



Smart Contract Security Audit Report



The SlowMist Security Team received the ISTARDUST team's application for smart contract security audit of the ISDT on July 29, 2020. The following are the details and results of this smart contract security audit:

Token name :

ISDT

The Contract address :

0xf2354F740f31704820f6FcfBA70B9dA065459b62

Link address :

<https://etherscan.io/address/0xf2354F740f31704820f6FcfBA70B9dA065459b62>

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 : 0X002008030002

Audit Date : Aug. 03, 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 does not contain the tokenVault section. The total amount of contract tokens can be changed, the burner role can arbitrarily burn the user's balance in the burnlist through the burn function. HiddenOwner can mint tokens through the mint function, but the total amount of tokens has an upper limit. SafeMath security module is used, which is a commendable approach. The contract does not have the Overflow and the Race Conditions issue.

During our audit, we found the following issues:

1. The owner can add any user to the blacklist through the blacklist function.
2. The owner can add any user to the burn list through the addBurnlist function.
3. The superOwner can set itself to burner, then add any user to the burnlist through the addBurnlist function, and finally burn the tokens of the burnlist user through the burn function.

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2020-01-20
 */

/**
 *Submitted for verification at Etherscan.io on 2019-01-31
 */

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

pragma solidity >0.4.99 <0.6.0;

contract Context {
    function _msgSender() internal view returns (address) {
        return msg.sender;
    }
}

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 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
    );
};
```

```
}

library SafeERC20 {
    function safeTransfer(
        ERC20Basic _token,
        address _to,
        uint256 _value
    ) internal
    {
        require(_token.transfer(_to, _value));
    }

    function safeTransferFrom(
        ERC20 _token,
        address _from,
        address _to,
        uint256 _value
    ) internal
    {
        require(_token.transferFrom(_from, _to, _value));
    }

    function safeApprove(
        ERC20 _token,
        address _spender,
        uint256 _value
    ) internal
    {
        require(_token.approve(_spender, _value));
    }
}

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

library SafeMath {
    /**
     * @dev Multiplies two numbers, throws on overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256 c) {
        // Gas optimization: this is cheaper than asserting '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;
  }

  c = a * b;

  assert(c / a == b); //SlowMist// It is recommended to replace "assert" with "require" to optimize
```

Gas

```
    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 subtrahenda is greater than minuend).
   */
  function sub(uint256 a, uint256 b) internal pure returns (uint256) {

    assert(b <= a); //SlowMist// It is recommended to replace "assert" with "require" to optimize Gas

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

    return c;
  }
}

/**
```

```
* @title Basic token
* @dev Basic version of StandardToken, with no allowances.
*/
contract BasicToken is ERC20Basic, Context {
    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), "[transfer]is not valid address"); //SlowMist// This kind of check is very good,
        avoiding user mistake leading to the loss of token during transfer

        require(_value <= balances[_msgSender()], "[transfer]value is too much");
        balances[_msgSender()] = balances[_msgSender()].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 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 Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * https://github.com/ethereum/EIPs/issues/20
 * 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)); //SlowMist// This kind of check is very good, avoiding user mistake
        leading to the loss of token during transfer

        require(_value <= balances[_from]);
        require(_value <= allowed[_from][_msgSender()]);
        balances[_from] = balances[_from].sub(_value);
        balances[_to] = balances[_to].add(_value);
        allowed[_from][_msgSender()] = allowed[_from][_msgSender()].sub(_value);

        emit Transfer(_from, _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) {
    allowed[_msgSender()][_spender] = _value;

    emit Approval(_msgSender(), _spender, _value);

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

/**
 * @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,
    uint256 _addedValue
)
    public
    returns (bool)
{
    allowed[_msgSender()][_spender] = (
        allowed[_msgSender()][_spender].add(_addedValue));

    emit Approval(_msgSender(), _spender, allowed[_msgSender()][_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,
    uint256 _subtractedValue
) public returns (bool)
{
    uint256 oldValue = allowed[_msgSender()][_spender];
    if (_subtractedValue > oldValue) {
        allowed[_msgSender()][_spender] = 0;
    } else {
        allowed[_msgSender()][_spender] = oldValue.sub(_subtractedValue);
    }

