Smart Contract Security Audit Report

The SlowMist Security Team received the The Midas Touch Gold team's application for smart contract security audit of the TMTG on May 23, 2020. The following are the details and results of this smart contract security audit:

Token name :

TMTG

The Contract address :

0x10086399dd8c1e3de736724af52587a2044c9fa2

Link address :

https://etherscan.io/address/0x10086399dd8c1e3de736724af52587a2044c9fa2

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	-	Not 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 : **Not Passed**

Audit Number : 0X002005270002

Audit Date : May 27, 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, owner can burn his own tokens through the burn function. SafeMath security module is used, which is a commendable approach. The contract does not have the Overflow and the Race Conditions issue.

During the audit, we found some issues:

1. Owner or operator can add any user to the blacklist through the blacklist function.

2. The visibility of the _transferFromInvestor function is `public`, which results in that the user could still manipulate the balance of the `investor` included in the blacklist.

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2018-07-22
*/
```

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

```
pragma solidity ^0.4.24;


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);
}
```

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

        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; //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][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; //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[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _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[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,
        uint256 _subtractedValue
    )
    public
    returns (bool)
    {
        uint256 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;
        }
    }

    /**
     * @title TMTGOwnable
     *
     * @dev Due to ownable change in zeppelin, the authorities in TMTGOwnable include hiddenOwner,
     *          superOwner, owner, centerBanker, and operator.
     *          Each authority has different roles.
     */
    contract TMTGOwnable {
        address public owner;
        address public centralBanker;
        address public superOwner;
        address public hiddenOwner;

        enum Role { owner, centralBanker, superOwner, hiddenOwner }

        mapping(address => bool) public operators;


        event TMTG_RoleTransferred(
            Role indexed ownerType,
            address indexed previousOwner,
            address indexed newOwner
        );

        event TMTG_SetOperator(address indexed operator);
        event TMTG_DeletedOperator(address indexed operator);

        modifier onlyOwner() {
            require(msg.sender == owner);
            _;
        }

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

        modifier onlyNotBankOwner(){
```

```
        require(msg.sender != centralBanker);
        _;
    }

    modifier onlyBankOwner(){
        require(msg.sender == centralBanker);
        _;
    }

    modifier onlySuperOwner() {
        require(msg.sender == superOwner);
        _;
    }

    modifier onlyhiddenOwner(){
        require(msg.sender == hiddenOwner);
        _;
    }

    constructor() public {
        owner = msg.sender;
        centralBanker = msg.sender;
        superOwner = msg.sender;
        hiddenOwner = msg.sender;
    }

    /**
    * @dev Set the address to operator
    * @param _operator has the ability to pause transaction, has the ability to blacklisting & unblacklisting.
    */
    function setOperator(address _operator) external onlySuperOwner {
        operators[_operator] = true;
        emit TMTG_SetOperator(_operator);
    }

    /**
    * @dev Remove the address from operator
    * @param _operator has the ability to pause transaction, has the ability to blacklisting & unblacklisting.
    */
    function delOperator(address _operator) external onlySuperOwner {
        operators[_operator] = false;
        emit TMTG_DeletedOperator(_operator);
```

```
    }

    /**
    * @dev It is possible to hand over owner’s authority. Only superowner is available.
    * @param newOwner
    */
    function transferOwnership(address newOwner) public onlySuperOwner {
        emit TMTG_RoleTransferred(Role.owner, owner, newOwner);
        owner = newOwner;
    }

    /**
    * @dev It is possible to hand over centerBanker’s authority. Only superowner is available.
    * @param newBanker centerBanker is a kind of a central bank, transaction is impossible.
    * The amount of money to deposit is determined in accordance with cash reserve ratio and the amount of currency in circulation
    * To withdraw money out of a wallet and give it to owner, audit is inevitable
    */
    function transferBankOwnership(address newBanker) public onlySuperOwner {
        emit TMTG_RoleTransferred(Role.centralBanker, centralBanker, newBanker);
        centralBanker = newBanker;
    }

    /**
    * @dev It is possible to hand over superOwner’s authority. Only hiddenowner is available.
    * @param newSuperOwner  SuperOwner manages all authorities except for hiddenOwner and superOwner
    */
    function transferSuperOwnership(address newSuperOwner) public onlyhiddenOwner {
        emit TMTG_RoleTransferred(Role.superOwner, superOwner, newSuperOwner);
        superOwner = newSuperOwner;
    }

    /**
    * @dev It is possible to hand over hiddenOwner’s authority. Only hiddenowner is available
    * @param newhiddenOwner NewhiddenOwner and hiddenOwner don’t have many functions,
    * but they can set and remove authorities of superOwner and hiddenOwner.
    */
    function changeHiddenOwner(address newhiddenOwner) public onlyhiddenOwner {
        emit TMTG_RoleTransferred(Role.hiddenOwner, hiddenOwner, newhiddenOwner);
        hiddenOwner = newhiddenOwner;
    }
}
```

```
/**
 * @title TMTGPausable
 *
 * @dev It is used to stop trading in emergency situation
 */
contract TMTGPausable is TMTGOwnable {
    event TMTG_Pause();
    event TMTG_Unpause();

    bool public paused = false;

    modifier whenNotPaused() {
        require(!paused);
        _;
    }

    modifier whenPaused() {
        require(paused);
        _;
    }
    /**
    * @dev Block trading. Only owner and operator are available.
    */
```

