Smart Contract Security Audit Report

The SlowMist Security Team received the BTC Token team's application for smart contract security audit of the BTC Token on September 14, 2020. The following are the details and results of this smart contract security audit:

Token name :

BTC Token

The Contract address :

TN3W4H6rK2ce4vX9YnFQHwKENnHjoxb3m9

Link address :

https://tronscan.org/#/contract/TN3W4H6rK2ce4vX9YnFQHwKENnHjoxb3m9

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 : 0X002009170002

Audit Date : September 17, 2020

Audit Team : SlowMist Security Team

Summary: This is a token contract that contain the MultiSig contract and does not contain the tokenVault section. The contract does not have the Overflow and the Race Conditions. Owner can add or delete blacklist address.The blacklist address cannot transfer out of tokens normally. Owner can deprecate current contract in favor of a new one. Owner can issue a new amount of tokens. It is suggested to set the owner to MultiSig contract to reduce the risk of being attacked.

The source code:

BlackList.sol

```
pragma solidity ^0.4.18;
import "./Ownable.sol";
contract BlackList is Ownable {

    /////// Getter to allow the same blacklist to be used also by other contracts (including upgraded Tether) ///////
    function getBlackListStatus(address _maker) external constant returns (bool) {
        return isBlackListed[_maker];
```

```
    }

    mapping (address => bool) public isBlackListed;

    //SlowMist// Owner can add or delete blacklist address.

    function addBlackList (address _evilUser) public onlyOwner {
        isBlackListed[_evilUser] = true;
        AddedBlackList(_evilUser);
    }

    function removeBlackList (address _clearedUser) public onlyOwner {
        isBlackListed[_clearedUser] = false;
        RemovedBlackList(_clearedUser);
    }

    event AddedBlackList(address indexed _user);

    event RemovedBlackList(address indexed _user);
}
```

BasicToken.sol

```
pragma solidity ^0.4.18;

import './SafeMath.sol';
/** * @title ERC20Basic * @dev Simpler version of ERC20 interface * @dev see
https://github.com/ethereum/EIPs/issues/179 */contract ERC20Basic {
  function totalSupply() public constant returns (uint);
  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;

  /**  * @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) {
```

//SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer.

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

    // SafeMath.sub will throw if there is not enough balance.
    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    Transfer(msg.sender, _to, _value);
    return true; //SlowMist// The return value conforms to the TIP20 specification.
  }

  /**  * @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 balance) {
    return balances[_owner];
  }
}
```

Ownable.sol

```
pragma solidity ^0.4.18;

/** * @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 public owner;


  event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);


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

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


  /**   * @dev Allows the current owner to transfer control of the contract to a newOwner.   * @param
newOwner The address to transfer ownership to.   */
```

//SlowMist// When using the transferOwnership function to update the Owner, it is suggested to add the newOwner to accept the confirmation function of the Owner. Only after the signature confirmation of the new address can the real permission transfer be carried out to avoid the loss of the permission.

```
  function transferOwnership(address newOwner) public onlyOwner {
    require(newOwner != address(0));
    OwnershipTransferred(owner, newOwner);
    owner = newOwner;
  }
}
```

Pausable.sol

```
pragma solidity ^0.4.18;


import "./Ownable.sol";
```

//SlowMist// Suspending all transactions upon major abnormalities is a recommended approach.

```
/** * @title Pausable * @dev Base contract which allows children to implement an emergency stop mechanism.
*/contract Pausable is Ownable {
  event Pause();
  event Unpause();

  bool public paused = false;


  /**   * @dev Modifier to make a function callable only when the contract is not paused.   */
  modifier whenNotPaused() {
```

```
    require(!paused);
    _;
  }

  /**   * @dev Modifier to make a function callable only when the contract is paused.   */
  modifier whenPaused() {
    require(paused);
    _;
  }

  /**   * @dev called by the owner to pause, triggers stopped state   */
  function pause() onlyOwner whenNotPaused public {
    paused = true;
    Pause();
  }

  /**   * @dev called by the owner to unpause, returns to normal state   */
  function unpause() onlyOwner whenPaused public {
    paused = false;
    Unpause();
  }}
```

SafeMath.sol

```
pragma solidity ^0.4.18;
```

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

```
/** * @title SafeMath * @dev Math operations with safety checks that throw on error */
library SafeMath {
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }
    uint256 c = a * b;
```

//SlowMist// It is recommended to replace "assert" with "require" to optimize Gas.

