SMART CONTRACTS EXPERTISE — RIGHT WAY TO SUCCESS

Cedex Coin Smart Contract Audit

If you have any questions concerning smart contract design and audit, feel free to contact zoia@oceanico.io

Content

Description of the complex of procedures for auditing a smart contract

1. Primary architecture review

- Checking the architecture of the contract.
- Correctness of the code.
- Check for linearity, shortness and self-documentation.
- Static verification and code analysis for validity and the presence of syntactic errors.

2. Comparison of requirements and implementation

- Checking the code of the smart contract for compliance with the requirements of the customer code logic, writing algorithms, matching the initial constant values.
- Identification of potential vulnerabilities

3. Testing according to the requirements

- Control testing of a smart contract for compliance with specified customer requirements.
- Running tests of the properties of the smart contract in test net.

CEDEX Coin Smart Contract

Overview

CEDEX is the First Ever Certified Blockchain Based Diamond Exchange

CEDEX.com is a global exchange that focuses on bridging the gap between the traditional diamond industry and the innovative financial markets. With its extensive industry knowledge, CEDEX wants to engineer a ground-breaking change – enabling people to liquidate and invest in diamonds like any other financial asset, in a transparent and secure way.

The CEDEX exchange enables anyone to invest in individual diamonds, shares of a high- value stone or shares in a basket of diamonds (ETF).

Investors, buyers and sellers are confident because the diamond value is transparent, using the DEX, our machine learning algorithm and blockchain technology that rates a diamond's asking price, liquid because it creates a two- sided market by enhancing both the supply and demand, and fungible because CEDEX creates a unique benchmark value, rate and contract for every stone.

CEDEX was founded by a group of experienced veterans from the FinTech, diamond, financial and online industries, all of whom have extensive industry knowledge.

Become part of one of the most revolutionary changes in the financial industry in centuries – enabling people to finally invest in diamonds like any other commodity.

For more information, visit the website and read the CEDEX Whitepaper.

About the coin

The CEDEX exchange will be powered by the CEDEX coin. This ERC-20 compitable token (source code of the contract you can find here) will be traded over the public Ethereum blockchain and will allow users to purchase diamonds on CEDEX, transforming their assets into diamonds. Use of the CEDEX coin will be the driven by trading volumes generated on the CEDEX Exchange and the demand of the diamond of the diamond ecosystem. The CEDEX coin will be the only means of payment used on the CEDEX exchange. The amount of CEDEX coin is limited to 100,000,000. Any coins offered for sale that are not sold will be burned

CEDEX Coin Requiremets

1. Standard ERC20 “CEDEX Token” with symbol “CEDEX”, 18 decimals and total supply of 100,000,000 units created at contract deployment and assigned to a specific wallet.

2. 50,000,000 tokens are assigned for the public token sale

3. Up to 25,000,000 of the tokens will be sold in the pre-sale and the reminder in the sale.

4. Up to 15,000,000 of the tokens will be given as bonuses for pre-sale and early sale participants.

5. 15,000,000 tokens are assigned for the Founders, team and advisory board. 18 month vesting period for them (10% vested after end of sale, 30% after 6 months, 30% after 12 months and 30% after 18 months)

6. 15,000,000 tokens are assigned for the company and be used in the future for: development and marketing expansion or for diamond purchases to facilitate future financial instruments offered on CEDEX and for operational costs

7. At the end of the token sale, any unsold tokens will be burned

8. Tokens are distributed to users up to three weeks after the sale ends. Once they are distributed, they are transferable.

9. Value of 1 CEDEX coin will not exceed $0.8. The hard cap of Token-Sale will be published in ETH, prior to the Token-Sale on March 10,2018 08:00 GMT, CEDEX will be locking and publishing the final ETH amount to be sold in the Token-sale

10. It should be possible to set a vesting period to tokens while sending them to an ICO contributor

11. 18 month vesting period for the Founders, team and advisory board (10% vested after end of sale, 30% after 6 months, 30% after 12 months and 30% after 18 months)

Installing

This was developed using Node 8.4.0, Truffle 3.4.11.

npm install -g truffle

npm install

git clone https://github.com/Cedex-Diamond-Exchange/contracts-cedex-token.git

cd ICO

npm install zeppelin-solidity

truffle compile

Testing

To run the tests, simply run

truffle test

Composition of the repo

The repo is composed as a Truffle project. The test suite can be found in test folder. The contract is in contracts/TestToken.sol. The deployment scripts are in the migrations folder.