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

```
    function pause() onlyOwnerOrOperator whenNotPaused public {
        paused = true;
        emit TMTG_Pause();
    }

    /**
    * @dev Unlock limit for trading. Owner and operator are available and this function can be operated in paused mode.
    */
    function unpause() onlyOwnerOrOperator whenPaused public {
        paused = false;
        emit TMTG_Unpause();
    }
}
```

```
/**
 * @title TMTGBlacklist
 *
 * @dev Block trading of the suspicious account address.
 */
contract TMTGBlacklist is TMTGOwnable {
    mapping(address => bool) blacklisted;

    event TMTG_Blacklisted(address indexed blacklist);
    event TMTG_Whitelisted(address indexed whitelist);

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

    /**
     * @dev Check a certain node is in a blacklist
     * @param node  Check whether the user at a certain node is in a blacklist
     */
    function isPermitted(address node) public view returns (bool) {
        return !blacklisted[node];
    }

    /**
     * @dev Process blacklisting
     * @param node Process blacklisting. Put the user in the blacklist.
     */
    function blacklist(address node) public onlyOwnerOrOperator {
        blacklisted[node] = true;
        emit TMTG_Blacklisted(node);
    }

    /**
     * @dev Process unBlacklisting.
     * @param node Remove the user from the blacklist.
     */
    function unblacklist(address node) public onlyOwnerOrOperator {
        blacklisted[node] = false;
        emit TMTG_Whitelisted(node);
    }
}
```

```
/**
 * @title HasNoEther
 */
contract HasNoEther is TMTGOwnable {

    /**
   * @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 {
    }

    /**
   * @dev Transfer all Ether held by the contract to the owner.
   */
    function reclaimEther() external onlyOwner {
        owner.transfer(address(this).balance);
    }
}

/**
 * @title TMTGBaseToken - Major functions such as authority setting on tockenlock are registered.
 */
contract TMTGBaseToken is StandardToken, TMTGPausable, TMTGBlacklist, HasNoEther {
    uint256 public openingTime;

    struct investor {
        uint256 _sentAmount;
        uint256 _initialAmount;
        uint256 _limit;
    }
```

```
mapping(address => investor) public searchInvestor;
mapping(address => bool) public superInvestor;
mapping(address => bool) public CEx;
mapping(address => bool) public investorList;

event TMTG_SetCEx(address indexed CEx);
event TMTG_DeleteCEx(address indexed CEx);

event TMTG_SetSuperInvestor(address indexed SuperInvestor);
event TMTG_DeleteSuperInvestor(address indexed SuperInvestor);

event TMTG_SetInvestor(address indexed investor);
event TMTG_DeleteInvestor(address indexed investor);

event TMTG_Stash(uint256 _value);
event TMTG_Unstash(uint256 _value);

event TMTG_TransferFrom(address indexed owner, address indexed spender, address indexed to, uint256 value);
event TMTG_Burn(address indexed burner, uint256 value);

/**
* @dev Register the address as a cex address
* @param _CEx   Register the address as a cex address
*/
function setCEx(address _CEx) external onlySuperOwner {
    CEx[_CEx] = true;

    emit TMTG_SetCEx(_CEx);
}

/**
* @dev Remove authorities of the address used in Exchange
* @param _CEx Remove authorities of the address used in Exchange
*/
function delCEx(address _CEx) external onlySuperOwner {
    CEx[_CEx] = false;

    emit TMTG_DeleteCEx(_CEx);
}

/**
```

```
    * @dev Register the address as a superinvestor address
    * @param _super Register the address as a superinvestor address
    */