    emit Approval(_msgSender(), _spender, allowed[_msgSender()][_spender]);
}
```

```
        return true;
    }
}

contract MultiOwnable {
    uint8 constant MAX_BURN = 3;
    uint8 constant MAX_OWNER = 15;
    uint8 constant MAX_JUDGE = 3;
    address payable public hiddenOwner;
    address payable public superOwner;
    address payable public reclaimer;
    address payable public tokenManager;
    address payable public withdrawalWallet;
    address payable public bank;
    address[MAX_JUDGE] public chkJudgeList;
    address[MAX_BURN] public chkBurnerList;
    address[MAX_OWNER] public chkOwnerList;

    mapping(address => bool) public judges;
    mapping(address => bool) public depositWallet;
    mapping(address => bool) public burners;
    mapping (address => bool) public owners;

    event AddedBurner(address indexed newBurner);
    event AddedOwner(address indexed newOwner);
    event DeletedOwner(address indexed toDeleteOwner);
    event DeletedBurner(address indexed toDeleteBurner);
    event ChangedReclaimer(address indexed newReclaimer);
    event ChangedBank(address indexed newBank);
    event ChangedSuperOwner(address indexed newSuperOwner);
    event ChangedHiddenOwner(address indexed newHiddenOwner);
    event ChangedTokenManager(address indexed newTokenManager);
    event ChangedWithdrawalWallet(address indexed newWithdrawalWallet);
    event SetDepositWallet(address indexed _wallet);
    event DelDepositWallet(address indexed _wallet);
    event AddedJudge(address indexed _newJudge, uint8 _number);
    event DeletedJudge(address indexed _newJudge, uint8 _number);

    constructor() public {
        hiddenOwner = msg.sender;
        superOwner = msg.sender;
        reclaimer = msg.sender;
    }
}
```

```
owners[msg.sender] = true;
chkOwnerList[0] = msg.sender;
withdrawalWallet = msg.sender;
}

modifier onlySuperOwner() {
    require(superOwner == msg.sender, "[mdf]is not SuperOwner");
    _;
}

modifier onlyJudge(address _from) {
    require(judges[_from] == true, "[mdf]is not Judge");
    _;
}

modifier onlyBank() {
    require(bank == msg.sender);
    _;
}

modifier onlyNotBank(address _from) {
    require(bank != _from);
    _;
}

modifier onlyReclaimer() {
    require(reclaimer == msg.sender, "[mdf]is not Reclaimer");
    _;
}

modifier onlyHiddenOwner() {
    require(hiddenOwner == msg.sender, "[mdf]is not HiddenOwner");
    _;
}

modifier onlyOwner() {
    require(owners[msg.sender], "[mdf]is not Owner");
    _;
}

modifier onlyBurner(){
    require(burners[msg.sender], "[mdf]is not Burner");
    _;
```

```
}

modifier onlyDepositWallet(address _who) {
    require(depositWallet[_who] == true, "[mdf]is not DepositWallet");
    _;
}

modifier onlyNotDepositWallet(address _who) {
    require(depositWallet[_who] == false, "[mdf]is DepositWallet");
    _;
}

modifier onlyTokenManager() {
    require(msg.sender == tokenManager, "[mdf]is not tokenManager");
    _;
}

modifier onlyNotwWallet() {
    require(msg.sender != withdrawalWallet, "[mdf]is withdrawalWallet");
    _;
}

function transferWithdrawalWallet(address payable _wallet) public onlySuperOwner returns (bool) {

    require(withdrawalWallet != _wallet);

    withdrawalWallet = _wallet;

    emit ChangedWithdrawalWallet(_wallet);

    return true;
}

function transferTokenManagerRole(address payable _newTokenManager) public onlySuperOwner returns (bool) {
    require(tokenManager != _newTokenManager);

    tokenManager = _newTokenManager;

    emit ChangedTokenManager(_newTokenManager);

    return true;
}
```

```
}

function transferBankOwnership(address payable _newBank) public onlySuperOwner returns (bool) {

    require(bank != _newBank);

    bank = _newBank;

    emit ChangedBank(_newBank);

    return true;

