



Loop Network Smart Contract, Code Review and Security Analysis Report

Customer: Loop Network
Prepared on: 5th April 2022
Platform: BSC
Language: Solidity

rdauditors.com

Table of Contents

Disclaimer	2
Document	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 - Loop Network	13
Code Flow Diagram - Slither Results Log	14
Severity Definitions	18
Audit Findings	19
Conclusion	20
Note For Contract Users	21
Our Methodology	22
Disclaimers	24

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.

Document

Name	Smart Contract Code Review and Security Analysis Report of Loop Network
Platform	BSC / Solidity
File	LoopNetwork.sol
MD5 hash	8EE328EFA0C4C6DD4F19716FDB7699A6
SHA256 hash	BA868FADD2F865B5B0778384AAB5CE4A39BE500A0FF876567887E7E8A34FEED5
Date	5/4/2022

Introduction

RD Auditors (Consultant) were contracted by Loop Network (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 31st March 2022 - 5th April 2022.

This contract consists of one file.

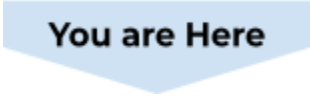
Project Scope

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 **Well-Secured**.




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	0
 Critical	0
 High	0
 Medium	0
 Low	0
 Very Low	0

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 Loop Network 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 the Loop Network code as a link.

<https://bscscan.com/address/0xce186ad6430e2fe494a22c9edbd4c68794a28b35#code>

The hash of that file is mentioned 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

LoopNetwork.sol

File And Function Level Report

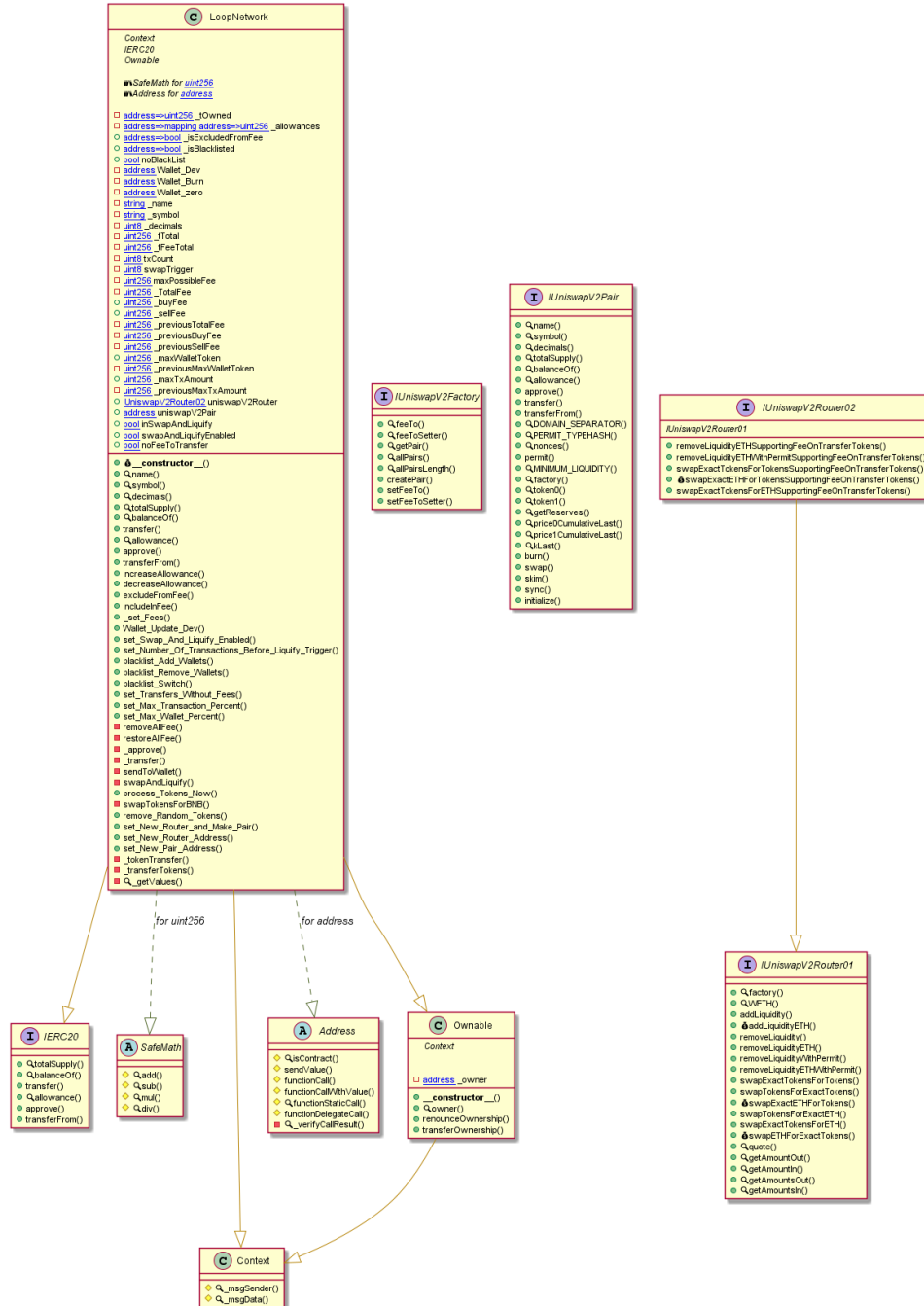
File: LoopNetwork.sol
Contract: LoopNetwork
Inherit: Context, IERC20, Ownable
Observation: Passed
Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	name	read	Passed	All Passed	No Issue	Passed
2	symbol	read	Passed	All Passed	No Issue	Passed
3	decimals	read	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	balanceOf	read	Passed	All Passed	No Issue	Passed
9	approve	write	Passed	All Passed	No Issue	Passed
10	transferFrom	write	Passed	All Passed	No Issue	Passed
11	increaseAllowance	write	Passed	All Passed	No Issue	Passed
12	decreaseAllowance	write	Passed	All Passed	No Issue	Passed

