FIN & NOM TOKEN SMART CONTRACT AUDIT RESULTS

FOR FINOM AG

05/08/2018

Made in Germany by chainsulting.de

Change history

Version	Date	Author	Changes
1.0	**07.05.2018**	**Y. Heinze**	**Audit created**
1.5	**08.05.2018**	**Y. Heinze**	**Vulnerability check**
2.0	**09.05.2018**	**Y. Heinze**	**Executive Summary**

Smart Contract Audit FINOM ICO

Table of Contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of FINOM AG. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

2. About the Project and Company

Company address:

Finom AG
Alte Haslenstrasse 5
9053 Teufen Switzerland

Company Check: https://www.easymonitoring.ch/handelsregister/finom-ag-357643

Project Overview:

Finom is a Blockchain company founded after the merger of market leaders - trading app TabTrader, mining multipool Nanopool, exchange Cryptonit - along with mining farm Cryptal and brokerage app Beetle.io. Finom is building the genetic code of the future economic and financial system.

Guided by the principles of transparency and security, they create one universal and low cost tool for an easy management and access to finance.

They use Blockchain technology to create trustworthy and convenient financial services. That allow users all over the world to become more independent, to improve their material prosperity and to be sure of safety of their investments.

Finom AG is offering two kind of Tokens, security token (FIN) and utility token (NOM).
Whitepaper::https://finom.io/files/whitepaper_eng.pdf

3. Vulnerability Level

0-Informational severity – A vulnerability that have informational character but is not effecting any of the code.

1-Low severity - A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.

2-Medium severity – A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.

3-High severity – A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.

4-Critical severity – A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.

4. Overview of the audit

The project has two tokens, the FIN Token which contains 290 lines of Solidity code and NOM Token which contains 285 lines of Solidity code. All the functions and state variables are well commented using the natspec documentation for the functions which is good to understand quickly how everything is supposed to work.

Etherscan:

FIN Token address:
https://etherscan.io/address/0xef6efb3fc5b9aba75af7250989db7974fd6361ba#code

NOM Token address:
https://etherscan.io/addres/0x23ab81fd565d49259675eb87209d6899bf2e814d#code

Fast overview about the spec.

Token Standard	Token burning mechanics	Solidity Version	Token minting mechanics
ERC20	Not supported	FIN v0.4.21 NOM v0.4.21	NOM Token
Min/max contribution	**Token bonus structure**	**Number of Tokens**	
Not supported	Not supported	FIN 2,623,304 NOM 5,650,000,000	

Verification used (FIN Token)

Verifiers can be added or removed by the owner of the smart contract. These Verifiers can approve the addresses. Transactions can only be performed between such approved addresses. Approved addresses are addresses, whose users have passed the KYC procedure.

Mintable function (NOM Token)

The first release of tokens (April 28, 2013)	3,390,000,000 (three billion 390 million)
The second additional release (July 28, 2018)	565 million (565 million)
The third additional release (October 28, 2018)	565 million (565 million)
The fourth additional release (January 28, 2019)	565 million (565 million)
The fifth additional release (April 28, 2019)	565 million (565 million)

Only the owner of a smart contract can issue tokens.

Used Code from other Smart Contracts

1. SafeMath (Math operations with safety checks that throw on error)
https://github.com/OpenZeppelin/zeppelin-solidity/tree/master/contracts/math

2. ERC20Basic (Simpler version of ERC20 interface)
https://github.com/ethereum/EIPs/issues/179
https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/token/ERC20/ERC20Basic.sol

3. Standard ERC20 (Based on code by FirstBlood)
https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/token/ERC20/StandardToken.sol

4. Mintable Token
https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/token/ERC20/MintableToken.sol

5. Attack made to the contract

Security Report

Attack: Using the approve function of the ERC-20 standard
The approve function of ERC-20 might lead to vulnerabilities.

FIN Token # 157-161
NOM Token #157-161

```
function approve(address _spender, uint256 _value) public returns(bool) {
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
}
```

Severity: 2

Result / Recommendation:
Only use the approve function of the ERC-20 standard to change allowed amount to 0 or from 0 (wait till transaction is mined and approved).

https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit

Attack: Functions transfer and transferFrom of ERC-20 Token should throw.
Functions of ERC-20 Token Standard should throw in special cases:

- transfer should throw if the _from account balance does not have enough tokens to spend
- transferFrom should throw unless the _from account has deliberately authorized the sender of the message via some mechanism

FIN Token # 283-289

```
function transfer(address _to, uint256 _value) public onlyVerified(msg.sender, _to) returns(bool) {
    super.transfer(_to, _value);
}

function transferFrom(address _from, address _to, uint256 _value) public onlyVerified(_from, _to) returns(bool) {
    super.transferFrom(_from, _to, _value);
}
```

Severity: 1

Result / Recommendation:
The ERC20 standard recommends throwing exceptions in functions transfer and transferFrom.