```
    assert(c / a == b);
    return c;
  }

  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 c;
  }

  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    //SlowMist// It is recommended to replace "assert" with "require" to optimize Gas.
    assert(b <= a);
    return a - b;
  }

  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    //SlowMist// It is recommended to replace "assert" with "require" to optimize Gas.
    assert(c >= a);
    return c;
  }}
```

StandardToken.sol

```
pragma solidity ^0.4.18;

import './BasicToken.sol';

/** * @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) {
```

//SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer.

```
    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);
    Transfer(_from, _to, _value);
    return true;
```

//SlowMist// The return value conforms to the TIP20 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) {
    allowed[msg.sender][_spender] = _value;
    Approval(msg.sender, _spender, _value);
    return true;
```

//SlowMist// The return value conforms to the TIP20 specification.

```
  }

  /**   * @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];
  }
```

```
  /**   * 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   */
  function increaseApproval(address _spender, uint _addedValue) public returns (bool) {
    allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_addedValue);
    Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
  }

  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);
    }
    Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
  }
}
```

StandardTokenWithFees.sol

```
pragma solidity ^0.4.18;
import "./StandardToken.sol";
import "./Ownable.sol";
contract StandardTokenWithFees is StandardToken, Ownable {

  // Additional variables for use if transaction fees ever became necessary
  uint256 public basisPointsRate = 0;
  uint256 public maximumFee = 0;
  uint256 constant MAX_SETTABLE_BASIS_POINTS = 20;
  uint256 constant MAX_SETTABLE_FEE = 50;

  string public name;
  string public symbol;
  uint8 public decimals;
  uint public _totalSupply;

  uint public constant MAX_UINT = 2**256 - 1;
```

```
function calcFee(uint _value) constant returns (uint) {
  uint fee = (_value.mul(basisPointsRate)).div(10000);
  if (fee > maximumFee) {
      fee = maximumFee;
  }
  return fee;
}

function transfer(address _to, uint _value) public returns (bool) {
  uint fee = calcFee(_value);
  uint sendAmount = _value.sub(fee);

  super.transfer(_to, sendAmount);
  if (fee > 0) {
    super.transfer(owner, fee);
  }
  return true; //SlowMist// The return value conforms to the TIP20 specification.

}

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]);

  uint fee = calcFee(_value);
  uint sendAmount = _value.sub(fee);

  balances[_from] = balances[_from].sub(_value);
  balances[_to] = balances[_to].add(sendAmount);
  if (allowed[_from][msg.sender] < MAX_UINT) {
      allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
  }
  Transfer(_from, _to, sendAmount);
  if (fee > 0) {
    balances[owner] = balances[owner].add(fee);
    Transfer(_from, owner, fee);
  }
  return true; //SlowMist// The return value conforms to the TIP20 specification.
```

```
  }

  function setParams(uint newBasisPoints, uint newMaxFee) public onlyOwner {
      // Ensure transparency by hardcoding limit beyond which fees can never be added
      require(newBasisPoints < MAX_SETTABLE_BASIS_POINTS);
      require(newMaxFee < MAX_SETTABLE_FEE);

      basisPointsRate = newBasisPoints;
      maximumFee = newMaxFee.mul(uint(10)**decimals);

      Params(basisPointsRate, maximumFee);
  }

  // Called if contract ever adds fees
  event Params(uint feeBasisPoints, uint maxFee);
}
```

