GVISP 1
your space

2018.03.12.

COIN PAYMENT PROCESSOR.ORG

#OPEN CONSORTIUM cPRO

development@coinpaymentprocessor.org

GVISP1 LTD

#CP PROCESSOR PARTNER

office@gvisp.com

cPRO TOKEN DISTRIBUTION

SMART CONTRACT AUDIT

GVISP1 LTD.

GVISP1 Ltd. **** Your space**** Bul.Vojvode Misica 17, 11000 Belgrade, Serbia, EU

1. Audit description

Two related contracts were checked: **distribution.sol** and **token.sol.**

The purpose of this audit is to check all the functionalities of both distribution.sol and token.sol contract, and to determine the level of security and the probability of adverse outcomes.

token.sol is the contract that generates cPRO tokens. cPRO tokens are then being distributed through **distribution.sol** contract to its initial investors.

In the following lines, contracts are being referenced by the name of the file where they were written in. All files contain exactly one contract, so there is no room for confusion.

2. Quick review

- ✓ **token.sol** contains all characteristics and functionalities of ERC20 standard.
- ✓ All functions and state variables are well commented which is good in order to understand quickly how everything is suppo sed to work.
- ✓ The number of token decimals is 18 which is a recommendation for all tokens (Ethereum has 18 decimals).
- ✓ After deployment of the contract, 100.000.000 cPRO tokens are created, of which 35% is sent (minted) to the **distribution.sol** contract's address, and the rest is sent to the address of the person who deployed the contract.
- ✓ **distribution.sol** is also well commented. It has a simple fallback function which is a good practice.
- ✓ It uses the interface of token.sol contract so it could manage tokens.
- ✓ Token distribution does not start immediately after deployment. In order to start the distribution, an authorized address must call the function StartDistribution.
- ✓ **distribution.sol** must be deployed before token.sol.

3. A brief review of contract's functionalities

token.sol

All the contract's functions are in accordance with ERC20 standard.
Only customization made is the automatic allocation of generated tokens to the creator and distribution.sol contract.

distribution.sol

There is no minimum amount of ETH that has to be collected through this distribution.
The contract has two states (modifiers, not variables):

1. isOn - allows sending ETH to the contract if distribution has started and is not over
2. isOver - allows claiming cPRO tokens after distribution

Distribution's duration is specified in the constructor (during deployment).

4. Functionalities test

token.sol

- Total amount of generated tokens = 100.000.000 : ✓
- Token symbol = cPRO : ✓
- Token name = CP Processor : ✓
- Number of token decimals = 18 : ✓
- Distribution contract's balance = 35.000.000 : ✓
- Token creator's balance = 65.000.000 : ✓
- transfer : ✓
- burn : ✓
- approve : ✓
- approveAndCall : ✓
- transferFrom: ✓
- burnFrom: ✓

distribution.sol

- StartDistribution : ✓
- endTimestamp : ✓
- creator : ✓
- totalTokens = 35.000.000 : ✓
- totalInvested = 0 (initially) : ✓
- started = true : ✓
- fallback : ✓
- claim : ✓
- proxyClaim : ✓
- Authorize : ✓
- Unauthorize: ✓
- withdrawEther : ✓

5. Detailed code check (line-by-line)

token.sol

- ✓ The contract is completely written by the ERC20 standard, so only the characteristics will be stated (not functions as they entirely belong to ERC20).
- ✓ Name = "CP Processor", symbol = "cPRO", decimals = 18 (recommended value). totalSupply = 100.000.000 and can not be increased, only decreased by burning (burn or burnFrom ERC20 function).
- ✓ The constructor takes distribution.sol contract's address as a parameter which

means that distribution.sol contract must be already on-chain at the moment of token.sol deployment.

distribution.sol

- ✓ token.sol file is imported to provide the interface for cPRO tokens management. This interface could be defined in the distribution.sol file itself.