Attack: Compiler version not fixed
Solidity source files indicate the versions of the compiler they can be compiled with.

pragma solidity ^0.4.21; // bad: compiles w 0.4.21 and above
pragma solidity   0.4.21; // good: compiles w 0.4.21 only

It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee.

Severity: 2

Result / Recommendation:
Specify the exact compiler version (pragma solidity x.y.z;).

You're specifying a pragma version with the caret symbol (^) up front which tells the compiler to use any version of solidity bigger than 0.4.21 .

This is not a good practice since there could be major changes between versions that would make your code unstable. That's why I recommend to set a fixed version without the caret like 0.4.21.

Attack: Unchecked math
Solidity is prone to integer over- and underflow. Overflow leads to unexpected effects and can lead to loss of funds if exploited by a malicious account.

FIN Token # 219

```
uint256 public constant INITIAL_SUPPLY = 2623304 * (10 * * uint256(decimals));
```

Severity: 2

Result / Recommendation:
Check against over- and underflow (use the SafeMath library).
SafeMath is already used in both contracts.

Attack: Unhandled Exception

A **call/send** instruction returns a non-zero value if an exception occurs during the execution of the instruction (e.g., out-of-gas). A contract must check the return value of these instructions and throw an exception.

Severity: 0

Result / Recommendation:

Catching exceptions is not yet possible.

Attack: Transactions May Affect Ether Receiver

A contract is exposed to this vulnerability if a miner (who executes and validates transactions) can reorder the transactions within a block in a way that affects the receiver of ether.

Severity: 1

Result / Recommendation:

Both contracts are not vulnerable to this vulnerability as the receiver of ether is **msg.sender**, which cannot be modified by previously executed transactions

6. Executive Summary

A majority of the code was standard and copied from widely-used and reviewed contracts and as a result, a lot of the code was reviewed before. It correctly implemented widely-used and reviewed contracts for safe mathematical operations. The audit identified no major security vulnerabilities, at the moment of audit.

7. General Summary

The issues identified were minor in nature, and do not affect the security of the contract. The code specifies Solidity version 0.4.21, which has only recently had a newer version of 0.4.24 released. As a result, FINOM AG should consider updating the pragma statements to require the latest version of Solidity.

Additionally, the code implements and uses a SafeMath contract, which defines functions for safe math operations that will throw errors in the cases of integer overflow or underflows. The simplicity of the audited contracts contributed greatly to their security. The minimalist approach in choosing which pieces of functionality to implement meant there was very little attack surface available.

Solidity Version Updates

Solidity 0.4.24 will add several features which could be useful in these contracts:

- Type Checker: Improve error message for failed function overload resolution.
- Type Checker: Do not complain about new-style constructor and fallback function to have the same name.
- Type Checker: Detect multiple constructor declarations in the new syntax and old syntax.
- Type Checker: Explicit conversion of bytesXX to contract is properly disallowed.

Also recommended is to Update the etherscan.io information with Logo/Website for example. That gives buyers more transparency.

NOM Token
https://etherscan.io/tokenupdate?a=0x23ab81fd565d49259675eb87209d6899bf2e814d

FIN Token
https://etherscan.io/tokenupdate?a=0xef6efb3fc5b9aba75af7250989db7974fd6361ba

