Smart Contract Security Audit Report

The SlowMist Security Team received the ROONEX team's application for smart contract security audit of the RNX on April 1, 2020. The following are the details and results of this smart contract security audit:

Token name :

RNX

The Contract address :

0x72a6344185B383035d6665C3f44a9DfCC73873c8

Link address :

https://etherscan.io/address/0x72a6344185B383035d6665C3f44a9DfCC73873c8

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

No.	Audit Items	Audit Subclass	Audit Subclass Result
1	Overflow Audit	-	Passed
2	Race Conditions Audit	-	Passed
3	Authority Control Audit	Permission vulnerability audit	Passed
		Excessive auditing authority	Passed
4	Safety Design Audit	Zeppelin module safe use	Passed
		Compiler version security	Passed
		Hard-coded address security	Passed
		Fallback function safe use	Passed
		Show coding security	Passed
		Function return value security	Passed
		Call function security	Passed
5	Denial of Service Audit	-	Passed
6	Gas Optimization Audit	-	Passed
7	Design Logic Audit	-	Passed
8	“False Deposit” vulnerability Audit	-	Passed

9	Malicious Event Log Audit	-	Passed
10	Scoping and Declarations Audit	-	Passed
11	Replay Attack Audit	ECDSA's Signature Replay Audit	Passed
12	Uninitialized Storage Pointers Audit	-	Passed
13	Arithmetic Accuracy Deviation Audit	-	Passed

Audit Result : **Passed**

Audit Number : 0X002004070001

Audit Date : April 07, 2020

Audit Team : SlowMist Security Team

(**Statement :** SlowMist only issues this report based on the fact that has occurred or existed before the report is issued, and bears the corresponding responsibility in this regard. For the facts occur or exist later after the report, SlowMist cannot judge the security status of its smart contract. SlowMist is not responsible for it. The security audit analysis and other contents of this report are based on the documents and materials provided by the information provider to SlowMist as of the date of this report (referred to as "the provided information"). SlowMist assumes that: there has been no information missing, tampered, deleted, or concealed. If the information provided has been missed, modified, deleted, concealed or reflected and is inconsistent with the actual situation, SlowMist will not bear any responsibility for the resulting loss and adverse effects. SlowMist will not bear any responsibility for the background or other circumstances of the project.)

Summary: This is a token contract that contains the tokenVault section .The total amount of contract tokens can be changed, users can burn their tokens through the burn function. The owner can freeze any user account. It is suggested to set the owner to MultiSig contract to reduce the risk of being attacked. The owner can lock any user account. OpenZeppelin’s SafeMath security module is used, which is a commendable approach. The contract does not have the Overflow and the Race Conditions issue.

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2019-12-19
*/

/**
 *Submitted for verification at Etherscan.io on 2019-05-28
*/
```

//SlowMist// The contract does not have the Overflow and the Race Conditions issue

```
pragma solidity ^0.4.24;

// File: contracts/openzeppelin/ownership/Ownable.sol

/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization control
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable {
  address private _owner;
  event OwnershipRenounced(address indexed previousOwner);
  event OwnershipTransferred(
    address indexed previousOwner,
    address indexed newOwner
  );


  /**
   * @dev The Ownable constructor sets the original `owner` of the contract to the sender
   * account.
   */
  constructor() public {
    _owner = msg.sender;
  }

  /**
   * @return the address of the owner.
   */
  function owner() public view returns(address) {
    return _owner;
  }

  /**
   * @dev Throws if called by any account other than the owner.
   */
  modifier onlyOwner() {
    require(isOwner());
    _;
  }
```

```
  /**
   * @return true if `msg.sender` is the owner of the contract.
   */
  function isOwner() public view returns(bool) {
    return msg.sender == _owner;
  }

  /**
   * @dev Allows the current owner to relinquish control of the contract.
   * @notice Renouncing to ownership will leave the contract without an owner.
   * It will not be possible to call the functions with the `onlyOwner`
   * modifier anymore.
   */
  function renounceOwnership() public onlyOwner {
    emit OwnershipRenounced(_owner);
    _owner = address(0);
  }

