RD AUDITORS

Rabbit Finance 2023, Contract,Code Review and Security Analysis Report

Customer: Rabbit Finance 2023
Prepared on: 14th December 2022
Platform: Binance
Language: Solidity

rdauditors.com

Table of Contents

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 Rabbit Finance 2023
Platform	Binance/ Solidity
File	RB.sol
MD5 hash	a2cd406f15903002edba1b0fed555da1
SHA256 hash	825032a7bfabcd506232ab24657b9cf42f2dc0af98541d35d8f749bd29d4cf16
Date	14/12/2022

Introduction

RD Auditors (Consultant) were contracted by Rabbit Finance 2023 (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 11th - 14th December 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 **Poorly-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	1
Critical	1
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 Rabbit Finance 2023 team has not provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Overall, the code is almost commented. Commenting can provide rich documentation for functions, return variables and more. Use of the Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

We were given the Rabbit Finance 2023 code as a link:

https://bscscan.com/token/0x6427B760730f2c6a62a7B8aD6274eE4964476d5C#code

The hash of that file is mentioned in the table. As mentioned above, it's recommended to write comments on 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

RB.sol

File And Function Level Report

Abstract: Context

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	_msgSender	internal	Passed	All Passed	No Issue	Passed
2	_msgData	internal	Passed	All Passed	No Issue	Passed

Interface: IBEP20

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	totalSupply	read	Passed	All Passed	No Issue	Passed
2	balanceOf	read	Passed	All Passed	No Issue	Passed
3	transfer	external	Passed	All Passed	No Issue	Passed
4	allowance	read	Passed	All Passed	No Issue	Passed
5	approve	external	Passed	All Passed	No Issue	Passed
6	transferFrom	external	Passed	All Passed	No Issue	Passed

Interface: IBEP20Metadata

Inherit: IBEP20

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

Contract: BEP20

Inherit: Context, IBEP20, IBEP20Metadata

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	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	_transfer	internal	Passed	All Passed	No Issue	Passed
13	_tokengeneration	internal	Passed	All Passed	No Issue	Passed
14	_approve	internal	Passed	All Passed	No Issue	Passed

Contract: Ownable

Inherit: Context

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	Owner	read	Passed	All Passed	No Issue	Passed
2	renounceOwnership	onlyowner	Passed	All Passed	No Issue	Passed
3	transferOwnership	onlyOwner	Passed	All Passed	No Issue	Passed
4	_setOwner	write	Passed	All Passed	No Issue	Passed

Interface: IRouter:

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	factory	external	Passed	All Passed	No Issue	Passed
2	WETH	external	Passed	All Passed	No Issue	Passed
3	addLiquidityETH	external	Passed	All Passed	No Issue	Passed
4	SwapExactTokensForETHSupportingFeeOnTransferTokens	external	Passed	All Passed	No Issue	Passed