8. Source Code – Smart Contracts

FIN Token (FIN)

```
pragma  solidity  ^  0.4.21;
/** * @title SafeMath * @dev Math operations with safety checks that throw on error */
library  SafeMath  {       /** * @dev Multiplies two numbers, throws on overflow.  */

    function  mul(uint256  a,  uint256  b)  internal  pure  returns(uint256  c)  {

        if  (a  ==  0)  {
            return  0;
        }
        c  =  a  *  b;
        assert(c  /  a  ==  b);
        return  c;
    }           /** * @dev Integer division of two numbers, truncating the quotient.    */

    function  div(uint256  a,  uint256  b)  internal  pure  returns(uint256)  {  // assert(
b > 0); // Solidity automatically throws when dividing by 0 // uint256 c = a / b; // assert
(a == b * c + a % b); // There is no case in which this doesn't hold

        return  a  /  b;
    }           /** * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is
 greater than minuend).  */
    function  sub(uint256  a,  uint256  b)  internal  pure  returns(uint256)  {

        assert(b  <=  a);
        return  a  -  b;
    }           /** * @dev Adds two numbers, throws on overflow.    */
    function  add(uint256  a,  uint256  b)  internal  pure  returns(uint256  c)  {

        c  =  a  +  b;
        assert(c  >=  a);
        return  c;
    }
}   /** * @title ERC20Basic * @dev Simpler version of ERC20 interface * @dev see https://gi
thub.com/ethereum/EIPs/issues/179 */
contract  ERC20Basic  {
    function  totalSupply()  public  view  returns(uint256);
    function  balanceOf(address  who)  public  view  returns(uint256);
    function  transfer(address  to,  uint256  value)  public  returns(bool);
    event  Transfer(address  indexed  from,  address  indexed  to,  uint256  value);
}   /** * @title Basic token * @dev Basic version of StandardToken, with no allowances. */

contract  BasicToken  is  ERC20Basic  {
    using  SafeMath
    for  uint256;
    mapping(address  =>  uint256)  balances;
    uint256  totalSupply_;       /** * @dev total number of tokens in existence  */

    function  totalSupply()  public  view  returns(uint256)  {
        return  totalSupply_;
    }           /** * @dev transfer token for a specified address    * @param _to The addres
s to transfer to.    * @param _value The amount to be transferred.   */
    function  transfer(address  _to,  uint256  _value)  public  returns(bool)  {

        require(_to  !=  address(0));
        require(_value  <=  balances[msg.sender]);
        balances[msg.sender]  =  balances[msg.sender].sub(_value);
        balances[_to]  =  balances[_to].add(_value);
        emit  Transfer(msg.sender,  _to,  _value);
        return  true;
    }           /** * @dev Gets the balance of the specified address.   * @param _owner The address to query the the balance of.    * @return An uint256 representing the amount owned by the passed address.   */
    function  balanceOf(address  _owner)  public  view  returns(uint256)  {
        return  balances[_owner];
    }
}   /** * @title ERC20 interface * @dev see https://github.com/ethereum/EIPs/issues/20 */

contract  ERC20  is  ERC20Basic  {
    function  allowance(address  owner,  address  spender)  public  view  returns(uint256);

    function  transferFrom(address  from,  address  to,  uint256  value)  public  returns(bool);
    function  approve(address  spender,  uint256  value)  public  returns(bool);
    event  Approval(address  indexed  owner,  address  indexed  spender,  uint256  value);

}   /** * @title Standard ERC20 token * * @dev Implementation of the basic standard token. * @dev https://github.com/ethereum/EIPs/issues/20 * @dev Based on code by FirstBlood: https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol */
contract  StandardToken  is  ERC20,  BasicToken  {
    mapping(address  =>  mapping(address  =>  uint256))  internal  allowed;      /**  * @dev Transfer tokens from one address to another  * @param _from address The address which you want to send tokens from   * @param _to address The address which you want to transfer to   * @param _value uint256 the amount of tokens to be transferred  */
    function  transferFrom(address  _from,  address  _to,  uint256  _value)  public  returns(bool)  {
        require(_to  !=  address(0));
        require(_value  <=  balances[_from]);
        require(_value  <=  allowed[_from][msg.sender]);
        balances[_from]  =  balances[_from].sub(_value);
        balances[_to]  =  balances[_to].add(_value);
        allowed[_from][msg.sender]  =  allowed[_from][msg.sender].sub(_value);

        emit  Transfer(_from,  _to,  _value);
        return  true;
    }           /**  * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.  *   * Beware that changing an allowance with this method brings the risk that someone may use both the old  * and the new allowance by unfortunate transaction ordering. One possible solution to mitigate this     * race condition is to first reduce the spender's allowance to 0 and set the desired value afterwards:  * https://github.com/ethereum/EIPs/issues/20#issuecomment-