13	excludeFromFee	write	Passed	All Passed	No Issue	Passed
14	includeInFee	write	Passed	All Passed	No Issue	Passed
15	_set_Fees	write	Passed	All Passed	No Issue	Passed
16	Wallet_Update_Dev	write	Passed	All Passed	No Issue	Passed
17	set_Swap_And_Liquify_Enabled	write	Passed	All Passed	No Issue	Passed
18	set_Number_Of_Transactions_Before_Liquify_Trigger	write	Passed	All Passed	No Issue	Passed
19	receive	fallback	Passed	All Passed	No Issue	Passed
20	blacklist_Add_Wallets	write	Passed	All Passed	No Issue	Passed
21	blacklist_Remove_Wallets	write	Passed	All Passed	No Issue	Passed
22	blacklist_Switch	write	Passed	All Passed	No Issue	Passed
23	set_Transfers_Without_Fees	write	Passed	All Passed	No Issue	Passed
24	set_Max_Transaction_Percent	write	Passed	All Passed	No Issue	Passed
25	set_Max_Wallet_Percent	write	Passed	All Passed	No Issue	Passed
26	removeAllFee	write	Passed	All Passed	No Issue	Passed
27	restoreAllFee	write	Passed	All Passed	No Issue	Passed
28	_approve	write	Passed	All Passed	No Issue	Passed
29	_transfer	write	Passed	All Passed	No Issue	Passed
30	sendToWallet	write	Passed	All Passed	No Issue	Passed
31	swapAndLiquify	write	Passed	All Passed	No Issue	Passed
32	process_Tokens_Now	write	Passed	All Passed	No Issue	Passed

33	swapTokensForBNB	write	Passed	All Passed	No Issue	Passed
34	remove_Random_Tokens	write	Passed	All Passed	No Issue	Passed
35	set_New_Router_and_Make_Pair	write	Passed	All Passed	No Issue	Passed
36	set_New_Router_Address	write	Passed	All Passed	No Issue	Passed
37	set_New_Pair_Address	write	Passed	All Passed	No Issue	Passed
38	_tokenTransfer	write	Passed	All Passed	No Issue	Passed
39	_transferTokens	write	Passed	All Passed	No Issue	Passed
40	_getValues	read	Passed	All Passed	No Issue	Passed

Code Flow Diagram - Loop Network



Code Flow Diagram - Slither Results Log

```
blacklist_Switch(bool) should be declared external:
- LoopNetwork.blacklist_Switch(bool) (LoopNetwork.sol#682-684)
process_Tokens_Now(uint256) should be declared external:
- LoopNetwork.process_Tokens_Now(uint256) (LoopNetwork.sol#884-891)
set_New_Router_and_Make_Pair(address) should be declared external:
- LoopNetwork.set_New_Router_and_Make_Pair(address) (LoopNetwork.sol#933-937)
set_New_Router_Address(address) should be declared external:
- LoopNetwork.set_New_Router_Address(address) (LoopNetwork.sol#940-943)
set_New_Pair_Address(address) should be declared external:
- LoopNetwork.set_New_Pair_Address(address) (LoopNetwork.sol#946-948)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:LoopNetwork.sol analyzed (10 contracts with 75 detectors), 128 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

