freshcoins

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

LULU Token

$LULU

22/02/2022

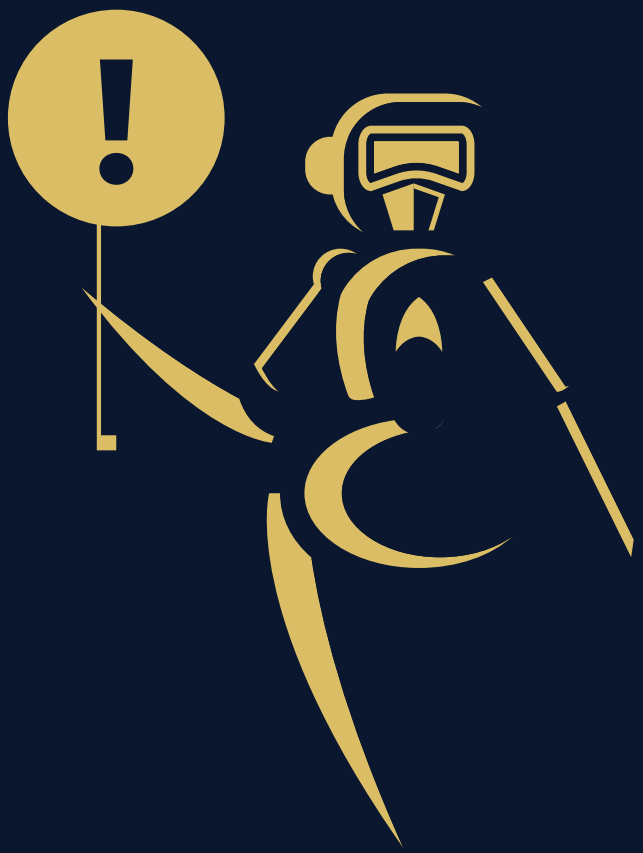
TABLE OF CONTENTS

DISCLAIMER

The information provided on this analysis document is only for general information and should not be used as a reason to invest.

FreshCoins Team will take no payment for manipulating the results of this audit.

The score and the result will stay on this project page information on our website https://freshcoins.io
FreshCoins Team does not guarantees that a project will not sell off team supply, or any other scam strategy (RUG or Honeypot etc)



INTRODUCTION

FreshCoins (Consultant) was contracted by
LULU Token (Customer) to conduct a Smart Contract Code Review
and Security Analysis.

0xF58E91f0C96e9a0F5F984Fc6CF476edD311a04BA

Network: Binance Smart Chain (BSC)

This report presents the findings of the security assessment of
Customer's smart contract and its code review conducted on 22/02/2022

AUDIT OVERVIEW

95

Security Score

94

Static Scan
Automatic scanning for common vulnerabilities

98

ERC Scan
Automatic checks for ERC's conformance

0 High

0 Medium

1 Low

0 Optimizations

0 Informational

No.	Issue description	Checking Status
1	Compiler Errors / Warnings	Passed
2	Reentrancy and Cross-function	Passed
3	Front running	Passed
4	Timestamp dependence	Passed
5	Integer Overflow and Underflow	Passed
6	Reverted DoS	Passed
7	DoS with block gas limit	Low
8	Methods execution permissions	Passed
9	Exchange rate impact	Passed
10	Malicious Event	Passed
11	Scoping and Declarations	Passed
12	Uninitialized storage pointers	Passed
13	Design Logic	Passed
14	Safe Zeppelin module	Passed

OWNER PRIVILEGES

Contract owner can't mint tokens after initial contract deploy

Contract owner can exclude/include wallet from fees

```
function includeOrExcludeFromFee(address _account, bool _value)
    public
    onlyOwner
  {
    _isExcludedFromFee[_account] = _value;
}
```

Contract owner can exclude/include wallet from rewards

```
function includeInReward(address _account) external onlyOwner {
    require(
      _isExcludedFromReward[_account],
      "BEP20: _Account is already excluded"
    );
    excludedTSupply = excludedTSupply.sub(_tOwned[_account]);
    excludedRSupply = excludedRSupply.sub(_rOwned[_account]);
    _rOwned[_account] = _tOwned[_account].mul(_getRate());
    _tOwned[_account] = 0;
    _isExcludedFromReward[_account] = false;
}

function excludeFromReward(address _account) public onlyOwner {
    require(
      !_isExcludedFromReward[_account],
      "BEP20: _Account is already excluded"
    );
    if (_rOwned[_account] > 0) {
      _tOwned[_account] = tokenFromReflection(_rOwned[_account]);
    }
    _isExcludedFromReward[_account] = true;
    excludedTSupply = excludedTSupply.add(_tOwned[_account]);
    excludedRSupply = excludedRSupply.add(_rOwned[_account]);
}
```

Contract owner can exclude/include wallet from tx limitations

```
function includeOrExcludeFromMaxHoldLimit(address _address, bool value)
    public
    onlyOwner
  {
    _isExcludedFromMaxHoldLimit[_address] = value;
}
```