263524729     * @param _spender The address which will spend the funds.    * @param _value The amount of tokens to be spent.    */
    function  approve(address  _spender,  uint256  _value)  public  returns(bool)  {

        allowed[msg.sender][_spender]  =  _value;
        emit  Approval(msg.sender,  _spender,  _value);
        return  true;
    }           /**  * @dev Function to check the amount of tokens that an owner allowed to a spender.   * @param _owner address The address which owns the funds.    * @param _spender address The address which will spend the funds.    * @return A uint256 specifying the amount of tokens still available for the spender.    */
    function  allowance(address  _owner,  address  _spender)  public  view  returns(uint256)  {
        return  allowed[_owner][_spender];
    }           /**  * @dev Increase the amount of tokens that an owner allowed to a spender.    *   * approve should be called when allowed[_spender] == 0. To increment    * allowed value is better to use this function to avoid 2 calls (and wait until     * the first transaction is mined)   * From MonolithDAO Token.sol    * @param _spender The address which will spend the funds.   * @param _addedValue The amount of tokens to increase the allowance by.    */
    function  increaseApproval(address  _spender,  uint  _addedValue)  public  returns(bool)  {
        allowed[msg.sender][_spender]  =  allowed[msg.sender][_spender].add(_addedValue);

        emit  Approval(msg.sender,  _spender,  allowed[msg.sender][_spender]);
        return true;
    }         /**  * @dev Decrease the amount of tokens that an owner allowed to a spender.    *   * approve should be called when allowed[_spender] == 0. To decrement    * allowed value is better to use this function to avoid 2 calls (and wait until     * the first transaction is mined)   * From MonolithDAO Token.sol    * @param _spender The address which will spend the funds.   * @param _subtractedValue The amount of tokens to decrease the allowance by.    */
    function decreaseApproval(address _spender, uint _subtractedValue) public returns (bool) {
        uint oldValue = allowed[msg.sender][_spender];
        if (_subtractedValue > oldValue) {
            allowed[msg.sender][_spender] = 0;
        }
        else {
            allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
        }
        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }
}
contract FinToken is StandardToken {
    address public owner;
    string public constant name = "FIN Token";
    string public constant symbol = "FIN";
    uint8 public constant decimals = 18;
    uint256 public constant INITIAL_SUPPLY = 2623304 * (10 * * uint256(decimals));
    mapping(address => bool) internal verificatorAddresses;
    mapping(address => bool) internal verifiedAddresses;
    event AddVerificator(address indexed verificator);
    event RemoveVerificator(address indexed verificator);
    event AddVerified(address indexed verificatorAddress, address indexed verified);
    event RemoveVerified(address indexed verificatorAddress, address indexed verified);
    event Mint(address indexed to, uint256 amount);
    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }
    modifier onlyVerificator() {
        require(isVerificator(msg.sender));
        _;
    }
    modifier onlyVerified(address _from, address _to) {
        require(isVerified(_from));
        require(isVerified(_to));
        _;
    }
    function FinToken() public {
        owner = msg.sender;
        totalSupply_ = INITIAL_SUPPLY;
        balances[msg.sender] = INITIAL_SUPPLY;
        emit Transfer(0x0, msg.sender, INITIAL_SUPPLY);
    }
    function addVerificatorAddress(address addr) public onlyOwner {
        verificatorAddresses[addr] = true;
        emit AddVerificator(addr);
    }
    function removeVerificatorAddress(address addr) public onlyOwner {

        delete verificatorAddresses[addr];
        emit RemoveVerificator(addr);
    }
    function isVerificator(address addr) public constant returns(bool) {

        return verificatorAddresses[addr];
    }
    function addVerifiedAddress(address addr) public onlyVerificator {
        verifiedAddresses[addr] = true;
        emit AddVerified(msg.sender, addr);
    }
    function removeVerifiedAddress(address addr) public onlyVerificator {
        delete verifiedAddresses[addr];
        emit RemoveVerified(msg.sender, addr);
    }
    function isVerified(address addr) public constant returns(bool) {
        return verifiedAddresses[addr];
    }
    function transfer(address _to, uint256 _value) public onlyVerified(msg.sender, _to) returns(bool) {
        super.transfer(_to, _value);
    }
    function transferFrom(address _from, address _to, uint256 _value) public onlyVerified(_from, _to) returns(bool) {
        super.transferFrom(_from, _to, _value);
    }
}
```

