# Smart Contract Security Audit Report

The SlowMist Security Team received the LEMD team's application for smart contract security audit of the LEMD token on Apr. 13, 2021. The following are the details and results of this smart contract security audit:

**Token name :**

LEMD

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

LEMD.sol: 25b7748533ac8dc5277bc958e912653298c361d6e25c411298fea520f86e2e04

**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 authority audit | 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 top-up” 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 : 0X002104150003

Audit Date : Apr. 15, 2021

Audit Team : SlowMist Security Team



**Summary: This is a token contract that does not contain the tokenVault section. The total amount of tokens in the contract remains unchanged. OpenZeppelin's SafeMath security module is used, which is a recommend approach. The contract does not have the Overflow and the Race Conditions issue.**

**During the audit, we found the following information:**

**The minter can mint tokens at will, but the number of minted tokens has an upper limit called cap.**

The source code:

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

pragma solidity 0.5.8;

interface IERC20 {
    function totalSupply() external view returns (uint);
    function balanceOf(address account) external view returns (uint);
    function transfer(address recipient, uint amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint);
```

```
    function approve(address spender, uint amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint value);
    event Approval(address indexed owner, address indexed spender, uint value);
}

contract Context {
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

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

contract ERC20 is Context, IERC20 {
    using SafeMath for uint;

    mapping (address => uint) private _balances;

    mapping (address => mapping (address => uint)) private _allowances;

    uint private _totalSupply;
    function totalSupply() public view returns (uint) {
        return _totalSupply;
    }
    function balanceOf(address account) external view returns (uint) {
        return _balances[account];
    }
    function transfer(address recipient, uint amount) external returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;//SlowMist// The return value conforms to the EIP20 specification
    }
    function allowance(address owner, address spender) external view returns (uint) {
        return _allowances[owner][spender];
    }
    function approve(address spender, uint amount) external returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;//SlowMist// The return value conforms to the EIP20 specification
    }
```

```
    function transferFrom(address sender, address recipient, uint amount) external returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));
        return true; //SlowMist// The return value conforms to the EIP20 specification
    }
    function increaseAllowance(address spender, uint addedValue) external returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }
    function decreaseAllowance(address spender, uint subtractedValue) external returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero"));
        return true;
    }
    function _transfer(address sender, address recipient, uint amount) internal {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");//SlowMist// The kind of check is very good,avoiding user mistake leading to the loss of token during transfer

        _beforeTokenTransfer(sender, recipient, amount);

        _balasnces[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }
    function _mint(address account, uint amount) internal {
        require(account != address(0), "ERC20: mint to the zero address");

        _beforeTokenTransfer(address(0), account, amount);

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }
    function _burn(address account, uint amount) internal {
        require(account != address(0), "ERC20: burn from the zero address");

        _beforeTokenTransfer(account, address(0), amount);
```

```
        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }
    function _approve(address owner, address spender, uint amount) internal {
        require(owner != address(0), "ERC20: approve from the zero address");

        require(spender != address(0), "ERC20: approve to the zero address");//SlowMist// The kind of check is very
```

**good,avoiding user mistake leading to approve errors**

```
        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

     /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be to transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal { }
}

contract ERC20Detailed is IERC20 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
```

```
    }
    function name() external view returns (string memory) {
        return _name;
    }
    function symbol() external view returns (string memory) {
        return _symbol;
    }
    function decimals() external view returns (uint8) {
        return _decimals;
    }

}
```

**//SlowMist// OpenZeppelin’s SafeMath security module is used，which is a recommend approach**

```
library SafeMath {
    function add(uint a, uint b) internal pure returns (uint) {
        uint c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }
    function sub(uint a, uint b) internal pure returns (uint) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }
    function sub(uint a, uint b, string memory errorMessage) internal pure returns (uint) {
        require(b <= a, errorMessage);
        uint c = a - b;

        return c;
    }
    function mul(uint a, uint b) internal pure returns (uint) {
        if (a == 0) {
            return 0;
        }

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

        return c;
    }
    function div(uint a, uint b) internal pure returns (uint) {
        return div(a, b, "SafeMath: division by zero");
```

```
    }
    function div(uint a, uint b, string memory errorMessage) internal pure returns (uint) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint c = a / b;

        return c;
    }
}

library Address {
    function isContract(address account) internal view returns (bool) {
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }
}

library SafeERC20 {
    using SafeMath for uint;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint value) internal {
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
```

```
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

/**
 * Token
 */
```

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

```
pragma solidity 0.5.8;

contract LEMD is ERC20, ERC20Detailed {
    using SafeERC20 for IERC20;
    using Address for address;
    using SafeMath for uint;

    uint256 private _cap;
    address public governance;
    address public pendingGov;

    mapping (address => bool) public minters;

    event NewPendingGov(address oldPendingGov, address newPendingGov);

    event NewGov(address oldGov, address newGov);

    event AddMinter(address minter);

    event RemoveMinter(address minter);

    event Mint(address minter, uint256 amount);

    // Modifiers
    modifier onlyGov() {
        require(msg.sender == governance, "You are not the governance");
        _;
```

```
    }

    constructor (uint256 cap_) public ERC20Detailed("lemond.money", "LEMD", 18) {
        require(cap_ > 0, "Cap is 0");
        _cap = cap_;
        governance = tx.origin;
    }

    /**
     * @dev Returns the cap on the token's total supply.
     */
    function cap() public view returns (uint256) {
        return _cap;
    }
    /**
     * Minte Token for Account
     * @param _account minter
     * @param _amount amount
     */
```

//SlowMist// The minter can mint tokens at will, but the number of minted tokens has an upper limit called cap

```
    function mint(address _account, uint256 _amount) external {
        require(minters[msg.sender], "You are not the minter");
        require(ERC20.totalSupply() + _amount <= cap(), "Cap exceeded");
        _mint(_account, _amount);

        emit Mint(_account, _amount);
    }
    /**
     * Add minter
     * @param _minter minter
     */
    function addMinter(address _minter) external onlyGov {
        minters[_minter] = true;

        emit AddMinter(_minter);
    }

    /**
     * Remove minter
```

```
     * @param _minter minter
     */
    function removeMinter(address _minter) external onlyGov {
        minters[_minter] = false;

        emit RemoveMinter(_minter);
    }

    /**
     * Set new governance
     * @param _pendingGov the new governance
     */
    function setPendingGov(address _pendingGov)
        external
        onlyGov
    {
        address oldPendingGov = pendingGov;
        pendingGov = _pendingGov;
        emit NewPendingGov(oldPendingGov, _pendingGov);
    }

    /**
     * lets msg.sender accept governance
     */
    function acceptGov()
        external {
        require(msg.sender == pendingGov, "You are not the pending governance");
        address oldGov = governance;
        governance = pendingGov;
        pendingGov = address(0);
        emit NewGov(oldGov, governance);
    }
    /**
     * @dev See {ERC20-_beforeTokenTransfer}.
     *
     * Requirements:
     *
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal {
        super._beforeTokenTransfer(from, to, amount);
    }
}
```

**Official Website**
www.slowmist.com

**E-mail**
team@slowmist.com

**Twitter**
@SlowMist_Team

**Github**
https://github.com/slowmist