



BlocX, Smart Contract, Code Review and Security Analysis Report

Customer: BlocX
Website: <https://blocxchain.org>
Prepared on: 26th April, 2023
Platform: Binance Smart Chain
Language: Solidity

rdauditors.com

Table of Contents

Disclaimer	2
Documentation	3
Introduction	4
Project Scope	5
Executive Summary	6
Code Quality	6
Documentation	8
Use of Dependencies	9
AS-IS Overview	10
Code Flow Diagram - BlocX	13
Code Flow Diagram - Slither Results Log	14
Severity Definitions	19
Audit Findings	20
Conclusion	24
Note For Contract Users	24
Our Methodology	26
Disclaimers	28

Disclaimer

This document may contain confidential information about its systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the customer or it can be disclosed publicly after all vulnerabilities are fixed - upon the decision of the customer.

Documentation

Name	Smart Contract Code Review and Security Analysis Report of BlocX
Platform	BSC / Solidity
File	BlocX.sol
Online Code	0x2f5fb3d3ab21bde07c5aa389e92366651b406c30
MD5 hash	AE23609E4E66D46E6596FC1BFE8AD0E2
SHA256 hash	78D74EA62E23698465351B6B378AAD3AA13342EF00AC69B905CE8 239DAE7D656
Date	26/04/2023

Introduction

RD Auditors (Consultant) were contracted by BlocX (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report represents the findings of the security assessment of the customer`s smart contract and its code review conducted between 24th - 26th April, 2023.

This contract consists of one file.

Project Scope





The scope of the project is a smart contract. We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but is not limited to):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary






According to the assessment, the customer's solidity smart contract is now **Secure**.

You are Here

 **Insecure**  **Poorly Secured**  **Secure**  **Well-Secured**

Automated checks are with smartDec, Mythril, Slither and remix IDE. All issues were performed by our team, which included the analysis of code functionality, the manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the AS-IS section and all issues found are located in the audit overview section.

We found the following;

Total Issues	6
 Critical	0
 High	0
 Medium	0
 Low	0
 Very Low	6

Code Quality

The libraries within this smart contract are part of a logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned to a specific address and its properties/methods can be reused many times by other contracts.

The BlocX team has not provided scenario and unit test scripts, which helped to determine the integrity of the code in an automated way.

Documentation

We were given a BlocX smart contract code in the form of a BSCScan web link. The hash of that code is mentioned above in the table. As mentioned above, it's recommended to write comments in the smart contract code, so anyone can quickly understand the programming flow as well as complex code logic.

Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure. Those were based on well known industry standard open source projects and even core code blocks that are written well and systematically.