NOM Token (NOM)

```
pragma solidity ^ 0.4.21;
/** * @title SafeMath * @dev Math operations with safety checks that throw on error */
library SafeMath {        /** * @dev Multiplies two numbers, throws on overflow.  */
    function mul(uint256 a, uint256 b) internal pure returns(uint256 c) {
        if (a == 0) {
            return 0;
        }
        c = a * b;
        assert(c / a == b);
        return c;
    }           /** * @dev Integer division of two numbers, truncating the quotient.    */
    function div(uint256 a, uint256 b) internal pure returns(uint256) {   // assert(b > 0); // Solidity automatically throws when dividing by 0 // uint256 c = a / b; // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return a / b;
    }           /** * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than minuend).  */
    function sub(uint256 a, uint256 b) internal pure returns(uint256) {
        assert(b <= a);
        return a - b;
    }           /** * @dev Adds two numbers, throws on overflow.    */
    function add(uint256 a, uint256 b) internal pure returns(uint256 c) {
        c = a + b;
        assert(c >= a);
        return c;
    }
```

```
}    /** * @title ERC20Basic * @dev Simpler version of ERC20 interface * @dev see https://github.com/ethereum/EIPs/issues/179 */
contract  ERC20Basic  {
    function  totalSupply()  public  view  returns(uint256);
    function  balanceOf(address  who)  public  view  returns(uint256);
    function  transfer(address  to,  uint256  value)  public  returns(bool);

    event  Transfer(address  indexed  from,  address  indexed  to,  uint256  value);
}    /** * @title Basic token * @dev Basic version of StandardToken, with no allowances. */

contract  BasicToken  is  ERC20Basic  {
    using  SafeMath
    for  uint256;
    mapping(address  =>  uint256)  balances;
    uint256  totalSupply_;       /** * @dev total number of tokens in existence  */

    function  totalSupply()  public  view  returns(uint256)  {

        return  totalSupply_;
    }            /** * @dev transfer token for a specified address   * @param _to The address to transfer to.    * @param _value The amount to be transferred.   */
    function  transfer(address  _to,  uint256  _value)  public  returns(bool)  {

        require(_to  !=  address(0));
        require(_value  <=  balances[msg.sender]);
        balances[msg.sender]  =  balances[msg.sender].sub(_value);

        balances[_to]  =  balances[_to].add(_value);
        emit  Transfer(msg.sender,  _to,  _value);
        return  true;
    }            /** * @dev Gets the balance of the specified address.   * @param _owner The address to query the the balance of.    * @return An uint256 representing the amount owned by the passed address.   */
    function  balanceOf(address  _owner)  public  view  returns(uint256)  {

        return  balances[_owner];
    }
}    /** * @title ERC20 interface * @dev see https://github.com/ethereum/EIPs/issues/20 */

contract  ERC20  is  ERC20Basic  {
    function  allowance(address  owner,  address  spender)  public  view  returns(uint256);
    function  transferFrom(address  from,  address  to,  uint256  value)  public  returns(bool);
    function  approve(address  spender,  uint256  value)  public  returns(bool);

    event  Approval(address  indexed  owner,  address  indexed  spender,  uint256  value);
}    /** * @title Standard ERC20 token * * @dev Implementation of the basic standard token.  * @dev https://github.com/ethereum/EIPs/issues/20 * @dev Based on code by FirstBlood: https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol */
contract  StandardToken  is  ERC20,  BasicToken  {
    mapping(address  =>  mapping(address  =>  uint256))  internal  allowed;        /**  * @dev Transfer tokens from one address to another  * @param _from address The address which you want to send tokens from   * @param _to address The address which you want to transfer to  * @param _value uint256 the amount of tokens to be transferred  */
    function  transferFrom(address  _from,  address  _to,  uint256  _value)  public  returns(bool)  {
        require(_to  !=  address(0));
        require(_value  <=  balances[_from]);
        require(_value  <=  allowed[_from][msg.sender]);
        balances[_from]  =  balances[_from].sub(_value);
        balances[_to]  =  balances[_to].add(_value);
        allowed[_from][msg.sender]  =  allowed[_from][msg.sender].sub(_value);

        emit  Transfer(_from,  _to,  _value);
        return true;
    }           /**   * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.   *   * Beware that changing an allowance with this method brings the risk that someone may use both the old  * and the new allowance by unfortunate transaction ordering. One possible solution to mitigate this     * race condition is to first reduce the spender's allowance to 0 and set the desired value afterwards:  * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729     * @param _spender The address which will spend the funds.   * @param _value The amount of tokens to be spent.    */
    function approve(address _spender, uint256 _value) public returns(bool) {

        allowed[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);
        return true;