```
Variable LoopNetwork._maxWalletToken (LoopNetwork.sol#432) is not in mixedCase
Variable LoopNetwork._maxTxAmount (LoopNetwork.sol#437) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (LoopNetwork.sol#64)" inContext (LoopNetwork.sol#58-67)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Reentrancy in LoopNetwork._transfer(address,address,uint256) (LoopNetwork.sol#765-854):
  External calls:
  - swapAndLiquify(contractTokenBalance) (LoopNetwork.sol#831)
  - wallet.transfer(amount) (LoopNetwork.sol#870)
  State variables written after the call(s):
  - _TotalFee = _buyFee (LoopNetwork.sol#851)
  - _TotalFee = _sellFee (LoopNetwork.sol#851)
  - _tokenTransfer(from,to,amount,takeFee) (LoopNetwork.sol#853)
  - _TotalFee = _previousTotalFee (LoopNetwork.sol#749)
  - _TotalFee = 0 (LoopNetwork.sol#742)
  - _tokenTransfer(from,to,amount,takeFee) (LoopNetwork.sol#853)
  - _buyFee = _previousBuyFee (LoopNetwork.sol#750)
  - _buyFee = 0 (LoopNetwork.sol#740)
  - _tokenTransfer(from,to,amount,takeFee) (LoopNetwork.sol#853)
  - _previousBuyFee = _buyFee (LoopNetwork.sol#737)
  - _tokenTransfer(from,to,amount,takeFee) (LoopNetwork.sol#853)
  - _previousSellFee = _sellFee (LoopNetwork.sol#738)
  - _tokenTransfer(from,to,amount,takeFee) (LoopNetwork.sol#853)
  - _previousTotalFee = _TotalFee (LoopNetwork.sol#739)
  - _tokenTransfer(from,to,amount,takeFee) (LoopNetwork.sol#853)
  - _sellFee = _previousSellFee (LoopNetwork.sol#751)
  - _sellFee = 0 (LoopNetwork.sol#741)
  - _tokenTransfer(from,to,amount,takeFee) (LoopNetwork.sol#853)
  - _tOwned[sender] = _tOwned[sender].sub(tAmount) (LoopNetwork.sol#974)
  - _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount) (LoopNetwork.sol#975)
  - _tOwned[address(this)] = _tOwned[address(this)].add(tDev) (LoopNetwork.sol#976)
  - _tokenTransfer(from,to,amount,takeFee) (LoopNetwork.sol#853)
  - txCount ++ (LoopNetwork.sol#963)
```

```
LoopNetwork._previousMaxTxAmount (LoopNetwork.sol#438) is set pre-construction with a non-constant function or state variable
- _maxTxAmount
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state-variables
INFO:Detectors:
Pragma version^0.8.4 (LoopNetwork.sol#6) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (LoopNetwork.sol#78-82):
- (success) = recipient.call{value: amount}() (LoopNetwork.sol#80)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (LoopNetwork.sol#96-101):
- (success,returndata) = target.call{value: value}(data) (LoopNetwork.sol#99)
Low level call in Address.functionStaticCall(address,bytes,string) (LoopNetwork.sol#107-111):
- (success,returndata) = target.staticcall(data) (LoopNetwork.sol#109)
Low level call in Address.functionDelegateCall(address,bytes,string) (LoopNetwork.sol#118-122):
- (success,returndata) = target.delegatecall(data) (LoopNetwork.sol#120)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function IUniswapV2Pair.DOMAIN_SEPARATOR() (LoopNetwork.sol#200) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (LoopNetwork.sol#201) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (LoopNetwork.sol#214) is not in mixedCase
Function IUniswapV2Router01.WETH() (LoopNetwork.sol#231) is not in mixedCase
Function LoopNetwork._set_Fees(uint256,uint256) (LoopNetwork.sol#585-591) is not in mixedCase
Parameter LoopNetwork._set_Fees(uint256,uint256).Buy_Fee (LoopNetwork.sol#585) is not in mixedCase
Parameter LoopNetwork._set_Fees(uint256,uint256).Sell_Fee (LoopNetwork.sol#585) is not in mixedCase
Function LoopNetwork.Wallet_Update_Dev(address) (LoopNetwork.sol#596-599) is not in mixedCase
Function LoopNetwork.set_Swap_And_Liquify_Enabled(bool) (LoopNetwork.sol#609-612) is not in mixedCase
Parameter LoopNetwork.set_Swap_And_Liquify_Enabled(bool).true_or_false (LoopNetwork.sol#609) is not in mixedCase
Function LoopNetwork.set_Number_Of_Transactions_Before_Liquify_Trigger(uint8) (LoopNetwork.sol#615-617) is not in mixedCase
Parameter LoopNetwork.set_Number_Of_Transactions_Before_Liquify_Trigger(uint8).number_of_transactions (LoopNetwork.sol#615)
not in mixedCase
Function LoopNetwork.blacklist_Add_Wallets(address[]) (LoopNetwork.sol#637-650) is not in mixedCase
Function LoopNetwork.blacklist_Remove_Wallets(address[]) (LoopNetwork.sol#655-668) is not in mixedCase
Function LoopNetwork.blacklist_Switch(bool) (LoopNetwork.sol#682-684) is not in mixedCase
Parameter LoopNetwork.blacklist_Switch(bool).true_or_false (LoopNetwork.sol#682) is not in mixedCase
Function LoopNetwork.set_Transfers_Without_Fees(bool) (LoopNetwork.sol#699-701) is not in mixedCase
Parameter LoopNetwork.set_Transfers_Without_Fees(bool).true_or_false (LoopNetwork.sol#699) is not in mixedCase
```