AS-IS Overview

BlocX.sol

File And Function Level Report

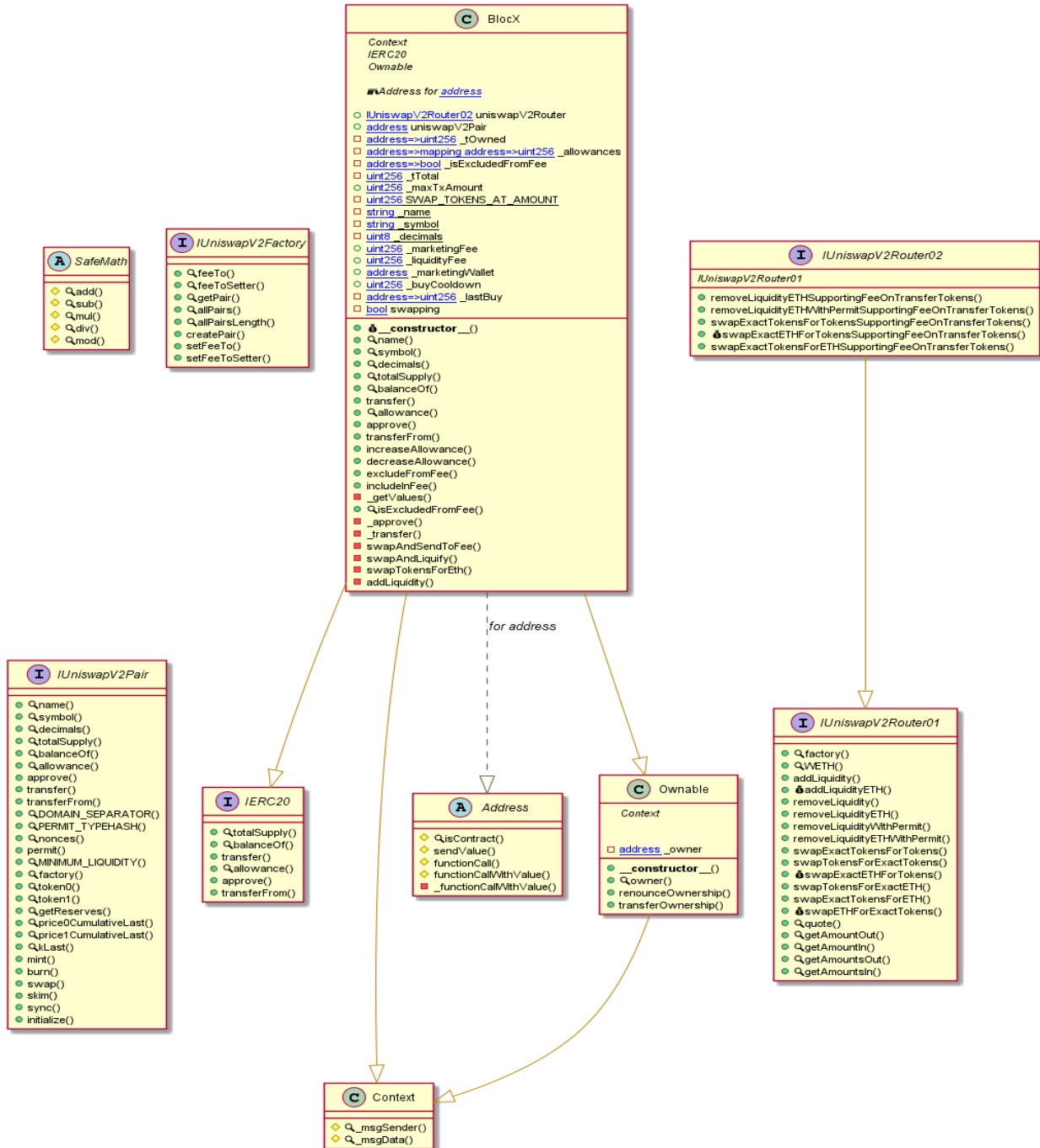
File : BlocX.sol
Contract: BlocX
Inherit: IERC20, Context, Ownable
Observation: Passed
Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	name	write	Passed	All Passed	No Issue	Passed
2	symbol	write	Passed	All Passed	No Issue	Passed
3	decimals	write	Passed	All Passed	No Issue	Passed
4	totalSupply	read	Passed	All Passed	No Issue	Passed
5	balanceOf	read	Passed	All Passed	No Issue	Passed
6	transfer	write	Passed	All Passed	No Issue	Passed
7	allowance	read	Passed	All Passed	No Issue	Passed
8	approve	write	Passed	All Passed	No Issue	Passed
9	transferFrom	write	Passed	All Passed	No Issue	Passed
10	increaseAllowance	write	Passed	All Passed	No Issue	Passed
11	decreaseAllowance	write	Passed	All Passed	No Issue	Passed

12	excludeFromFee	write	Access only Owner	All Passed	No Issue	Passed
13	includeInFee	write	Access only Owner	All Passed	No Issue	Passed
14	receive	external	Passed	All Passed	No Issue	Passed
15	_getValues	write	Passed	All Passed	No Issue	Passed
16	isExcludedFromFee	read	Passed	All Passed	No Issue	Passed
17	_approve	read	Passed	All Passed	No Issue	Passed
18	_transfer	write	A variable used in a function, but not updated in action	All Passed	Refer to audit findings	Passed
19	swapAndSendToFee	write	Passed	All Passed	No Issue	Passed
20	swapAndLiquify	write	Unused function swapAndLiquify, An addLiquidity function is called by swapAndLiquify function which is in turn unused	All Passed	Refer to audit findings	Passed
21	swapTokensForEth	write	Passed	All Passed	No Issue	Passed
22	addLiquidity	write	An addLiquidity function is called by swapAndLiquify function which is in turn unused	All Passed	Refer to audit findings	Passed
23	owner	read	Passed	All Passed	No Issue	Passed

24	onlyOwner	modifier	Passed	All Passed	No Issue	Passed
25	renounceOwnership	write	Access only Owner	All Passed	No Issue	Passed
26	transferOwnership	write	Access only Owner	All Passed	No Issue	Passed