    function setSuperInvestor(address _super) external onlySuperOwner {
        superInvestor[_super] = true;

        emit TMTG_SetSuperInvestor(_super);
    }

    /**
    * @param _super Remove authorities of the address as a superinvestor
    */
    function delSuperInvestor(address _super) external onlySuperOwner {
        superInvestor[_super] = false;

        emit TMTG_DeleteSuperInvestor(_super);
    }

    /**
    * @param _addr   Remove authorities of the address as a investor .
    */
    function delInvestor(address _addr) onlySuperOwner public {
        investorList[_addr] = false;
        searchInvestor[_addr] = investor(0,0,0);
        emit TMTG_DeleteInvestor(_addr);
    }

    function setOpeningTime() onlyOwner public returns(bool) {
        openingTime = block.timestamp;

    }

    /**
    * @dev   After one month, the amount will be 1, which means 10% of the coins can be used.
    * After 7 months, 70% of the amount can be used.
    */
    function getLimitPeriod() external view returns (uint256) {
        uint256 presentTime = block.timestamp;
        uint256 timeValue = presentTime.sub(openingTime);
        uint256 result = timeValue.div(31 days);
        return result;
    }
```

```
/**
* @dev Check the latest limit
* @param who Check the latest limit.
* Return the limit value of the user at the present moment. After 3 months, _result value will be 30% of initialAmount
*/
function _timelimitCal(address who) internal view returns (uint256) {
    uint256 presentTime = block.timestamp;
    uint256 timeValue = presentTime.sub(openingTime);
    uint256 _result = timeValue.div(31 days);

    return _result.mul(searchInvestor[who]._limit);
}

/**
* @dev In case of investor transfer, values will be limited by timelock
* @param _to address to send
* @param _value tmtg's amount
*/
function _transferInvestor(address _to, uint256 _value) internal returns (bool ret) {
    uint256 addedValue = searchInvestor[msg.sender]._sentAmount.add(_value);

    require(_timelimitCal(msg.sender) >= addedValue);

    searchInvestor[msg.sender]._sentAmount = addedValue;
    ret = super.transfer(_to, _value);
    if (!ret) {
    searchInvestor[msg.sender]._sentAmount = searchInvestor[msg.sender]._sentAmount.sub(_value);
    }
}

/**
* @dev When the transfer function is run,
* there are two different types; transfer from superinvestors to investor and to non-investors.
* In the latter case, the non-investors will be investor and 10% of the initial amount will be allocated.
* And when investor operates the transfer function, the values will be limited by timelock.
* @param _to address to send
* @param _value tmtg's amount
*/
function transfer(address _to, uint256 _value) public
whenPermitted(msg.sender) whenPermitted(_to) whenNotPaused onlyNotBankOwner
returns (bool) {
```

```
        if(investorList[msg.sender]) {
            return _transferInvestor(_to, _value);

        } else {
            if (superInvestor[msg.sender]) {
                require(_to != owner);
                require(!superInvestor[_to]);
                require(!CEx[_to]);

                if(!investorList[_to]){
                    investorList[_to] = true;
                    searchInvestor[_to] = investor(0, _value, _value.div(10));
                    emit TMTG_SetInvestor(_to);
                }
            }
            return super.transfer(_to, _value);
        }
    }

    /**
    * @dev If investor is from in transforFrom, values will be limited by timelock
    * @param _from send amount from this address
    * @param _to address to send
    * @param _value tmtg's amount
    */
```