Contract owner can exclude/include wallet from sell limitations

```
function setExcludeFromMaxSellLimit(address _address, bool value)
      public
      onlyOwner
   {
      _isExcludedFromMaxSellLimit[_address] = value;
}
```

Contract owner can exclude/include an address from transactions

```
function addInBlackList(address account) external onlyOwner {
      require(
         account != address(dexRouter),
         "We can not blacklist pancakeRouter"
      );
      require(!_isBlackListed[account], "Account is already blacklisted");
      _isBlackListed[account] = true;
      _confirmedBlackLister.push(account);

      emit SniperBotAddded(account);
}

function removeFromBlackList(address account) external onlyOwner {
      require(_isBlackListed[account], "Account is not blacklisted");
      for (uint256 i = 0; i < _confirmedBlackLister.length; i++) {
         if (_confirmedBlackLister[i] == account) {
            _confirmedBlackLister[i] = _confirmedBlackLister[
               _confirmedBlackLister.length - 1
            ];
            _isBlackListed[account] = false;
            _confirmedBlackLister.pop();
            break;
         }
      }

      emit SniperBotRemoved(account);
}
```

Contract owner can change max wallet tx

```
function setMaxHoldingAmount(uint256 _amount) public onlyOwner {
      maxHoldingAmount = _amount;
}
```

Contract owner can enable/disable fees

```
function enableOrDisableFees(bool _state) external onlyOwner {
      reflectionFees = _state;
}
```

Contract owner can enable/disable wallet limitations

```
function enableOrDisableMaxHoldLimit(bool _state) external onlyOwner {
    isMaxHoldLimitValid = _state;
}
```

Contract owner can change the fees up to 100%

```
function setBuyFeePercent(
    uint256 _redistributionFee,
    uint256 _liquidityFee,
    uint256 _farmingPoolFee,
    uint256 _marketWalletFee
  ) external onlyOwner {
    redistributionFeeOnBuying = _redistributionFee;
    liquidityFeeOnBuying = _liquidityFee;
    farmingPoolFeeOnBuying = _farmingPoolFee;
    marketWalletFeeOnBuying = _marketWalletFee;
}

function setSellFeePercent(
    uint256 _redistributionFee,
    uint256 _liquidityFee,
    uint256 _farmingPoolFee,
    uint256 _marketWalletFee
  ) external onlyOwner {
    redistributionFeeOnSelling = _redistributionFee;
    liquidityFeeOnSelling = _liquidityFee;
    farmingPoolFeeOnSelling = _farmingPoolFee;
    marketWalletFeeOnSelling = _marketWalletFee;
}

function setNormalFeePercent(
    uint256 _redistributionFee,
    uint256 _liquidityFee,
    uint256 _farmingPoolFee,
    uint256 _marketWalletFee
  ) external onlyOwner {
    redistributionFee = _redistributionFee;
    liquidityFee = _liquidityFee;
    farmingPoolFee = _farmingPoolFee;
    marketWalletFee = _marketWalletFee;
}
```

Contract owner can change swap settings

```
function enableOrDisableSwapAndLiquify(bool _state) public onlyOwner {
    swapAndLiquifyEnabled = _state;
    emit SwapAndLiquifyEnabledUpdated(_state);
}
```

Contract owner can renounce ownership

```
function renounceOwnership() public virtual onlyOwner {
    _setOwner(address(0));
}
```

Contract owner can transfer ownership

```
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(
      newOwner != address(0),
      "Ownable: new owner is the zero address"
    );
    _setOwner(newOwner);
}
```

Contract owner can change farmingPool and marketWallet addresses

Current values:

farmingPool : 0x33a0aab5d27197ad3a9cf3dcc1630f21268c1e6a

marketWallet : 0x33a0aab5d27197ad3a9cf3dcc1630f21268c1e6a

```
function setfarmingPoolAddress(address payable _newAddress)
      external
      onlyOwner
  {
      farmingPool = _newAddress;
}

function setmarketWalletAddress(address payable _newAddress)
      external
      onlyOwner
  {
      marketWallet = _newAddress;
}
```

CONCLUSION AND ANALYSIS

Smart Contracts within the scope were manually reviewed and analyzed with static tools.

Audit report overview contains all found security vulnerabilities and other issues in the reviewed code.

Found 1 LOW issue during the first review.

TOKEN DETAILS

Details

Buy fees:	5%
Sell fees:	5%
Max TX:	N/A
Max Sell:	20,000 per transaction (Whale Protection)

Honeypot Risk

Ownership:	Owned
Blacklist:	Detected (Bot Protection)
Modify Max TX:	Not detected
Modify Max Sell:	Not detected
Disable Trading:	Not detected

Rug Pull Risk

Liquidity:	94,1% Locked with Unicrypt
Holders:	Clean

TECHNICAL DISCLAIMER

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. The audit can't guarantee the explicit security of the audited project / smart contract.