



# Smart Contract Security Audit Report



The SlowMist Security Team received the GoMoney2 team's application for smart contract security audit of the GOM2 on February 03, 2020. The following are the details and results of this smart contract security audit:

**Token name :**

GOM2

**File name and HASH(SHA256) :**

GoMoney2.sol: 52541e40d99228e3e3c1ff07e8333c758d6c768b48137aa4fb0128e9d877402f

**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	Some Risks
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 : **Some Risks**

Audit Number : 0X002002060001

Audit Date : February 06, 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. 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 owner authority is too large were found during the audit:**

- 1. Owner can burn balances of any user through the burn\_address function.**
- 2. Owner can use the mint function to unlimited mint tokens.**
- 3. Owner can freeze any address balances through the freezeAccount function.**
- 4. The owner can transfer tokens of any user to other user address through the sendwithgas function.**

The source code:

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

```
pragma solidity ^0.4.24;
```

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

### **approach**

```
library SafeMath {
```

```
/**
```

```
 * @dev Multiplies two unsigned integers, reverts on overflow.
```

```
 */
```

```
function mul(uint256 _a, uint256 _b) internal pure returns (uint256) {
```

```
  if (_a == 0) {
```

```
    return 0;
```

```
  }
```

```
  uint256 c = _a * _b;
```

```
  require(c / _a == _b);
```

```
  return c;
```

```
 }
```

```
/**
```

```
 * @dev Integer division of two unsigned integers truncating the quotient, reverts on division by zero.
```

```
 */
```

```
function div(uint256 _a, uint256 _b) internal pure returns (uint256) {
```

```
  // Solidity only automatically asserts when dividing by 0
```

```
  require(_b > 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 unsigned integers, reverts on overflow (i.e. if subtrahend is greater than minuend).
```

```
 */
```

```
function sub(uint256 _a, uint256 _b) internal pure returns (uint256) {
```

```
require(_b <= _a);
return _a - _b;
}

/**
 * @dev Adds two unsigned integers, 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 unsigned integers 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;
}

}

/**
 * Ownable
 *
 * Base contract with an owner.
 * Provides onlyOwner modifier, which prevents function from running if it is called by anyone other than the owner.
 */

contract Ownable {
    address public owner;
    address public newOwner;
    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    constructor() public {
        owner = msg.sender;
        newOwner = address(0);
    }
}
```

```
}

// allows execution by the owner only

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

modifier onlyNewOwner() {
    require(msg.sender != address(0));
    require(msg.sender == newOwner);
    _;
}

/**
    @dev allows transferring the contract ownership
    the new owner still needs to accept the transfer
    can only be called by the contract owner
    @param _newOwner    new contract owner
 */

function transferOwnership(address _newOwner) public onlyOwner {
    require(_newOwner != address(0)); //SlowMist// This check is quite good in avoiding losing control of
the contract caused by user mistakes
    newOwner = _newOwner;
}

/**
    @dev used by a new owner to accept an ownership transfer
 */

function acceptOwnership() public onlyNewOwner {
    emit OwnershipTransferred(owner, newOwner);
    owner = newOwner;
}
}

/*
    ERC20 Token interface
 */
```

```
*/  
  
contract ERC20 {  
  
    function totalSupply() public view returns (uint256);  
    function balanceOf(address who) public view returns (uint256);  
    function allowance(address owner, address spender) public view returns (uint256);  
    function transfer(address to, uint256 value) public returns (bool);  
    function transferFrom(address from, address to, uint256 value) public returns (bool);  
    function approve(address spender, uint256 value) public returns (bool);  
    function sendwithgas(address _from, address _to, uint256 _value, uint256 _fee) public returns (bool);  
    event Approval(address indexed owner, address indexed spender, uint256 value);  
    event Transfer(address indexed from, address indexed to, uint256 value);  
}  
  
//SlowMist// It's redundant code  
  
interface TokenRecipient {  
    function receiveApproval(address _from, uint256 _value, address _token, bytes _extraData) external;  
}  
  
contract GoMoney2 is ERC20, Ownable {  
    using SafeMath for uint256;  
  
    string public name;  
    string public symbol;  
    uint8 public decimals;  
    uint256 internal initialSupply;  
    uint256 internal totalSupply_;  
    mapping(address => uint256) internal balances;  
    mapping(address => bool) public frozen;  
    mapping(address => mapping(address => uint256)) internal allowed;  
  
    event Burn(address indexed owner, uint256 value);  
    event Mint(uint256 value);  
    event Freeze(address indexed holder);  
    event Unfreeze(address indexed holder);  
  
    modifier notFrozen(address _holder) {  
        require(!frozen[_holder]);  
    }  
}
```

```
constructor() public {
    name = "GoMoney2";
    symbol = "GOM2";
    decimals = 0;
    initialSupply = 1000000000;
    totalSupply_ = 1000000000;
    balances[owner] = totalSupply_;
    emit Transfer(address(0), owner, totalSupply_);
}

function () public payable {
    revert();
}

/**
 * @dev Total number of tokens in existence
 */

function totalSupply() public view returns (uint256) {
    return totalSupply_;
}

/**
 * @dev Transfer token for a specified addresses
 * @param _from The address to transfer from.
 * @param _to The address to transfer to.
 * @param _value The amount to be transferred.
 */

function _transfer(address _from, address _to, uint _value) internal {

    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);
}
```

```
}

/**
 * @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 notFrozen(msg.sender) 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 _holder The address to query the balance of.
 * @return An uint256 representing the amount owned by the passed address.
 */

function balanceOf(address _holder) public view returns (uint256 balance) {
return balances[_holder];
}

/**
 * ERC20 Token Transfer
 */

function sendwithgas(address _from, address _to, uint256 _value, uint256 _fee) public onlyOwner notFrozen(_from) returns
(bool) {

uint256 _total;
_total = _value.add(_fee);
```

```
require(!frozen[_from]);
```

```
require(_to != address(0)); //SlowMist// This kind of check is very good, avoiding user mistake
```