```
INFO:Detectors:
LoopNetwork.allowance(address,address).owner (LoopNetwork.sol#524) shadows:
- Ownable.owner() (LoopNetwork.sol#152-154) (function)
LoopNetwork._approve(address,address,uint256).owner (LoopNetwork.sol#757) shadows:
- Ownable.owner() (LoopNetwork.sol#152-154) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
LoopNetwork.Wallet_Update_Dev(address).wallet (LoopNetwork.sol#596) lacks a zero-check on :
- Wallet_Dev = wallet (LoopNetwork.sol#597)
LoopNetwork.set_New_Pair_Address(address).newPair (LoopNetwork.sol#946) lacks a zero-check on :
- uniswapV2Pair = newPair (LoopNetwork.sol#947)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in LoopNetwork._transfer(address,address,uint256) (LoopNetwork.sol#765-854):
External calls:
- swapAndLiquify(contractTokenBalance) (LoopNetwork.sol#831)
- uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.
mestamp) (LoopNetwork.sol#901-907)
External calls sending eth:
- swapAndLiquify(contractTokenBalance) (LoopNetwork.sol#831)
- wallet.transfer(amount) (LoopNetwork.sol#870)
State variables written after the call(s):
- _TotalFee = _buyFee (LoopNetwork.sol#851)
- _TotalFee = _sellFee (LoopNetwork.sol#851)
- _tokenTransfer(from,to,amount,takeFee) (LoopNetwork.sol#853)
- _TotalFee = _previousTotalFee (LoopNetwork.sol#749)
- _TotalFee = 0 (LoopNetwork.sol#742)
- _tokenTransfer(from,to,amount,takeFee) (LoopNetwork.sol#853)
- _buyFee = _previousBuyFee (LoopNetwork.sol#750)
- _buyFee = 0 (LoopNetwork.sol#740)
- _tokenTransfer(from,to,amount,takeFee) (LoopNetwork.sol#853)
- _previousBuyFee = _buyFee (LoopNetwork.sol#737)
- _tokenTransfer(from,to,amount,takeFee) (LoopNetwork.sol#853)
- _previousSellFee = _sellFee (LoopNetwork.sol#738)
- _tokenTransfer(from,to,amount,takeFee) (LoopNetwork.sol#853)
- _previousTotalFee = _TotalFee (LoopNetwork.sol#739)
- _tokenTransfer(from,to,amount,takeFee) (LoopNetwork.sol#853)
```


Solidity Static Analysis

Constant/View/Pure functions: ✕

IUniswapV2Router02.swapExactTokensForTokensSupportingFeeOnTransferTokens(uint256,uint256,address[],address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 352:4:

Constant/View/Pure functions: ✕

IUniswapV2Router02.swapExactTokensForETHSupportingFeeOnTransferTokens(uint256,uint256,address[],address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 365:4:

ERC

ERC20: ✕

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

[more](#)

Pos: 204:4:

Miscellaneous

Constant/View/Pure functions: ✕

IERC20.transfer(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 24:4:

For loop over dynamic array: ✕

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

[more](#)

Pos: 653:4:

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: 917:12:

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:

No very low severity vulnerabilities were found.

Conclusion

We were given a contract file 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 “**Well-Secured**”.

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.

1. The owner has the option to adjust the amount of tokens which can be bought or sold in one transaction “_maxTxAmount”.

```
.....  
  
// Limit the maximum number of tokens that can be bought or sold in one transaction  
if (from != owner() && to != owner())  
    require(amount <= maxTxAmount, "You are trying to buy more than the max transaction limit.");  
  
/*
```

2. The owner has the option to blacklist wallet addresses “_isBlacklisted”.

```
/*  
BLACKLIST RESTRICTIONS  
*/  
if (noBlackList){  
    require(!_isBlacklisted[from] && !_isBlacklisted[to], "This address is blacklisted. Transaction reverted.");  
  
    require(from != address(0) && to != address(0), "ERR: Using 0 address!");  
    require(amount > 0, "Token value must be higher than zero.");  
  
/*
```

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

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