Bitcoin.sol

```
pragma solidity ^0.4.18;
import "./StandardTokenWithFees.sol";
import "./Pausable.sol";
import "./BlackList.sol";
contract UpgradedStandardToken is StandardToken {
    // those methods are called by the legacy contract
    // and they must ensure msg.sender to be the contract address
    uint public _totalSupply;
    function transferByLegacy(address from, address to, uint value) public returns (bool);
    function transferFromByLegacy(address sender, address from, address spender, uint value) public
returns (bool);
    function approveByLegacy(address from, address spender, uint value) public returns (bool);
    function increaseApprovalByLegacy(address from, address spender, uint addedValue) public returns
(bool);
    function decreaseApprovalByLegacy(address from, address spender, uint subtractedValue) public
returns (bool);}

contract Bitcoin is Pausable, StandardTokenWithFees, BlackList {

    address public upgradedAddress;
    bool public deprecated;
```

```
    //  The contract can be initialized with a number of tokens
    //  All the tokens are deposited to the owner address
    //
    // @param _balance Initial supply of the contract
    // @param _name Token Name
    // @param _symbol Token symbol
    // @param _decimals Token decimals
    function Bitcoin(uint _initialSupply, string _name, string _symbol, uint8 _decimals) public {
        _totalSupply = _initialSupply;
        name = _name;
        symbol = _symbol;
        decimals = _decimals;
        balances[owner] = _initialSupply;
        deprecated = false;
    }

    // Forward ERC20 methods to upgraded contract if this one is deprecated
```

//SlowMist// The return value conforms to the TIP20 specification.

```
    function transfer(address _to, uint _value) public whenNotPaused returns (bool) {
```

//SlowMist// It is recommended to add checks on whether "to" is in the blacklist.

```
        require(!isBlackListed[msg.sender]);
        if (deprecated) {
            return UpgradedStandardToken(upgradedAddress).transferByLegacy(msg.sender, _to, _value);
        } else {
            return super.transfer(_to, _value);
        }
    }

    // Forward ERC20 methods to upgraded contract if this one is deprecated
```

//SlowMist// The return value conforms to the TIP20 specification.

```
    function transferFrom(address _from, address _to, uint _value) public whenNotPaused returns (bool)
{
```

//SlowMist// It is recommended to add checks on whether "msg. sender" and to are in the blacklist.

```
        require(!isBlackListed[_from]);
        if (deprecated) {
```

```
            return UpgradedStandardToken(upgradedAddress).transferFromByLegacy(msg.sender, _from, _to,
_value);
        } else {
            return super.transferFrom(_from, _to, _value);
        }
    }

    // Forward ERC20 methods to upgraded contract if this one is deprecated
    function balanceOf(address who) public constant returns (uint) {
        if (deprecated) {
            return UpgradedStandardToken(upgradedAddress).balanceOf(who);
        } else {
            return super.balanceOf(who);
        }
    }

    // Allow checks of balance at time of deprecation
    function oldBalanceOf(address who) public constant returns (uint) {
        if (deprecated) {
            return super.balanceOf(who);
        }
    }

    // Forward ERC20 methods to upgraded contract if this one is deprecated
    function approve(address _spender, uint _value) public whenNotPaused returns (bool) {
        if (deprecated) {
            return UpgradedStandardToken(upgradedAddress).approveByLegacy(msg.sender, _spender,
_value);
        } else {
            return super.approve(_spender, _value);
        }
    }

    function increaseApproval(address _spender, uint _addedValue) public whenNotPaused returns (bool)
{
        if (deprecated) {
            return UpgradedStandardToken(upgradedAddress).increaseApprovalByLegacy(msg.sender,
_spender, _addedValue);
        } else {
            return super.increaseApproval(_spender, _addedValue);
```

```
        }
    }

    function decreaseApproval(address _spender, uint _subtractedValue) public whenNotPaused returns
(bool) {
        if (deprecated) {
            return UpgradedStandardToken(upgradedAddress).decreaseApprovalByLegacy(msg.sender,
_spender, _subtractedValue);
        } else {
            return super.decreaseApproval(_spender, _subtractedValue);
        }
    }

    // Forward ERC20 methods to upgraded contract if this one is deprecated
    function allowance(address _owner, address _spender) public constant returns (uint remaining) {
        if (deprecated) {
            return StandardToken(upgradedAddress).allowance(_owner, _spender);
        } else {
            return super.allowance(_owner, _spender);
        }
    }
```