//SlowMist// The visibility of this function is `public`, which results in that the user could still manipulate the balance of the `investor` included in the blacklist

//SlowMist// It is recommend changing visibility to `internal`

```
    function _transferFromInvestor(address _from, address _to, uint256 _value)
    public returns(bool ret) {
        uint256 addedValue = searchInvestor[_from]._sentAmount.add(_value);
        require(_timelimitCal(_from) >= addedValue);
        searchInvestor[_from]._sentAmount = addedValue;
        ret = super.transferFrom(_from, _to, _value);

        if (!ret) {
            searchInvestor[_from]._sentAmount = searchInvestor[_from]._sentAmount.sub(_value);
        }else {
```

```
            emit TMTG_TransferFrom(_from, msg.sender, _to, _value);
        }
    }

    /**
    * @dev If from is superinvestor in transforFrom, the function can’t be used because of limit in Approve.
    * And if from is investor, the amount of coins to send is limited by timelock.
    * @param _from send amount from this address
    * @param _to address to send
    * @param _value tmtg's amount
    */
    function transferFrom(address _from, address _to, uint256 _value)
    public whenNotPaused whenPermitted(_from) whenPermitted(_to) whenPermitted(msg.sender)
    returns (bool ret)
    {
        if(investorList[_from]) {
            return _transferFromInvestor(_from, _to, _value);
        } else {
            ret = super.transferFrom(_from, _to, _value);
            emit TMTG_TransferFrom(_from, msg.sender, _to, _value);
        }
    }

    function approve(address _spender, uint256 _value) public
    whenPermitted(msg.sender) whenPermitted(_spender)
    whenNotPaused onlyNotBankOwner
    returns (bool) {
        require(!superInvestor[msg.sender]);
        return super.approve(_spender,_value);
    }

    function increaseApproval(address _spender, uint256 _addedValue) public
    whenNotPaused onlyNotBankOwner
    whenPermitted(msg.sender) whenPermitted(_spender)
    returns (bool) {
        require(!superInvestor[msg.sender]);
        return super.increaseApproval(_spender, _addedValue);
    }

    function decreaseApproval(address _spender, uint256 _subtractedValue) public
    whenNotPaused onlyNotBankOwner
    whenPermitted(msg.sender) whenPermitted(_spender)
```

```
returns (bool) {
    require(!superInvestor[msg.sender]);
    return super.decreaseApproval(_spender, _subtractedValue);
}

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

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

    emit Transfer(_who, address(0), _value);
    emit TMTG_Burn(_who, _value);
}

function burn(uint256 _value) onlyOwner public returns (bool) {
    _burn(msg.sender, _value);
    return true;
}

function burnFrom(address _from, uint256 _value) onlyOwner public returns (bool) {
    require(_value <= allowed[_from][msg.sender]);

    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
    _burn(_from, _value);

    return true;
}

/**
* @dev onlyOwner is available and the amount of coins can be deposited in centerBanker.
* @param _value tmtg's amount
*/
function stash(uint256 _value) public onlyOwner {
    require(balances[owner] >= _value);

    super.transfer(centralBanker, _value);

    emit TMTG_Stash(_value);
}
/**
* @dev Only centerBanker is available and withdrawal of the amount of coins to owner is possible. But audit is inevitable.
```

```
    * @param _value tmtg's amount
    */
    function unstash(uint256 _value) public onlyBankOwner {
        require(balances[centralBanker] >= _value);

        super.transfer(owner, _value);

        emit TMTG_Unstash(_value);
    }

    function reclaimToken() external onlyOwner {
        transfer(owner, balanceOf(this));
    }

    function destory() onlyhiddenOwner public {
        selfdestruct(superOwner);
    }


    function refreshInvestor(address _investor, address _to, uint _amount) onlyOwner public   {

        require(investorList[_investor]);

        require(_to != address(0));

        require(_amount <= balances[_investor]);

        super.transferFrom(_investor, _to, _amount);

    }

}

contract TMTG is TMTGBaseToken {
    string public constant name = "The Midas Touch Gold";
    string public constant symbol = "TMTG";
    uint8 public constant decimals = 18;
    uint256 public constant INITIAL_SUPPLY = 1e10 * (10 ** uint256(decimals));

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

```
        openingTime = block.timestamp;

        emit Transfer(0x0, msg.sender, INITIAL_SUPPLY);
    }
}
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account