State variables of the contract:

- ✓ cPro tokenContract - cPRO token instance
- ✓ Address creator - address of the person who started the distribution (not necessarily the person who deployed the contract); this information is not necessary, just serves for informational purposes
- ✓ Uint256 **totalTokens** - amount of cPRO tokens this contract owns; after deployment of distribution.sol will be equal to 0 until token.sol is deployed when it increases to 35.000.000; during distribution, this number is constant, and after distribution it decreases as investors claim their cPRO tokens.
- ✓ Uint256 **totalInvested** - amount of ETH investors sent during distribution; not equal to real amount of ETH in the contract because authorized addresses may withdraw ETH at any time during distribution
- ✓ Uint256 **endTimeStamp** - timestamp of distribution completion
- ✓ Bool started - equals “true” if distribution has started, “false” otherwise

There are three types of mapping:

- ➢ (address => uint256) **invested** - remembers the amount of ETH each address sent.
- ➢ (address => bool) **claimed** - remembers if address had claimed it’s tokens.
- ➢ (address => bool) **authorized** - remembers if address is authorized for certain actions (to start distribution or to withdraw ETH from the contract).

There are three modifiers:

- ➢ **isOn** - checks whether the distribution is on
- ➢ **isOver** - checks whether the distribution has ended
- ➢ **isAuthorized** - checks whether the address is authorized

- ✓ Constructor takes one address as a parameter, and that address becomes the initial authorized address. Nothing else happens after deployment of the contract.
- ✓ StartDistribution - takes cPRO token address and timestamp when distribution is suppose to end as parameters, so it may be called only after deployment of token.sol. Only authorized addresses may call this function, and it can be called only once (if it wasn’t called already). In that moment, totalTokens increase to 35.000.000, creator becomes the address of the person who called the function, endTimeStamp becomes equal to defined parameter, started

becomes true and cannot be changed again which ensures exactly one call of this function.

- ✓ Fallback function - sending ETH to the contract is possible only during distribution. Fallback function is simple as it should be, and only remembers how much ETH each investor sent to the contract.
- ✓ proxyClaim - takes the address of the investor to whom the cPRO tokens should be sent as a parameter. This function can be called only after the end of distribution.
- ✓ Claim - called when investors want to claim their tokens

6. Static analysis test, vulnerabilities and outcomes

✓ Static analysis of the code was conducted and no security flows were found.

https://oyente.melon.fund
browser/token.sol:cPro
EVM Code Coverage : 82.2%
Callstack Depth Attack Vulnerability : False
Re-Entrancy Vulnerability : False
Assertion Failure: False
Parity Multisig Bug 2 : False
Transaction-Ordering Dependence (TOD) : False

https://oyente.melon.fund
browser/distribution.sol:Distribution
EVM Code Coverage : 50.9%
Callstack Depth Attack Vulnerability : False
Re-Entrancy Vulnerability : False
Assertion Failure: False
Parity Multisig Bug 2 : False
Transaction-Ordering Dependence (TOD) : False

Generally, it is not a good practice to rely on “now” (distribution.sol contract) as miners could affect the timestamp to a certain extent, but in this case “now” is used for calculating the 30 - days period, and not for generating random or decision making of any kind.

Over & Underflows

- ✓ Subtraction occurs only inside proxyClaim function. Each time invested[_address] increases, totalInvested will increase for the same value, so at any moment amount will be less than or equal to totalTokens, that makes underflow theoretically impossible.
- ✓ Overflow is possible only if total amount of investments has the order of magnitude larger than 10^{59} Eth, which is also theoretically impossible.

7. Final comments

distribution.sol file imports the **token.sol** file but uses only two functions of its interface, so it could be more practical to define the interface inside the distribution.sol file itself.

In solidity, names of functions usually begin with a lowercase letter and event names begin with a capital letter.

The event that fires when investors claim their tokens could be added.

The alternative for “now” would be defining the average number of blocks mined per day, but that wouldn’t be elegant for many reasons: distribution would last 30 ± ∆N days where ∆N could vary from hours to days in special cases (hard fork is possible), so “now” should be left as it is.