that a require statement has been put to confirm if *ElyfiToken#_newElyfiToken* variable is Zero Address.

However, because the default value of the *ElyfiToken#_newElyfiToken* variable is Zero Address and there is a require statement that requires *ElyfiToken#_newElyfiToken* not to be Zero Address, it always reverts when the corresponding function is called.

Recommendation

It is advised to modify the require statement of *ElyfiToken#initMigration()* into a require() statement that requires *ElyfiToken#_newElyfiToken* to be Zero Address.

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

TIPS

The following is a list of 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 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.

```
ElyfiToken
#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
      1) should emit Burn event
  #snapshot()
    ✓ should fail if msg.sender is not snapshotMaker (52ms)
  #initMigration()
    ✓ should fail if msg.sender is not admin
    2) should fail if already initialized
    valid case
      ✓ _newElyfiToken should set properly
  #migrate()
    ✓ should fail if not-yet-initialized
    ✓ should fail if userBalance is zero
    valid case
      ✓ msg.sender's old token balance should be zero
      ✓ msg.sender's new token balance should increase properly
      ✓ should emit Migrate event


  ERC20Snapshot
  #_snapshot()
    ✓ should creates increasing snapshots ids, starting from 1
    valid case
      ✓ should emits a snapshot event
```

```
#totalSupplyAt()
  ✓ should fail if snapshot id is zero
  ✓ should fail if snapshot is not-yet-created
  valid case
    with no supply changes after the snapshot
      ✓ should return current total supply
    with supply changes after the snapshot
      ✓ should return total supply before the changes
      with a second snapshot after supply changes
        ✓ should return the supply before and after the changes
      with multiple snapshots after supply changes
        ✓ all posterior snapshots should return supply after the changes
#balanceOfAt()
  ✓ should fail if snapshot id is zero
  ✓ should fail if snapshot is not-yet-created
  valid case
    with no balance changes after the snapshot
      ✓ should return current balance for all accounts
    with balance changes after the snapshot
      ✓ should returns the balances before the changes
      with a second snapshot after supply changes
        ✓ snapshots should return the balances before and after the changes
      with multiple snapshots after supply changes
        ✓ all posterior snapshots should return the supply after the changes (40ms)
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
access/					
ElyfiAccessControl.sol	**100**	**100**	**100**	**100**	
token/					
ElyfiToken.sol	**100**	**100**	**100**	**100**	

[Table 1] Test Case Coverage