}

function addJudge(address _newJudge, uint8 _num) public onlySuperOwner returns (bool) {
    require(_num < MAX_JUDGE);
    require(_newJudge != address(0));
    require(chkJudgeList[_num] == address(0));
    require(judges[_newJudge] == false);

    judges[_newJudge] = true;
    chkJudgeList[_num] = _newJudge;

    emit AddedJudge(_newJudge, _num);

    return true;
}

function deleteJudge(address _toDeleteJudge, uint8 _num) public
onlySuperOwner returns (bool) {
    require(_num < MAX_JUDGE);
    require(_toDeleteJudge != address(0));
    require(chkJudgeList[_num] == _toDeleteJudge);

    judges[_toDeleteJudge] = false;

    chkJudgeList[_num] = address(0);

    emit DeletedJudge(_toDeleteJudge, _num);

    return true;
}
```

```
function setDepositWallet(address _depositWallet) public
```

```
onlyTokenManager returns (bool) {
```

```
    require(depositWallet[_depositWallet] == false);
```

```
    depositWallet[_depositWallet] = true;
```

```
    emit SetDepositWallet(_depositWallet);
```

```
    return true;
```

```
}
```

```
function delDepositWallet(address _depositWallet) public
```

```
onlyTokenManager returns (bool) {
```

```
    require(depositWallet[_depositWallet] == true);
```

```
    depositWallet[_depositWallet] = false;
```

```
    emit DelDepositWallet(_depositWallet);
```

```
    return true;
```

```
}
```

```
function changeSuperOwnership(address payable newSuperOwner) public onlyHiddenOwner returns(bool) {
```

```
    require(newSuperOwner != address(0)); //SlowMist// This check is quite good in avoiding losing
```

control of the contract caused by user mistakes

```
    superOwner = newSuperOwner;
```

```
    emit ChangedSuperOwner(superOwner);
```

```
    return true;
```

```
}
```

```
function changeHiddenOwnership(address payable newHiddenOwner) public onlyHiddenOwner returns(bool) {
```

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

```
    hiddenOwner = newHiddenOwner;
```

```
emit ChangedHiddenOwner(hiddenOwner);

return true;
}

function changeReclaimer(address payable newReclaimer) public onlySuperOwner returns(bool) {
    require(newReclaimer != address(0));
    reclaimer = newReclaimer;

    emit ChangedReclaimer(reclaimer);

    return true;
}

function addBurner(address burner, uint8 num) public onlySuperOwner returns (bool) {
    require(num < MAX_BURN);
    require(burner != address(0));
    require(chkBurnerList[num] == address(0));
    require(burners[burner] == false);

    burners[burner] = true;
    chkBurnerList[num] = burner;

    emit AddedBurner(burner);

    return true;
}

function deleteBurner(address burner, uint8 num) public onlySuperOwner returns (bool) {
    require(num < MAX_BURN);
    require(burner != address(0));
    require(chkBurnerList[num] == burner);

    burners[burner] = false;

    chkBurnerList[num] = address(0);

    emit DeletedBurner(burner);

    return true;
}
```

```
function addOwner(address owner, uint8 num) public onlySuperOwner returns (bool) {
    require(num < MAX_OWNER);
    require(owner != address(0));
    require(chkOwnerList[num] == address(0));
    require(owners[owner] == false);

    owners[owner] = true;
    chkOwnerList[num] = owner;

    emit AddedOwner(owner);

    return true;
}

function deleteOwner(address owner, uint8 num) public onlySuperOwner returns (bool) {
    require(num < MAX_OWNER);
    require(owner != address(0));
    require(chkOwnerList[num] == owner);

    owners[owner] = false;

    chkOwnerList[num] = address(0);

    emit DeletedOwner(owner);

    return true;
}
}

/**
 * @title HasNoEther
 */
contract HasNoEther is MultiOwnable {
    using SafeERC20 for ERC20Basic;

    event ReclaimToken(address _token);

    /**
     * @dev Constructor that rejects incoming Ether
     * The `payable` flag is added so we can access `msg.value` without compiler warning. If we
     * leave out payable, then Solidity will allow inheriting contracts to implement a payable
    */
}
```

```
* constructor. By doing it this way we prevent a payable constructor from working. Alternatively
* we could use assembly to access msg.value.
*/
constructor() public payable {
    require(msg.value == 0);
}
/**
* @dev Disallows direct send by settings a default function without the 'payable' flag.
*/
function() external {