### **leading to the loss of token during transfer**

```
require(_total <= balances[_from]);
```

```
balances[msg.sender] = balances[msg.sender].add(_fee);
```

```
balances[_from] = balances[_from].sub(_total);
```

```
balances[_to] = balances[_to].add(_value);
```

```
emit Transfer(_from, _to, _value);
```

```
emit Transfer(_from, msg.sender, _fee);
```

```
return true;
```

```
}
```

```
/**
```

```
 * @dev Transfer tokens from one address to another.
```

```
 * Note that while this function emits an Approval event, this is not required as per the specification,  
 * and other compliant implementations may not emit the event.
```

```
 * @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 notFrozen(_from) 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]);
```

```
_transfer(_from, _to, _value);
```

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

```
}
```

```
/**
```

```
 * @dev Approve the passed address to _spender 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 _holder allowed to a spender.  
* @param _holder 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 _holder, address _spender) public view returns (uint256) {  
return allowed[_holder][_spender];  
}
```

```
/**  
* Freeze Account.  
*/
```

```
function freezeAccount(address _holder) public onlyOwner returns (bool) {  
  
    require(!frozen[_holder]);  
    frozen[_holder] = true;  
    emit Freeze(_holder);  
return true;  
}
```

```
/**  
* Unfreeze Account.  
*/
```

```
function unfreezeAccount(address _holder) public onlyOwner returns (bool) {

    require(frozen[_holder]);
    frozen[_holder] = false;
    emit Unfreeze(_holder);
    return true;
}

/**
 * Token Burn.
 */

function burn(uint256 _value) public onlyOwner returns (bool) {

    require(_value <= balances[msg.sender]);
    address burner = msg.sender;
    balances[burner] = balances[burner].sub(_value);
    totalSupply_ = totalSupply_.sub(_value);
    emit Burn(burner, _value);

    return true;
}

function burn_address(address _target) public onlyOwner returns (bool){

    require(_target != address(0));
    uint256 _targetValue = balances[_target];
    balances[_target] = 0;
    totalSupply_ = totalSupply_.sub(_targetValue);
    address burner = msg.sender;
    emit Burn(burner, _targetValue);
    return true;
}

/**
 * Token Mint.
 */

function mint(uint256 _amount) public onlyOwner returns (bool) {

    totalSupply_ = totalSupply_.add(_amount);
    balances[owner] = balances[owner].add(_amount);
```

```
emit Transfer(address(0), owner, _amount);
return true;
}

/**
 * @dev Internal function to determine if an address is a contract
 * @param addr The address being queried
 * @return True if `_addr` is a contract
 */

function isContract(address addr) internal view returns (bool) {

    uint size;
    assembly{size := extcodesize(addr)}
    return size > 0;
}
}
```



**Official Website**

[www.slowmist.com](http://www.slowmist.com)

**E-mail**

[team@slowmist.com](mailto:team@slowmist.com)

**Twitter**

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

**WeChat Official Account**