  /**
   * @dev Allows the current owner to transfer control of the contract to a newOwner.
   * @param newOwner The address to transfer ownership to.
   */
  function transferOwnership(address newOwner) public onlyOwner {
    _transferOwnership(newOwner);
  }

  /**
   * @dev Transfers control of the contract to a newOwner.
   * @param newOwner The address to transfer ownership to.
   */
  function _transferOwnership(address newOwner) internal {
    require(newOwner != address(0)); //SlowMist// This check is quite good in avoiding losing control of the contract caused by user mistakes
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
  }
}

// File: contracts/Administratable.sol
```

```
pragma solidity 0.4.24;

// August 21, 2018
// remove all admin code.


contract Administratable is Ownable {
        mapping (address => bool) public superAdmins;

        event AddSuperAdmin(address indexed admin);
        event RemoveSuperAdmin(address indexed admin);


    modifier validateAddress( address _addr )
    {
        require(_addr != address(0x0));
        require(_addr != address(this));
        _;
    }

        modifier onlySuperAdmins {
                require(msg.sender == owner() || superAdmins[msg.sender]);
                _;
        }

        function addSuperAdmin(address _admin) public onlyOwner validateAddress(_admin){
                require(!superAdmins[_admin]);
                superAdmins[_admin] = true;
                emit AddSuperAdmin(_admin);
        }

        function removeSuperAdmin(address _admin) public onlyOwner validateAddress(_admin){
                require(superAdmins[_admin]);
                superAdmins[_admin] = false;
                emit RemoveSuperAdmin(_admin);
        }
}

// File: contracts/TimeLockable.sol

pragma solidity 0.4.24;
```

```
contract TimeLockable is Administratable{

    uint256 private constant ADVISOR_LOCKUP_END      = 0;
    uint256 private constant TEAM_LOCKUP_END         = 0;

    mapping (address => uint256) public timelockedAccounts;
    event LockedFunds(address indexed target, uint256 timelocked);

    modifier isNotTimeLocked() {
        require(now >= timelockedAccounts[msg.sender]);
        _;
    }

    modifier isNotTimeLockedFrom( address _from ) {
        require(now >= timelockedAccounts[_from] && now >= timelockedAccounts[msg.sender]);
        _;
    }

    function timeLockAdvisor(address _target) public onlySuperAdmins validateAddress(_target) {
        require(timelockedAccounts[_target] == 0);
        timelockedAccounts[_target] = ADVISOR_LOCKUP_END;
        emit LockedFunds(_target, ADVISOR_LOCKUP_END);
    }

    function timeLockTeam(address _target) public onlySuperAdmins validateAddress(_target) {
        require(timelockedAccounts[_target] == 0);
        timelockedAccounts[_target] = TEAM_LOCKUP_END;
        emit LockedFunds(_target, TEAM_LOCKUP_END);
    }
}

// File: contracts/Freezable.sol
pragma solidity 0.4.24;


contract Freezable is Administratable {

    bool public frozenToken;
    mapping (address => bool) public frozenAccounts;
```

```
    event FrozenFunds(address indexed _target, bool _frozen);
    event FrozenToken(bool _frozen);

    modifier isNotFrozen( address _to ) {
        require(!frozenToken);
        require(!frozenAccounts[msg.sender] && !frozenAccounts[_to]);
        _;
    }

    modifier isNotFrozenFrom( address _from, address _to ) {
        require(!frozenToken);
        require(!frozenAccounts[msg.sender] && !frozenAccounts[_from] && !frozenAccounts[_to]);
        _;
    }

    function freezeAccount(address _target, bool _freeze) public onlySuperAdmins validateAddress(_target) {
        require(frozenAccounts[_target] != _freeze);
        frozenAccounts[_target] = _freeze;
        emit FrozenFunds(_target, _freeze);
    }

    function freezeToken(bool _freeze) public onlySuperAdmins {
        require(frozenToken != _freeze);
        frozenToken = _freeze;
        emit FrozenToken(frozenToken);
    }
}