Code Flow Diagram - BlocX



Code Flow Diagram - Slither Results Log

```

BlocX.allowance(address,address).owner (BlocX.sol#743) shadows:
  - Ownable.owner() (BlocX.sol#422-424) (function)
BlocX._approve(address,address,uint256).owner (BlocX.sol#794) shadows:
  - Ownable.owner() (BlocX.sol#422-424) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Reentrancy in BlocX.swapAndLiquify() (BlocX.sol#848-860):
  External calls:
    - newBalance = swapTokensForEth(half) (BlocX.sol#853)
      - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (BlocX.sol#872-878)
    - addLiquidity(otherHalf,newBalance) (BlocX.sol#857)
      - (ethFromLiquidity) = uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (BlocX.sol#887-894)
      - (success) = recipient.call{value: amount}() (BlocX.sol#307)
      - address(_marketingWallet).sendValue(ethAmount - ethFromLiquidity) (BlocX.sol#897)
  External calls sending eth:
    - addLiquidity(otherHalf,newBalance) (BlocX.sol#857)
      - (ethFromLiquidity) = uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (BlocX.sol#887-894)
      - (success) = recipient.call{value: amount}() (BlocX.sol#307)
  State variables written after the call(s):
    - addLiquidity(otherHalf,newBalance) (BlocX.sol#857)
      - _allowances[owner][spender] = amount (BlocX.sol#798)
Reentrancy in BlocX.transferFrom(address,address,uint256) (BlocX.sol#752-756):
  External calls:
    - _transfer(sender,recipient,amount) (BlocX.sol#753)
      - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (BlocX.sol#872-878)
  External calls sending eth:
    - _transfer(sender,recipient,amount) (BlocX.sol#753)
      - address(_marketingWallet).transfer(ethToSend) (BlocX.sol#845)
  State variables written after the call(s):
    - _approve(sender,msgSender(),_allowances[sender][_msgSender()] - amount) (BlocX.sol#754)
    - _allowances[owner][spender] = amount (BlocX.sol#798)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in BlocX.swapAndLiquify() (BlocX.sol#848-860):
  External calls:
    - newBalance = swapTokensForEth(half) (BlocX.sol#853)
      - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (BlocX.sol#872-878)
    - addLiquidity(otherHalf,newBalance) (BlocX.sol#857)
      - (ethFromLiquidity) = uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (BlocX.sol#887-894)
      - (success) = recipient.call{value: amount}() (BlocX.sol#307)
      - address(_marketingWallet).sendValue(ethAmount - ethFromLiquidity) (BlocX.sol#897)
  External calls sending eth:
    - addLiquidity(otherHalf,newBalance) (BlocX.sol#857)
      - (ethFromLiquidity) = uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (BlocX.sol#887-894)
      - (success) = recipient.call{value: amount}() (BlocX.sol#307)
  Event emitted after the call(s):
    - Approval(owner,spender,amount) (BlocX.sol#799)
    - addLiquidity(otherHalf,newBalance) (BlocX.sol#857)
    - SwapAndLiquify(half,newBalance,otherHalf) (BlocX.sol#858)

BlocX._transfer(address,address,uint256) (BlocX.sol#802-838) uses timestamp for comparisons
  Dangerous comparisons:
    - require(bool,string)(_lastBuy[to] + _buyCooldown < block.timestamp, Must wait til after cooldown to buy) (BlocX.sol#816)
BlocX.addLiquidity(uint256,uint256) (BlocX.sol#882-898) uses timestamp for comparisons
  Dangerous comparisons:
    - ethAmount - ethFromLiquidity > 0 (BlocX.sol#896)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Address.isContract(address) (BlocX.sol#276-285) uses assembly
  - INLINE ASM (BlocX.sol#283)
Address._functionCallWithValue(address,bytes,uint256,string) (BlocX.sol#369-390) uses assembly
  - INLINE ASM (BlocX.sol#382-385)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

```

```

Address.functionCallWithValue(address,bytes,uint256,string) (BlocX.sol#369-390) is never used and should be removed
Address.functionCall(address,bytes) (BlocX.sol#329-331) is never used and should be removed
Address.functionCall(address,bytes,string) (BlocX.sol#339-341) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (BlocX.sol#354-356) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (BlocX.sol#364-367) is never used and should be removed
Address.isContract(address) (BlocX.sol#276-285) is never used and should be removed
Address.sendValue(address,uint256) (BlocX.sol#303-309) is never used and should be removed
BlocX.addLiquidity(uint256,uint256) (BlocX.sol#882-898) is never used and should be removed
BlocX.swapAndLiquify() (BlocX.sol#848-860) is never used and should be removed
Context._msgData() (BlocX.sol#248-251) is never used and should be removed
SafeMath.add(uint256,uint256) (BlocX.sol#111-116) is never used and should be removed
SafeMath.div(uint256,uint256) (BlocX.sol#185-187) is never used and should be removed
SafeMath.div(uint256,uint256,string) (BlocX.sol#201-207) is never used and should be removed
SafeMath.mod(uint256,uint256) (BlocX.sol#221-223) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (BlocX.sol#237-240) is never used and should be removed
SafeMath.mul(uint256,uint256) (BlocX.sol#159-171) is never used and should be removed
SafeMath.sub(uint256,uint256) (BlocX.sol#128-130) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (BlocX.sol#142-147) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