    }           /**   * @dev Function to check the amount of tokens that an owner allowed to a spender.   * @param _owner address The address which owns the funds.   * @param _spender address The address which will spend the funds.   * @return A uint256 specifying the amount of tokens still available for the spender.    */
    function allowance(address _owner, address _spender) public view returns(uint256) {
        return allowed[_owner][_spender];
    }           /**   * @dev Increase the amount of tokens that an owner allowed to a spender.    *   * approve should be called when allowed[_spender] == 0. To increment    * allowed value is better to use this function to avoid 2 calls (and wait until     * the first transaction is mined)   * From MonolithDAO Token.sol    * @param _spender The address which will spend the funds.   * @param _addedValue The amount of tokens to increase the allowance by.    */
    function increaseApproval(address _spender, uint _addedValue) public returns(bool) {
        allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_addedValue);

        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);

        return true;
    }           /**   * @dev Decrease the amount of tokens that an owner allowed to a spender.    *   * approve should be called when allowed[_spender] == 0. To decrement    * allowed value is better to use this function to avoid 2 calls (and wait until     * the first transaction is mined)   * From MonolithDAO Token.sol    * @param _spender The address which will spend the funds.   * @param _subtractedValue The amount of tokens to decrease the allowance by.    */
    function decreaseApproval(address _spender, uint _subtractedValue) public returns(bool) {
        uint oldValue = allowed[msg.sender][_spender];
        if (_subtractedValue > oldValue) {
            allowed[msg.sender][_spender] = 0;
        }
        else {
            allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);

        }
        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);

        return true;
    }
}
contract NomToken is StandardToken {
    event Mint(address indexed to, uint256 amount);
    address public owner;
    string public constant name = "NOM Token";
    string public constant symbol = "NOM";
        uint8 public constant decimals = 18;
        uint256 public constant totalTokens = 5650000000 * (10 * * uint256(decimals));
        uint256 public initialIssueMinting = totalTokens.mul(60).div(100); //60% of tokens

        uint public constant initialIssueMintingDate = 1524873600; //28.04.2018 UTC

        bool public initialIssueMinted = false;
        uint256 public firstStageMinting = totalTokens.mul(10).div(100); //10% of tokens

        uint public constant firstStageMintingDate = 1532736000; //28.07.2018 UTC

        bool public firstStageMinted = false;
        uint256 public secondStageMinting = totalTokens.mul(10).div(100); //10% of tokens

        uint public constant secondStageMintingDate = 1540684800; //28.10.2018 UTC

        bool public secondStageMinted = false;
        uint256 public thirdStageMinting = totalTokens.mul(10).div(100); //10% of tokens

        uint public constant thirdStageMintingDate = 1548633600; //28.01.2019 UTC

        bool public thirdStageMinted = false;
        uint256 public fourthStageMinting = totalTokens.mul(10).div(100); //10% of tokens

        uint public constant fourthStageMintingDate = 1556409600; //28.04.2019 UTC

        bool public fourthStageMinted = false;
        function NomToken() public {
            owner = msg.sender;
        }   /**  * @dev Function to mint tokens  * @return A boolean that indicates if the operation was successful.    */
        function mint() public returns(bool) {
            require(msg.sender == owner);
            uint256 tokensToMint = 0;
            if (now > initialIssueMintingDate && !initialIssueMinted) {
                tokensToMint = tokensToMint.add(initialIssueMinting);
                initialIssueMinted = true;
            }
            if (now > firstStageMintingDate && !firstStageMinted) {
                tokensToMint = tokensToMint.add(firstStageMinting);
                firstStageMinted = true;
            }
            if (now > secondStageMintingDate && !secondStageMinted) {
                tokensToMint = tokensToMint.add(secondStageMinting);
                secondStageMinted = true;
            }
            if (now > thirdStageMintingDate && !thirdStageMinted) {
                tokensToMint = tokensToMint.add(thirdStageMinting);
                thirdStageMinted = true;
            }
            if (now > fourthStageMintingDate && !fourthStageMinted) {
                tokensToMint = tokensToMint.add(fourthStageMinting);
```

```
                fourthStageMinted = true;
            }
            require(tokensToMint > 0);
            uint256 newTotalSupply = totalSupply_.add(tokensToMint);

            require(newTotalSupply <= totalTokens);
            totalSupply_ = totalSupply_.add(tokensToMint);
            balances[owner] = balances[owner].add(tokensToMint);

            emit Mint(owner, tokensToMint);
            emit Transfer(0x0, owner, tokensToMint);
            return true;
        }
    }
```