Built With

Truffle - The most popular development framework for Ethereum

OpenZeppelin - Open framework of reusable and secure smart contracts in the Solidity language.

Style Guide - Coding conventions for writing solidity code.

List of audited files

https://github.com/Cedex-Diamond-Exchange/contracts

Github of the project

The following 9 .sol files were the object of the audit:

- BasicToken.sol
- BurnableToken.sol
- Cedex.sol
- ERC20.sol
- ERC20Basic.sol
- Migrations.sol
- Ownable.sol
- StandardToken.sol
- SafeMath.sol

Review of smart contract #1

Cedex token contract review #1

https://github.com/Cedex-Diamond-Exchange/contracts

Important:

It is recommended to import all the contracts that you use (except Cedex.sol and Migrations.sol), from OpenZeppelin. How to do it:

$ npm init

$ npm install zeppelin-solidity

replace the lines

import "./BurnableToken.sol";

import "./Ownable.sol";

in Cedex.sol with

import "zeppelin-solidity/contracts/token/ERC20/BurnableToken.sol";

import "zeppelin-solidity/contracts/ownership/Ownable.sol";

and remove SafeMath, BasicToken, BurnableToken, ERC20, ERC20Basic, Ownable, StandardToken contracts from your contracts directory.

Following the standards will increase the security of your project, reliability and quality of the code.

Review of smart contract #1

Code quality

1. You do not follow the Solidity coding conventions that you refer to. Note the indentation in Cedex.sol. You use tabs instead of spaces in lines 14-17, 25.

2. If you use require not once, you can create a modifier and use it in the method signature:

```
modifier onlyAfterAllowedDate() {
  require(now > transferAllowedDates[msg.sender]);
  _;
}
function transfer(address _to, uint _value) onlyAfterAllowedDate
returns (bool) {
 return super.transfer(_to, _value);
}
```

Testing (all passed)

1. **Successful** Deploy in Ropsten

2. **Successful** Check name, symbol, decimals.

3. **Successful** Checking the distribution function of tokens. Transfer before the date specified in the distribution is not possible.

4. **Successful** Checking the transfer of tokens after the specified date. The balance of the beneficiary increased, the sender's balance decreased.

Review of smart contract #2

Cedex token contract review #2

https://github.com/Cedex-Diamond-Exchange/contracts

Important:

The token is based on the open source of “OpenZepplin”, which is standard in the development environment. No new remarks have been revealed.

Code quality

Previously, comments and recommendations on the quality of the code were taken into account by the developers. The code quality corresponds to the Solidity coding conventions.

To the quality of the code there are no remarks.

Testing (all passed)

Repeated testing of the smart contract code did not reveal any errors. All tests were successful.

1. **Successful** Deploy in Ropsten

2. **Successful** Check name, symbol, decimals.

3. **Successful** Checking the distribution function of tokens. Transfer before the date specified in the distribution is not possible.

4. **Successful** Checking the transfer of tokens after the specified date. The balance of the beneficiary increased, the sender’s balance decreased.

5. **Successful** Checking the distributeToken and transfer. Performing a distributeToken with the specified date and number of tokens. It is impossible to transfer funds by

transfer method, before or after the specified date for a larger transfer amount. The transfer is possible after the specified date with a lower transfer amount.