```

```

Pragma version^0.8.6 (BlocX.sol#12) allows old versions
solc-0.8.6 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

```

```

Low level call in Address.sendValue(address,uint256) (BlocX.sol#303-309):
- (success) = recipient.call{value: amount}() (BlocX.sol#307)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (BlocX.sol#369-390):
- (success,returndata) = target.call{value: weiValue}(data) (BlocX.sol#373)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

```

```

Function IUniswapV2Pair.DOMAIN_SEPARATOR() (BlocX.sol#493) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (BlocX.sol#494) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (BlocX.sol#511) is not in mixedCase
Function IUniswapV2Router01.WETH() (BlocX.sol#533) is not in mixedCase
Variable BlocX._maxTxAmount (BlocX.sol#683) is not in mixedCase
Constant BlocX._name (BlocX.sol#685) is not in UPPER_CASE_WITH_UNDERSCORES
Constant BlocX._symbol (BlocX.sol#686) is not in UPPER_CASE_WITH_UNDERSCORES
Constant BlocX._decimals (BlocX.sol#687) is not in UPPER_CASE_WITH_UNDERSCORES
Variable BlocX._marketingFee (BlocX.sol#689) is not in mixedCase
Variable BlocX._liquidityFee (BlocX.sol#690) is not in mixedCase
Variable BlocX._marketingWallet (BlocX.sol#691) is not in mixedCase
Variable BlocX._buyCooldown (BlocX.sol#693) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

```

```

Redundant expression "this (BlocX.sol#249)" inContext (BlocX.sol#243-252)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

```

```

Reentrancy in BlocX.transferFrom(address,address,uint256) (BlocX.sol#752-756):
External calls:
- _transfer(sender,recipient,amount) (BlocX.sol#753)
- address(_marketingWallet).transfer(ethToSend) (BlocX.sol#845)
State variables written after the call(s):
- _approve(sender,_msgSender(),_allowances[sender][_msgSender()] - amount) (BlocX.sol#754)
- _allowances[owner][spender] = amount (BlocX.sol#798)
Event emitted after the call(s):
- Approval(owner,spender,amount) (BlocX.sol#799)
- approve(sender,_msgSender(),_allowances[sender][_msgSender()] - amount) (BlocX.sol#754)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4

```

```

Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (BlocX.sol#538) is too similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (BlocX.sol#539)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

```

```

BlocX._buyCooldown (BlocX.sol#693) should be constant
BlocX._liquidityFee (BlocX.sol#690) should be constant
BlocX._marketingFee (BlocX.sol#689) should be constant
BlocX._marketingWallet (BlocX.sol#691) should be constant
BlocX._maxTxAmount (BlocX.sol#683) should be constant
BlocX._tTotal (BlocX.sol#682) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

```

```

BlocX.uniswapV2Pair (BlocX.sol#675) should be immutable
BlocX.uniswapV2Router (BlocX.sol#674) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
BlocX.sol analyzed (10 contracts with 84 detectors), 58 result(s) found

```

Solidity Static Analysis

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in `BlocX.swapTokensForEth(uint256)`: Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 862:4:

Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

[more](#)

Pos: 382:16:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

[more](#)

Pos: 893:12:

Low level calls:

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.

[more](#)

Pos: 373:50:

Gas & Economy

Gas costs:

Gas requirement of function BlocX.decreaseAllowance is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 763:4:

ERC

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type

[more](#)

Pos: 484:4:

Miscellaneous

Constant/View/Pure functions:

IUniswapV2Router02.swapExactTokensForETHSupportingFeeOnTransferTokens(ui
: Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 660:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

[more](#)

Pos: 816:12:

Data truncated:

Division of integer values yields an integer value again. That means e.g. $10 / 100 = 0$ instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Pos: 851:23:

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lost tokens etc.
High	High level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial functions.
Medium	Medium level vulnerabilities are important to fix; however, they cannot lead to lost tokens.
Low	Low level vulnerabilities are most related to outdated, unused etc. These code snippets cannot have a significant impact on execution.
Lowest Code Style/ Best Practice	Lowest level vulnerabilities, code style violations and information statements cannot affect smart contract execution and can be ignored.