//SlowMist// Owner can deprecate current contract in favour of a new one.

```
    function deprecate(address _upgradedAddress) public onlyOwner {
        require(_upgradedAddress != address(0));
        deprecated = true;
        upgradedAddress = _upgradedAddress;
        Deprecate(_upgradedAddress);
    }

    // deprecate current contract if favour of a new one
    function totalSupply() public constant returns (uint) {
        if (deprecated) {
            return StandardToken(upgradedAddress).totalSupply();
        } else {
            return _totalSupply;
        }
    }
```

```
// Issue a new amount of tokens
// these tokens are deposited into the owner address
//
// @param _amount Number of tokens to be issued
```

//SlowMist// Owner can issue a new amount of tokens.

```
function issue(uint amount) public onlyOwner {
    balances[owner] = balances[owner].add(amount);
    _totalSupply = _totalSupply.add(amount);
    Issue(amount);
    Transfer(address(0), owner, amount);
}

// Redeem tokens.
// These tokens are withdrawn from the owner address
// if the balance must be enough to cover the redeem
// or the call will fail.
// @param _amount Number of tokens to be issued
function redeem(uint amount) public onlyOwner {
    _totalSupply = _totalSupply.sub(amount);
    balances[owner] = balances[owner].sub(amount);
    Redeem(amount);
    Transfer(owner, address(0), amount);
}
```

//SlowMist// Owner can destroy black funds from the blacklist address.

```
function destroyBlackFunds (address _blackListedUser) public onlyOwner {
    require(isBlackListed[_blackListedUser]);
    uint dirtyFunds = balanceOf(_blackListedUser);
    balances[_blackListedUser] = 0;
    _totalSupply = _totalSupply.sub(dirtyFunds);
    DestroyedBlackFunds(_blackListedUser, dirtyFunds);
}

event DestroyedBlackFunds(address indexed _blackListedUser, uint _balance);

// Called when new token are issued
event Issue(uint amount);

// Called when tokens are redeemed
```

```
    event Redeem(uint amount);

    // Called when contract is deprecated
    event Deprecate(address newAddress);
    }
```

Migrations.sol

```
pragma solidity ^0.4.4;
/* solhint-disable var-name-mixedcase */

contract Migrations {
    address public owner;
    uint public last_completed_migration;

    modifier restricted() {
        if (msg.sender == owner) _;
    }

    function Migrations() public {
        owner = msg.sender;
    }

    function setCompleted(uint completed) public restricted {
        last_completed_migration = completed;
    }

    function upgrade(address newAddress) public restricted {
        Migrations upgraded = Migrations(newAddress);
        upgraded.setCompleted(last_completed_migration);
    }}
```