// File: contracts/openzeppelin/token/ERC20/IERC20.sol

/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
interface IERC20 {
  function totalSupply() external view returns (uint256);

  function balanceOf(address who) external view returns (uint256);

  function allowance(address owner, address spender)
    external view returns (uint256);
```

```
    function transfer(address to, uint256 value) external returns (bool);

    function approve(address spender, uint256 value)
        external returns (bool);

    function transferFrom(address from, address to, uint256 value)
        external returns (bool);

    event Transfer(
        address indexed from,
        address indexed to,
        uint256 value
    );

    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );
}

// File: contracts/openzeppelin/math/SafeMath.sol

/**
 * @title SafeMath
 * @dev Math operations with safety checks that revert on error
 */
```

//SlowMist// OpenZeppelin ' s SafeMath security Module is used, which is a recommend approach

```
library SafeMath {

    /**
     * @dev Multiplies two numbers, reverts on overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
        if (a == 0) {
            return 0;
```

```
    }

    uint256 c = a * b;
    require(c / a == b);

    return c;
}

/**
* @dev Integer division of two numbers truncating the quotient, reverts on division by zero.
*/
function div(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b > 0); // Solidity only automatically asserts 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 c;
}

/**
* @dev Subtracts two numbers, reverts on overflow (i.e. if subtrahend is greater than minuend).
*/
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b <= a);
    uint256 c = a - b;

    return c;
}

/**
* @dev Adds two numbers, reverts on overflow.
*/
function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a);

    return c;
}

/**
* @dev Divides two numbers and returns the remainder (unsigned integer modulo),
* reverts when dividing by zero.
```

```
    */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}

// File: contracts/openzeppelin/token/ERC20/ERC20.sol

/**
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
 * Originally based on code by FirstBlood:
https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
 */
contract ERC20 is IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowed;

    uint256 private _totalSupply;

    /**
     * @dev Total number of tokens in existence
     */
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    /**
     * @dev Gets the balance of the specified address.
     * @param owner The address to query 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];
    }
```

```
/**
 * @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 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(value <= _balances[msg.sender]);

  require(to != address(0)); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer


  _balances[msg.sender] = _balances[msg.sender].sub(value);
  _balances[to] = _balances[to].add(value);
  emit Transfer(msg.sender, to, value);

  return true; //SlowMist// The return value conforms to the EIP20 specification

}

/**
 * @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) {
    require(spender != address(0)); //SlowMist// This kind of check is very good, avoiding user mistake leading to approve errors

    _allowed[msg.sender][spender] = value;
    emit Approval(msg.sender, spender, value);

    return true; //SlowMist// The return value conforms to the EIP20 specification
  }

  /**
    * @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(value <= _balances[from]);
    require(value <= _allowed[from][msg.sender]);

    require(to != address(0)); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer

    _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; //SlowMist// The return value conforms to the EIP20 specification
```

```
}

/**
  * @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 increaseAllowance(
    address spender,
    uint256 addedValue
)
    public
    returns (bool)
{
    require(spender != address(0));

    _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 decreaseAllowance(
    address spender,
    uint256 subtractedValue
)
    public
    returns (bool)
{
```

```
    require(spender != address(0));

    _allowed[msg.sender][spender] = (
      _allowed[msg.sender][spender].sub(subtractedValue));
    emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
    return true;
}

/**
 * @dev Internal function that mints an amount of the token and assigns it to
 * an account. This encapsulates the modification of balances such that the
 * proper events are emitted.
 * @param account The account that will receive the created tokens.
 * @param amount The amount that will be created.
 */
function _mint(address account, uint256 amount) internal {
    require(account != 0);
    _totalSupply = _totalSupply.add(amount);
    _balances[account] = _balances[account].add(amount);
    emit Transfer(address(0), account, amount);
}