Audit Findings

Critical:

No critical severity vulnerabilities were found.

High:

No high severity vulnerabilities were found.

Medium:

No medium severity vulnerabilities were found.

Low:

No low severity vulnerabilities were found.

Very Low:

(1) SafeMath Library:

SafeMath Library is used in this contract code, but the compiler version is greater than or equal to 0.8.0, Then it will be not required to use, solidity automatically handles overflow/underflow.

Resolution: Remove the SafeMath library and use normal math operators, It will improve code size, and less gas consumption.

(2) Make variables constant:

```
uint256 public _marketingFee = 8;  
uint256 public _liquidityFee = 2;  
address public _marketingWallet = 0x0B67f86128A538639f575F9bee1cDB43675eF6ab;
```

```
uint256 private _tTotal = 1_000_000_000 * 10**9;  
uint256 public maxTxAmount = 20 000 000 * 10**9; //
```

These variables will be unchanged. So, please make it constant. It will save some gas.

Resolution: Declare those variables as constant. Just put a constant keyword.

(3) A variable used in a function, but not updated in action:

```
uint256 public _buyCooldown = 0 minutes;
```

```
function _transfer(  
    address from,  
    address to,  
    uint256 amount  
) private {  
    require(from != address(0), "ERC20: transfer from the zero address");  
    require(to != address(0), "ERC20: transfer to the zero address");  
    require(amount > 0, "Transfer amount must be greater than zero");  
  
    if (from != owner() && to != owner() && to != uniswapV2Pair)  
        require(balanceOf(to) + amount <= _maxTxAmount, "Transfer amount exceeds the maxTxAmount.");  
  
    if (from == uniswapV2Pair) {  
        require (_lastBuy[to] + _buyCooldown < block.timestamp, "Must wait til after cooldown to buy");  
        _lastBuy[to] = block.timestamp;  
    }  
}
```

Variable `_buyCooldown` used in a `_transfer()` function, but not updated anywhere, variable value is always 0 minutes.

Resolution: We suggest checking if this variable value is required or not, if this variable is not required then remove it from the code. Cross-verify before moving into production.

(4) Please use the latest compiler version when deploying contract:

This is not a severe issue, but we suggest using the latest compiler version at the time of contract deployment, which is 0.8.18 at the time of this audit. Using the latest compiler version is always recommended which prevents any compiler level issues.

(5) Unused function `swapAndLiquify`:

```
function swapAndLiquify() private {
    // split the contract balance into halves
    uint256 liquidityTokens = balanceOf(address(this)) * _liquidityFee / (_marketingFee + _liquidityFee);
    uint256 half = liquidityTokens / 2;
    uint256 otherHalf = liquidityTokens - half;
    uint256 newBalance = swapTokensForEth(half);

    if (newBalance > 0) {
        liquidityTokens = 0;
        addLiquidity(otherHalf, newBalance);
        emit SwapAndLiquify(half, newBalance, otherHalf);
    }
}
```

The `swapAndLiquify` function is unused in the contract.

Resolution: Suggest removing if not used.

(6) An addLiquidity function is called by swapAndLiquify function which is in turn unused:

```
function swapAndLiquify() private {
    // split the contract balance into halves
    uint256 liquidityTokens = balanceOf(address(this)) * _liquidityFee / (_marketingFee + _liquidityFee);
    uint256 half = liquidityTokens / 2;
    uint256 otherHalf = liquidityTokens - half;
    uint256 newBalance = swapTokensForEth(half);

    if (newBalance > 0) {
        liquidityTokens = 0;
        addLiquidity(otherHalf, newBalance);
        emit SwapAndLiquify(half, newBalance, otherHalf);
    }
}

function swapTokensForEth(uint256 tokenAmount) private returns (uint256) { ...
}

function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(uniswapV2Router), tokenAmount);
}
```

An addLiquidity function is called by swapAndLiquify function which is in turn unused.

Resolution: Suggest removing if not used.

Conclusion

We were given a contract code in the form of a bscscan.com link and have used all possible tests based on the given object. So it is ready to go for production. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

The security state of the reviewed contract is "**Secure**".

Note For Contract Users

Technical auditing does not guarantee the project's ethical side.

Please do your due diligence before investing. Our audit report is never an investment advice.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyse the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

RD Auditors Disclaimer

The smart contracts given for audit have been analysed in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Because the total number of test cases are unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

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



Email: info@rdauditors.com

Website: www.rdauditors.com