MultiSigWallet.sol

```
pragma solidity ^0.4.10;
/// @title Multisignature wallet - Allows multiple parties to agree on transactions before execution.///
@author Stefan George - <stefan.george@consensys.net>contract MultiSigWallet {

    uint constant public MAX_OWNER_COUNT = 50;

    event Confirmation(address indexed sender, uint indexed transactionId);
    event Revocation(address indexed sender, uint indexed transactionId);
```

```
event Submission(uint indexed transactionId);
event Execution(uint indexed transactionId);
event ExecutionFailure(uint indexed transactionId);
event Deposit(address indexed sender, uint value);
event OwnerAddition(address indexed owner);
event OwnerRemoval(address indexed owner);
event RequirementChange(uint required);

mapping (uint => Transaction) public transactions;
mapping (uint => mapping (address => bool)) public confirmations;
mapping (address => bool) public isOwner;
address[] public owners;
uint public required;
uint public transactionCount;

struct Transaction {
    address destination;
    uint value;
    bytes data;
    bool executed;
}

modifier onlyWallet() {
    if (msg.sender != address(this))
        throw;
    _;
}

modifier ownerDoesNotExist(address owner) {
    if (isOwner[owner])
        throw;
    _;
}

modifier ownerExists(address owner) {
    if (!isOwner[owner])
        throw;
    _;
}
```

```
    modifier transactionExists(uint transactionId) {
        if (transactions[transactionId].destination == 0)
            throw;
        _;
    }

    modifier confirmed(uint transactionId, address owner) {
        if (!confirmations[transactionId][owner])
            throw;
        _;
    }

    modifier notConfirmed(uint transactionId, address owner) {
        if (confirmations[transactionId][owner])
            throw;
        _;
    }

    modifier notExecuted(uint transactionId) {
        if (transactions[transactionId].executed)
            throw;
        _;
    }

    modifier notNull(address _address) {
        if (_address == 0)
            throw;
        _;
    }

    modifier validRequirement(uint ownerCount, uint _required) {
        if (   ownerCount > MAX_OWNER_COUNT
            || _required > ownerCount
            || _required == 0
            || ownerCount == 0)
            throw;
        _;
    }

    /// @dev Fallback function allows to deposit ether.
```

```
function()
    payable
{
    if (msg.value > 0)
        Deposit(msg.sender, msg.value);
}

/*     * Public functions     */
/// @dev Contract constructor sets initial owners and required number of confirmations.
/// @param _owners List of initial owners.
/// @param _required Number of required confirmations.
function MultiSigWallet(address[] _owners, uint _required)
    public
    validRequirement(_owners.length, _required)
{
    for (uint i=0; i<_owners.length; i++) {
        if (isOwner[_owners[i]] || _owners[i] == 0)
            throw;
        isOwner[_owners[i]] = true;
    }
    owners = _owners;
    required = _required;
}

/// @dev Allows to add a new owner. Transaction has to be sent by wallet.
/// @param owner Address of new owner.
function addOwner(address owner)
    public
    onlyWallet
    ownerDoesNotExist(owner)
    notNull(owner)
    validRequirement(owners.length + 1, required)
{
    isOwner[owner] = true;
    owners.push(owner);
    OwnerAddition(owner);
}

/// @dev Allows to remove an owner. Transaction has to be sent by wallet.
/// @param owner Address of owner.
```

```
function removeOwner(address owner)
    public
    onlyWallet
    ownerExists(owner)
{
    isOwner[owner] = false;
    for (uint i=0; i<owners.length - 1; i++)
        if (owners[i] == owner) {
            owners[i] = owners[owners.length - 1];
            break;
        }
    owners.length -= 1;
    if (required > owners.length)
        changeRequirement(owners.length);
    OwnerRemoval(owner);
}

/// @dev Allows to replace an owner with a new owner. Transaction has to be sent by wallet.
/// @param owner Address of owner to be replaced.
/// @param owner Address of new owner.
function replaceOwner(address owner, address newOwner)
    public
    onlyWallet
    ownerExists(owner)
    ownerDoesNotExist(newOwner)
{
    for (uint i=0; i<owners.length; i++)
        if (owners[i] == owner) {
            owners[i] = newOwner;
            break;
        }
    isOwner[owner] = false;
    isOwner[newOwner] = true;
    OwnerRemoval(owner);
    OwnerAddition(newOwner);
}

/// @dev Allows to change the number of required confirmations. Transaction has to be sent by wallet.
/// @param _required Number of required confirmations.
function changeRequirement(uint _required)
```

```
    public
    onlyWallet
    validRequirement(owners.length, _required)
{
    required = _required;
    RequirementChange(_required);
}

/// @dev Allows an owner to submit and confirm a transaction.
/// @param destination Transaction target address.
/// @param value Transaction ether value.
/// @param data Transaction data payload.
/// @return Returns transaction ID.
function submitTransaction(address destination, uint value, bytes data)
    public
    returns (uint transactionId)
{
    transactionId = addTransaction(destination, value, data);
    confirmTransaction(transactionId);
}

/// @dev Allows an owner to confirm a transaction.
/// @param transactionId Transaction ID.