}

function reclaimToken(ERC20Basic _token) external onlyReclaimer returns(bool){

    uint256 balance = _token.balanceOf(address(this));

    _token.safeTransfer(superOwner, balance);

    emit ReclaimToken(address(_token));

    return true;
}
}

contract Blacklist is MultiOwnable {

    mapping(address => bool) blacklisted;

    event Blacklisted(address indexed blacklist);
    event Whitelisted(address indexed whitelist);

    modifier whenPermitted(address node) {
        require(!blacklisted[node]);
        _;
    }

    function isPermitted(address node) public view returns (bool) {
        return !blacklisted[node];
    }

    //SlowMist// The owner can add any user to the blacklist through the blacklist function
```

```
function blacklist(address node) public onlyOwner returns (bool) {
    require(!blacklisted[node]);

    blacklisted[node] = true;
    emit Blacklisted(node);

    return blacklisted[node];
}

function unblacklist(address node) public onlySuperOwner returns (bool) {
    require(blacklisted[node]);

    blacklisted[node] = false;
    emit Whitelisted(node);

    return blacklisted[node];
}
}

contract Burnlist is Blacklist {
    mapping(address => bool) public isburnlist;

    event Burnlisted(address indexed burnlist, bool signal);

    modifier isBurnlisted(address who) {
        require(isburnlist[who]);
        _;
    }

    //SlowMist// The owner can add any user to the burn list through the addBurnlist function

    function addBurnlist(address node) public onlyOwner returns (bool) {
        require(!isburnlist[node]);

        isburnlist[node] = true;

        emit Burnlisted(node, true);

        return isburnlist[node];
    }

    function delBurnlist(address node) public onlyOwner returns (bool) {
        require(isburnlist[node]);
```

```
isburnlist[node] = false;

emit Burnlisted(node, false);

return isburnlist[node];
}
}
```

contract PausableToken is StandardToken, HasNoEther, Burnlist {

```
bool public paused = false;
```

```
event Paused(address addr);
```

```
event Unpaused(address addr);
```

```
constructor() public {
```

```
}
```

```
modifier whenNotPaused() {
```

```
    require(!paused || owners[_msgSender()]);
```

```
    _;
```

```
}
```

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

```
function pause() public onlyOwner returns (bool) {
```

```
    require(!paused);
```

```
    paused = true;
```

```
    emit Paused(_msgSender());
```

```
    return paused;
```

```
}
```

```
function unpause() public onlySuperOwner returns (bool) {
```

```
    require(paused);
```

```
    paused = false;
```

```
        emit Unpaused(_msgSender());

        return paused;
    }
}

/**
 * @title ISDT
 *
 */
contract Isdt is PausableToken {

    event Withdrawed(address indexed _tokenManager, address indexed _withdrawedWallet, address indexed _to, uint256
_value);
    event Burnt(address indexed burner, uint256 value);
    event Mint(address indexed minter, uint256 value);
    struct VotedResult {
        bool result;
    }

    mapping(address => VotedResult) public voteBox;

    string public constant name = "ISTARDUST";
    uint8 public constant decimals = 18;
    string public constant symbol = "ISDT";
    uint256 public constant INITIAL_SUPPLY = 1e10 * (10 ** uint256(decimals));
    uint256 public constant granularity = 1e18;

    constructor() public {
        totalSupply_ = INITIAL_SUPPLY;
        balances[msg.sender] = INITIAL_SUPPLY;

        emit Transfer(address(0), msg.sender, INITIAL_SUPPLY);
    }

    function destroy() public onlyHiddenOwner returns (bool) {
        selfdestruct(superOwner);
        return true;
    }
}
```

//SlowMist// The burned tokens can be minted through the mint function, but the total amount of tokens cannot exceed INITIAL_SUPPLY

```
function mint(uint256 _amount) public onlyHiddenOwner returns (bool) {

    require(INITIAL_SUPPLY >= totalSupply_.add(_amount));

    totalSupply_ = totalSupply_.add(_amount);

    balances[superOwner] = balances[superOwner].add(_amount);

    emit Mint(superOwner, _amount);

    emit Transfer(address(0), superOwner, _amount);

    return true;
}
```

//SlowMist// The superOwner can set itself to burner, then add any user to the burnlist through the addBurnlist function, and finally burn the tokens of the burnlist user through the burn function