/**
 * @dev Internal function that burns an amount of the token of a given
 * account.
 * @param account The account whose tokens will be burnt.
 * @param amount The amount that will be burnt.
 */
function _burn(address account, uint256 amount) internal {
    require(account != 0);
    require(amount <= _balances[account]);

    _totalSupply = _totalSupply.sub(amount);
    _balances[account] = _balances[account].sub(amount);
    emit Transfer(account, address(0), amount);
}

/**
 * @dev Internal function that burns an amount of the token of a given
 * account, deducting from the sender's allowance for said account. Uses the
 * internal burn function.
 * @param account The account whose tokens will be burnt.
```

```
    * @param amount The amount that will be burnt.
    */
```

//SlowMist// Because _burnFrom() and transferFrom() share the _allowed amount of approve(), if the agent be evil, there is the possibility of malicious burn

```
  function _burnFrom(address account, uint256 amount) internal {
    require(amount <= _allowed[account][msg.sender]);

    // Should https://github.com/OpenZeppelin/zeppelin-solidity/issues/707 be accepted,
    // this function needs to emit an event with the updated approval.
    _allowed[account][msg.sender] = _allowed[account][msg.sender].sub(
      amount);
    _burn(account, amount);
  }
}

// File: contracts/openzeppelin/token/ERC20/ERC20Burnable.sol

/**
 * @title Burnable Token
 * @dev Token that can be irreversibly burned (destroyed).
 */
contract ERC20Burnable is ERC20 {

  /**
   * @dev Burns a specific amount of tokens.
   * @param value The amount of token to be burned.
   */
```

//SlowMist// The frozen address cannot be transferred, but can burn its own tokens through the burn function

```
  function burn(uint256 value) public {
    _burn(msg.sender, value);
  }

  /**
   * @dev Burns a specific amount of tokens from the target address and decrements allowance
   * @param from address The address which you want to send tokens from
   * @param value uint256 The amount of token to be burned
   */
```

```
  function burnFrom(address from, uint256 value) public {
    _burnFrom(from, value);
  }

  /**
   * @dev Overrides ERC20._burn in order for burn and burnFrom to emit
   * an additional Burn event.
   */
  function _burn(address who, uint256 value) internal {
    super._burn(who, value);
  }
}

pragma solidity 0.4.24;

contract RoonexToken is ERC20Burnable, TimeLockable, Freezable
{
    string  public     constant name                   = "ROONEX";
    string  public     constant symbol         = "RNX";
    uint8   public     constant decimals       = 18;

    event Burn(address indexed _burner, uint _value);

    constructor( address _registry, uint _totalTokenAmount ) public
    {
        _mint(_registry, _totalTokenAmount);
        addSuperAdmin(_registry);
    }



    /**
     * @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, uint _value) public validateAddress(_to) isNotTimeLocked() isNotFrozen(_to) returns (bool)
    {
        return super.transfer(_to, _value);
    }

    /**
     * @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, uint _value) public validateAddress(_to) isNotTimeLockedFrom(_from) isNotFrozenFrom(_from, _to) returns (bool)
    {
        return super.transferFrom(_from, _to, _value);
    }

    function approve(address _spender, uint256 _value) public validateAddress(_spender) isNotFrozen(_spender) isNotTimeLocked() returns (bool)
    {
        return super.approve(_spender, _value);
    }

    function increaseAllowance( address _spender, uint256 _addedValue ) public validateAddress(_spender) isNotFrozen(_spender) isNotTimeLocked() returns (bool)
    {
        return super.increaseAllowance(_spender, _addedValue);
    }

    function decreaseAllowance(address _spender, uint256 _subtractedValue) public validateAddress(_spender) isNotFrozen(_spender) isNotTimeLocked() returns (bool)
    {
        return super.decreaseAllowance(_spender, _subtractedValue);
    }



    /**
    * @dev Token Contract Emergency Drain
    * @param _token - Token to drain
    * @param _amount - Amount to drain
    */
    function emergencyERC20Drain( IERC20 _token, uint _amount ) public onlyOwner {
        _token.transfer( owner(), _amount );
    }
}
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account