function confirmTransaction(uint transactionId)
    public
    ownerExists(msg.sender)
    transactionExists(transactionId)
    notConfirmed(transactionId, msg.sender)
{
    confirmations[transactionId][msg.sender] = true;
    Confirmation(msg.sender, transactionId);
    executeTransaction(transactionId);
}

/// @dev Allows an owner to revoke a confirmation for a transaction.
/// @param transactionId Transaction ID.
function revokeConfirmation(uint transactionId)
    public
    ownerExists(msg.sender)
    confirmed(transactionId, msg.sender)
```

```
    notExecuted(transactionId)
{
    confirmations[transactionId][msg.sender] = false;
    Revocation(msg.sender, transactionId);
}

/// @dev Allows anyone to execute a confirmed transaction.
/// @param transactionId Transaction ID.
function executeTransaction(uint transactionId)
    public
    notExecuted(transactionId)
{
    if (isConfirmed(transactionId)) {
        Transaction tx = transactions[transactionId];
        tx.executed = true;
        if (tx.destination.call.value(tx.value)(tx.data))
            Execution(transactionId);
        else {
            ExecutionFailure(transactionId);
            tx.executed = false;
        }
    }
}

/// @dev Returns the confirmation status of a transaction.
/// @param transactionId Transaction ID.
/// @return Confirmation status.
function isConfirmed(uint transactionId)
    public
    constant
    returns (bool)
{
    uint count = 0;
    for (uint i=0; i<owners.length; i++) {
        if (confirmations[transactionId][owners[i]])
            count += 1;
        if (count == required)
            return true;
    }
}
```

```
/*    * Internal functions    */
/// @dev Adds a new transaction to the transaction mapping, if transaction does not exist yet.
/// @param destination Transaction target address.
/// @param value Transaction ether value.
/// @param data Transaction data payload.
/// @return Returns transaction ID.
function addTransaction(address destination, uint value, bytes data)
    internal
    notNull(destination)
    returns (uint transactionId)
{
    transactionId = transactionCount;
    transactions[transactionId] = Transaction({
        destination: destination,
        value: value,
        data: data,
        executed: false
    });
    transactionCount += 1;
    Submission(transactionId);
}

/*    * Web3 call functions    */
/// @dev Returns number of confirmations of a transaction.
/// @param transactionId Transaction ID.
/// @return Number of confirmations.
function getConfirmationCount(uint transactionId)
    public
    constant
    returns (uint count)
{
    for (uint i=0; i<owners.length; i++)
        if (confirmations[transactionId][owners[i]])
            count += 1;
}

/// @dev Returns total number of transactions after filers are applied.
/// @param pending Include pending transactions.
/// @param executed Include executed transactions.
```

```
    /// @return Total number of transactions after filters are applied.
    function getTransactionCount(bool pending, bool executed)
        public
        constant
        returns (uint count)
    {
        for (uint i=0; i<transactionCount; i++)
            if (   pending && !transactions[i].executed
                || executed && transactions[i].executed)
                count += 1;
    }

    /// @dev Returns list of owners.
    /// @return List of owner addresses.
    function getOwners()
        public
        constant
        returns (address[])
    {
        return owners;
    }

    /// @dev Returns array with owner addresses, which confirmed transaction.
    /// @param transactionId Transaction ID.
    /// @return Returns array of owner addresses.
    function getConfirmations(uint transactionId)
        public
        constant
        returns (address[] _confirmations)
    {
        address[] memory confirmationsTemp = new address[](owners.length);
        uint count = 0;
        uint i;
        for (i=0; i<owners.length; i++)
            if (confirmations[transactionId][owners[i]]) {
                confirmationsTemp[count] = owners[i];
                count += 1;
            }
        _confirmations = new address[](count);
        for (i=0; i<count; i++)
```

```
        _confirmations[i] = confirmationsTemp[i];
    }

    /// @dev Returns list of transaction IDs in defined range.
    /// @param from Index start position of transaction array.
    /// @param to Index end position of transaction array.
    /// @param pending Include pending transactions.
    /// @param executed Include executed transactions.
    /// @return Returns array of transaction IDs.
    function getTransactionIds(uint from, uint to, bool pending, bool executed)
        public
        constant
        returns (uint[] _transactionIds)
    {
        uint[] memory transactionIdsTemp = new uint[](transactionCount);
        uint count = 0;
        uint i;
        for (i=0; i<transactionCount; i++)
            if (   pending && !transactions[i].executed
                || executed && transactions[i].executed)
            {
                transactionIdsTemp[count] = i;
                count += 1;
            }
        _transactionIds = new uint[](to - from);
        for (i=from; i<to; i++)
            _transactionIds[i - from] = transactionIdsTemp[i];
    }}
```

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist