Smart Contract Security Audit Report

The SlowMist Security Team received the MileVerse team's application for smart contract security audit of the MVC on Aug. 19, 2020. The following are the details and results of this smart contract security audit:

Token name :

MVC

The Contract address :

0x581911b360B6eB3a14eF295a83a91DC2bCE2D6f7

Link address :

https://etherscan.io/address/0x581911b360B6eB3a14eF295a83a91DC2bCE2D6f7

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 : 0X002008240001

Audit Date : Aug. 24, 2020

Audit Team : SlowMist Security Team

Summary: This is a token contract that contains the tokenVault section. The total amount of tokens in the contract remains unchanged. OpenZeppelin's 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 the following issues:

1. The owner can freeze any user's account through the freeze function.

2. In the transfer function, the frozen state of [to] was not checked.

3. In the transferFrom function, the frozen state of [msg.sender] and [to] was not checked.

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2020-06-24
*/
```

```
// File: contracts\library\SafeMath.sol

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

pragma solidity 0.6.10;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */

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

library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
```

```
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     *
     * _Available since v2.4.0._
     */
    function sub(uint256 a, uint256 b, string memory errorMessage)
        internal
        pure
        returns (uint256)
    {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
```

```
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function div(uint256 a, uint256 b, string memory errorMessage)
        internal
        pure
```

```
        returns (uint256)
    {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function mod(uint256 a, uint256 b, string memory errorMessage)
        internal
        pure
        returns (uint256)
```

```
    {
        require(b != 0, errorMessage);
        return a % b;
    }
}

// File: contracts\erc20\ERC20.sol

pragma solidity 0.6.10;


abstract contract ERC20 {
    using SafeMath for uint256;

    uint256 private _totalSupply;
    mapping(address => uint256) internal _balances;
    mapping(address => mapping(address => uint256)) internal _allowances;

    event Transfer(address indexed from, address indexed to, uint256 amount);
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 amount
    );

    /*
    * Internal functions for ERC20 standard logics
    */

    function _transfer(address from, address to, uint256 amount)
        internal
        returns (bool success)
    {
        _balances[from] = _balances[from].sub(
            amount,
            "ERC20/transfer : cannot transfer more than token owner balance"
        );
        _balances[to] = _balances[to].add(amount);
        emit Transfer(from, to, amount);
        success = true;
    }
```

```
function _approve(address owner, address spender, uint256 amount)
    internal
    returns (bool success)
{
    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
    success = true;
}

function _mint(address recipient, uint256 amount)
    internal
    returns (bool success)
{
    _totalSupply = _totalSupply.add(amount);
    _balances[recipient] = _balances[recipient].add(amount);
    emit Transfer(address(0), recipient, amount);
    success = true;
}
```

//SlowMist// It's redundant code

```
function _burn(address burned, uint256 amount)
    internal
    returns (bool success)
{
    _balances[burned] = _balances[burned].sub(
        amount,
        "ERC20Burnable/burn : Cannot burn more than user's balance"
    );
    _totalSupply = _totalSupply.sub(
        amount,
        "ERC20Burnable/burn : Cannot burn more than totalSupply"
    );
    emit Transfer(burned, address(0), amount);
    success = true;
}

/*
* public view functions to view common data
*/

function totalSupply() external view returns (uint256 total) {
    total = _totalSupply;
```

```
    }
    function balanceOf(address owner) external view returns (uint256 balance) {
        balance = _balances[owner];
    }

    function allowance(address owner, address spender)
        external
        view
        returns (uint256 remaining)
    {
        remaining = _allowances[owner][spender];
    }

    /*
    * External view function interface to implement on final contract
    */
    function name() virtual external view returns (string memory tokenName);
    function symbol() virtual external view returns (string memory tokenSymbol);
    function decimals() virtual external view returns (uint8 tokenDecimals);

    /*
    * External function interface to implement on final contract
    */
    function transfer(address to, uint256 amount)
        virtual
        external
        returns (bool success);
    function transferFrom(address from, address to, uint256 amount)
        virtual
        external
        returns (bool success);
    function approve(address spender, uint256 amount)
        virtual
        external
        returns (bool success);
}

// File: contracts\library\Ownable.sol

pragma solidity 0.6.10;

contract Ownable {
```

```
    address internal _owner;

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

    constructor() internal {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), msg.sender);
    }

    modifier onlyOwner() {
        require(
            msg.sender == _owner,
            "Ownable : Function called by unauthorized user."
        );
        _;
    }

    function owner() external view returns (address ownerAddress) {
        ownerAddress = _owner;
    }

    function transferOwnership(address newOwner)
        public
        onlyOwner
        returns (bool success)
    {
        require(newOwner != address(0), "Ownable/transferOwnership : cannot transfer ownership to zero address");
//SlowMist// This check is quite good in avoiding losing control of the contract caused by user
mistakes
        success = _transferOwnership(newOwner);
    }

    function renounceOwnership() external onlyOwner returns (bool success) {
        success = _transferOwnership(address(0));
    }

    function _transferOwnership(address newOwner) internal returns (bool success) {
```

```
            emit OwnershipTransferred(_owner, newOwner);
            _owner = newOwner;
            success = true;
        }
    }

    // File: contracts\erc20\ERC20Lockable.sol

    pragma solidity 0.6.10;



    abstract contract ERC20Lockable is ERC20, Ownable {
        struct LockInfo {
            uint256 amount;
            uint256 due;
        }

        mapping(address => LockInfo[]) internal _locks;
        mapping(address => uint256) internal _totalLocked;

        event Lock(address indexed from, uint256 amount, uint256 due);
        event Unlock(address indexed from, uint256 amount);

        modifier checkLock(address from, uint256 amount) {
            require(_balances[from] >= _totalLocked[from].add(amount), "ERC20Lockable/Cannot send more than unlocked
    amount");
            _;
        }

        function _lock(address from, uint256 amount, uint256 due)
            internal
            returns (bool success)
        {
            require(due > now, "ERC20Lockable/lock : Cannot set due to past");
            require(
                _balances[from] >= amount.add(_totalLocked[from]),
                "ERC20Lockable/lock : locked total should be smaller than balance"
            );
            _totalLocked[from] = _totalLocked[from].add(amount);
            _locks[from].push(LockInfo(amount, due));
            emit Lock(from, amount, due);
```

```
        success = true;
    }

    function _unlock(address from, uint256 index) internal returns (bool success) {
        LockInfo storage lock = _locks[from][index];
        _totalLocked[from] = _totalLocked[from].sub(lock.amount);
        emit Unlock(from, lock.amount);
        _locks[from][index] = _locks[from][_locks[from].length - 1];
        _locks[from].pop();
        success = true;
    }

    function unlock(address from) external returns (bool success) {
        for(uint256 i = 0; i < _locks[from].length; i++){
            if(_locks[from][i].due < now){
                _unlock(from, i);
            }
        }
        success = true;
    }

    function releaseLock(address from)
        external
        onlyOwner
        returns (bool success)
    {
        for(uint256 i = 0; i < _locks[from].length; i++){
            _unlock(from, i);
        }
        success = true;
    }
```

//SlowMist// The owner can lock the token it sends via the transferWithLockUp function

```
    function transferWithLockUp(address recipient, uint256 amount, uint256 due)
        external
        onlyOwner
        returns (bool success)
    {
        require(
            recipient != address(0),
            "ERC20Lockable/transferWithLockUp : Cannot send to zero address"
        );
```

```
        _transfer(msg.sender, recipient, amount);
        _lock(recipient, amount, due);
        success = true;
    }

    function lockInfo(address locked, uint256 index)
        external
        view
        returns (uint256 amount, uint256 due)
    {
        LockInfo memory lock = _locks[locked][index];
        amount = lock.amount;
        due = lock.due;
    }

    function totalLocked(address locked) external view returns(uint256 amount, uint256 length){
        amount = _totalLocked[locked];
        length = _locks[locked].length;
    }
}

// File: contracts\library\Pausable.sol

pragma solidity 0.6.10;


contract Pausable is Ownable {
    bool internal _paused;

    event Paused();
    event Unpaused();

    modifier whenPaused() {
        require(_paused, "Paused : This function can only be called when paused");
        _;
    }

    modifier whenNotPaused() {
        require(!_paused, "Paused : This function can only be called when not paused");
        _;
    }
```

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

```
    function pause() external onlyOwner whenNotPaused returns (bool success) {
        _paused = true;
        emit Paused();
        success = true;
    }

    function unPause() external onlyOwner whenPaused returns (bool success) {
        _paused = false;
        emit Unpaused();
        success = true;
    }

    function paused() external view returns (bool) {
        return _paused;
    }
}

// File: contracts\library\Freezable.sol

pragma solidity 0.6.10;



contract Freezable is Ownable {
    mapping(address => bool) private _frozen;

    event Freeze(address indexed target);
    event Unfreeze(address indexed target);

    modifier whenNotFrozen(address target) {
        require(!_frozen[target], "Freezable : target is frozen");
        _;
    }

    //SlowMist// The owner can freeze any user's account through the freeze function

    function freeze(address target) external onlyOwner returns (bool success) {
        _frozen[target] = true;
        emit Freeze(target);
        success = true;
    }

    function unFreeze(address target)
        external
```

```
        onlyOwner
        returns (bool success)
    {
        _frozen[target] = false;
        emit Unfreeze(target);
        success = true;
    }

    function isFrozen(address target)
        external
        view
        returns (bool frozen)
    {
        return _frozen[target];
    }
}

// File: contracts\MileVerseToken.sol

pragma solidity 0.6.10;




contract MileVerseToken is
    ERC20Lockable,
    Pausable,
    Freezable
{
    string constant private _name = "MileVerse";
    string constant private _symbol = "MVC";
    uint8 constant private _decimals = 18;
    uint256 constant private _initial_supply = 3_000_000_000;

    constructor() public Ownable() {
        _mint(msg.sender, _initial_supply * (10**uint256(_decimals)));
    }

    function transfer(address to, uint256 amount)
        override
        external

        whenNotFrozen(msg.sender) //SlowMist// The frozen state of [to] was not checked
```

```
        whenNotPaused
        checkLock(msg.sender, amount)
        returns (bool success)
    {
        require(
            to != address(0), //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer
            "SAM/transfer : Should not send to zero address"
        );
        _transfer(msg.sender, to, amount);
        success = true; //SlowMist// The return value conforms to the EIP20 specification
    }

    function transferFrom(address from, address to, uint256 amount)
        override
        external
        whenNotFrozen(from) //SlowMist// The frozen state of [msg.sender] and [to] was not checked
        whenNotPaused
        checkLock(from, amount)
        returns (bool success)
    {
        require(
            to != address(0), //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer
            "SAM/transferFrom : Should not send to zero address"
        );
        _transfer(from, to, amount);
        _approve(
            from,
            msg.sender,
            _allowances[from][msg.sender].sub(
                amount,
                "SAM/transferFrom : Cannot send more than allowance"
            )
        );
        success = true; //SlowMist// The return value conforms to the EIP20 specification
```

```
    }

    function approve(address spender, uint256 amount)
        override
        external
        returns (bool success)
    {
        require(
            spender != address(0), //SlowMist// This kind of check is very good, avoiding user mistake
leading to approve errors
            "SAM/approve : Should not approve zero address"
        );
        _approve(msg.sender, spender, amount);
        success = true; //SlowMist// The return value conforms to the EIP20 specification
    }

    function name() override external view returns (string memory tokenName) {
        tokenName = _name;
    }

    function symbol() override external view returns (string memory tokenSymbol) {
        tokenSymbol = _symbol;
    }

    function decimals() override external view returns (uint8 tokenDecimals) {
        tokenDecimals = _decimals;
    }
}
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account