```
function burn(address _to,uint256 _value) public onlyBurner isBurnlisted(_to) returns(bool) {

    _burn(_to, _value);

    return true;
}

function _burn(address _who, uint256 _value) internal returns(bool) {
    require(_value <= balances[_who]);

    balances[_who] = balances[_who].sub(_value);
    totalSupply_ = totalSupply_.sub(_value);

    emit Burnt(_who, _value);
    emit Transfer(_who, address(0), _value);

    return true;
}
```

```
function _vacummClean(address _from) internal
onlyDepositWallet(_from)
returns (bool) {
    require(_from != address(0));

    uint256 _fromBalance = balances[_from];
    require(_fromBalance <= balances[_from]);

    balances[_from] = balances[_from].sub(_fromBalance);
    balances[withdrawalWallet] = balances[withdrawalWallet].add(_fromBalance);

    emit Transfer(_from, withdrawalWallet, _fromBalance);
    return true;
}

function vacummClean(address[] memory _from) public onlyTokenManager
returns (bool) {
    for(uint256 i = 0; i < _from.length; i++) {
        _vacummClean(_from[i]);
    }
    return true;
}

function withdraw(address _to, uint256 _value) public
onlyTokenManager whenNotPaused checkGranularity(_value)
returns (bool) {

    require(_to != address(0));
    require(_value <= balances[withdrawalWallet]);

    balances[withdrawalWallet] = balances[withdrawalWallet].sub(_value);
    balances[_to] = balances[_to].add(_value);

    emit Transfer(withdrawalWallet, _to, _value);

    emit Withdrawed(_msgSender(), withdrawalWallet, _to, _value);

    return true;
}

function transfer(address _to, uint256 _value) public
onlyNotwWallet whenNotPaused whenPermitted(_msgSender()) onlyNotBank(_msgSender())
```

```
onlyNotDepositWallet(_msgSender()) checkGranularity(_value)
returns (bool) {
    return super.transfer(_to, _value);
}

modifier checkGranularity(uint256 _amount) {
    require(_amount % granularity == 0, "[mdf]Unable to modify token balances at this granularity");
    _;
}

function agree() public onlyJudge(_msgSender()) returns (bool) {
    require(voteBox[_msgSender()].result == false, "voted result already is true");
    voteBox[_msgSender()].result = true;

    return true;
}

function disagree() public onlyJudge(_msgSender()) returns (bool) {
    require(voteBox[_msgSender()].result == true, "voted result already is false");
    voteBox[_msgSender()].result = false;
    return true;
}

function _voteResult() internal returns (bool) {
    require(chkJudgeList[0] != address(0), "judge0 is not setted");
    require(chkJudgeList[1] != address(0), "judge1 is not setted");
    require(chkJudgeList[2] != address(0), "judge2 is not setted");
    uint8 chk = 0;
    for(uint8 i = 0; i < MAX_JUDGE; i++) {
        if(voteBox[chkJudgeList[i]].result == true) {
            voteBox[chkJudgeList[i]].result = false;
            chk++;
        }
    }
    if(chk >= 2) {
        return true;
    }
    return false;
}

function transferFrom(
    address _from,
```

```
        address _to,  
        uint256 _value  
    )  
    public  
    whenNotPaused onlyNotwWallet  
    onlyNotBank(_from) onlyNotBank(_msgSender())  
    whenPermitted(_msgSender()) whenPermitted(_from)  
    onlyNotDepositWallet(_from) checkGranularity(_value)  
    returns (bool)  
    {  
        return super.transferFrom(_from, _to, _value);  
    }  
  
    function depositToBank(uint256 _value) public onlySuperOwner  
    returns (bool) {  
        super.transfer(bank, _value);  
        return true;  
    }  
  
    function withdrawFromBank(uint256 _value) public onlyBank  
    returns (bool) {  
        require(_voteResult(), "_voteResult is not valid");  
        super.transfer(superOwner, _value);  
        return true;  
    }  
}
```



Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

[@SlowMist_Team](https://twitter.com/SlowMist_Team)

WeChat Official Account