6. **Successful** Checking the distributeToken and transferFrom. Performing a distributeToken with the specified date and number of tokens. It is impossible to transfer funds by transferFrom method, before or after the specified date for a larger transfer amount. The transfer is possible after the specified date with a lower transfer amount.

```
Contract: Cedex
  ✓ #1 construct (63ms)
  ✓ #2 parameters (107ms)
  ✓ #3 transfer (103ms)
  ✓ #4 transferFrom tokens (117ms)
  ✓ #5 distributeToken and transfer (223ms)
  ✓ #6 distributeToken and transferFrom (177ms)


6 passing (914ms)
```

The contract was tested in several stages. At the first stage, testing was carried out using ganache-cli (https://github.com/trufflesuite/ganache-cli).

At the second stage, the testing was carried out in the Ropsten test network (https://gist.github.com/kolya-t/3c3868fa1a5fc7eeda1f78820a5b39ee)

Review of smart contract #2

```
42      it('#1 construct', async () => {
43          const cedex = await Cedex.new();
44      });
45
46      it('#2 parameters', async () => {
47          const cedex = await Cedex.new();
48          (await cedex.name()).should.be.equals("CEDEX");
49          (await cedex.symbol()).should.be.equals("CEDEX");
50          (await cedex.decimals()).should.be.bignumber.equals(18);
51      });
52
53      it('#3 transfer', async () => {
54          const cedex = await Cedex.new();
55          await cedex.transfer(BUYER_2, 1000);
56          (await cedex.balanceOf(BUYER_2)).should.be.bignumber.equals(1000);
57      });
58
59      it('#4 transferFrom tokens', async () => {
60          const cedex = await Cedex.new();
61          await cedex.approve(BUYER_2, 1000);
62          await cedex.transferFrom(OWNER, BUYER_2, 1000, {from: BUYER_2});
63          (await cedex.balanceOf(BUYER_2)).should.be.bignumber.equals(1000);
64      });
65
66      it('#5 distributeToken and transfer', async () => {
67          const cedex = await Cedex.new();
68          await cedex.distributeToken(BUYER_1, 1000, NOW + 5);
69          await cedex.transfer(BUYER_2, 200, {from: BUYER_1}).should.eventually.be.rejected;
70          await increaseTime(6);
71          await cedex.transfer(BUYER_2, 2000, {from: BUYER_1}).should.eventually.be.rejected;
72          await cedex.transfer(BUYER_2, 200, {from: BUYER_1});
73          (await cedex.balanceOf(BUYER_1)).should.be.bignumber.equals(800);
74          (await cedex.balanceOf(BUYER_2)).should.be.bignumber.equals(200);
75      });
76
77      it('#6 distributeToken and transferFrom', async () => {
78          const cedex = await Cedex.new();
79          await cedex.approve(BUYER_2, 2000, {from: BUYER_1});
80          await cedex.distributeToken(BUYER_1, 1000, NOW + 5);
81          await cedex.transferFrom(BUYER_1, BUYER_2, 200, {from: BUYER_2}).should.eventually.be.rejected;
82          await increaseTime(6);
83          await cedex.transferFrom(BUYER_1, BUYER_2, 2000, {from: BUYER_2}).should.eventually.be.rejected;
84          await cedex.transferFrom(BUYER_1, BUYER_2, 200, {from: BUYER_2});
85          (await cedex.balanceOf(BUYER_1)).should.be.bignumber.equals(800);
86          (await cedex.balanceOf(BUYER_2)).should.be.bignumber.equals(200);
```

In both cases, the tests showed the same results confirming that the code of the “OpenZepplin” was copied correctly and corresponds to the technical task of creating a Cedex token.

Results of contract audit

https://github.com/Cedex-Diamond-Exchange/contracts

The information in this report is a list of tips and recommendations on what to look for and what needs to be done to ensure the performance of a smart contract.

The OCEANICO experts conducted the verification of the smart contract in two iterations and completed the full testing. Based on the results of each iteration, the customer's developers were given recommendations for optimizing the smart contract code to fix bugs and vulnerabilities.

This smart contract complies with the specifications specified in the terms of reference, full testing has shown and does not contain critical code and vulnerability errors.

If changes are made to the functionality of the contract, please submit the smart contract for re-examination to the OCEANICO experts (zoia@oceanico.io).