Contract: RB

Inherit: BEP20, Ownable

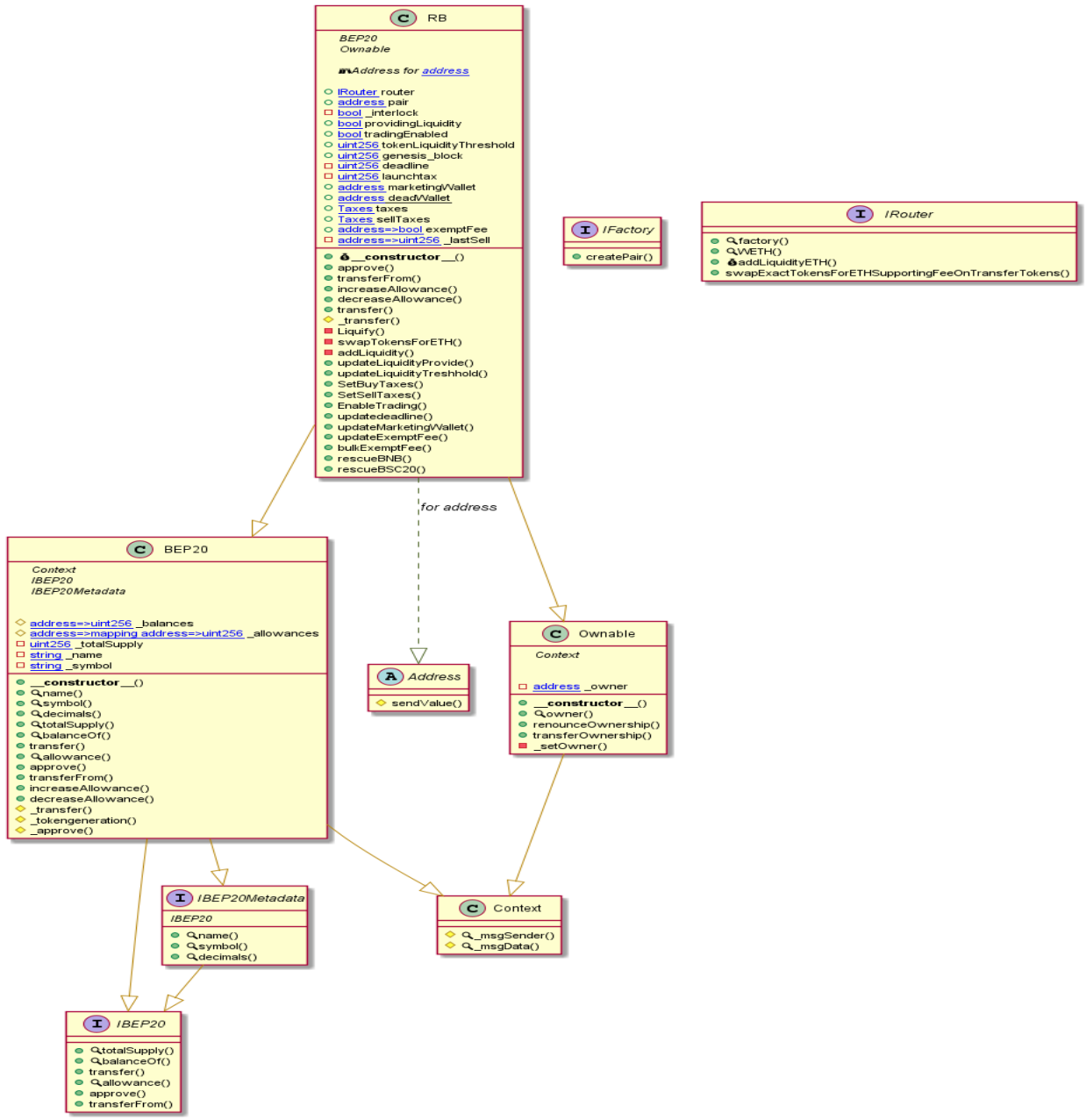
Observation: Not Passed

Test Report: Not Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	approve	write	Passed	All Passed	No Issue	Passed
2	transferFrom	write	Passed	All Passed	No Issue	Passed
3	increaseAllowance	write	Passed	All Passed	No Issue	Passed

4	decreaseAllowance	write	Passed	All Passed	No Issue	Passed
5	transfer	write	Passed	All Passed	No Issue	Passed
6	_transfer	internal	Passed	All Passed	No Issue	Passed
7	Liquify	write	Not Passed	Not Passed	Not passed	Not Passed
8	SwapTokensForETH	write	Passed	All Passed	No Issue	Passed
9	addLiquidity	write	Passed	All Passed	No Issue	Passed
10	updateLiquidityProvide	onlyowner	Passed	All Passed	No Issue	Passed
11	UpdateLiquidityTreshhold	onlyowner	Passed	All Passed	No Issue	Passed
12	setBuyTaxes	onlyowner	Passed	All Passed	No Issue	Passed
13	SetSellTaxes	onlyowner	Passed	All Passed	No Issue	Passed
14	EnableTrading	onlyowner	Passed	All Passed	No Issue	Passed
15	Updatedeadline	onlyowner	Passed	All Passed	No Issue	Passed
16	UpdateMarketingWallet	onlyowner	Passed	All Passed	No Issue	Passed
17	bulkExemptFee	onlyowner	Passed	All Passed	No Issue	Passed
18	rescueBNB	onlyowner	Passed	All Passed	No Issue	Passed
19	rescueBSC20	onlyowner	Passed	All Passed	No Issue	Passed

Code Flow Diagram - RB.sol



Code Flow Diagram - Slither Results Log

File: RB.sol

```
INFO:Detectors:
RB.rescueBNB(uint256) (RB.sol#680-682) sends eth to arbitrary user
        Dangerous calls:
        - address(owner()).transfer(weiAmount) (RB.sol#681)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
Reentrancy in RB._transfer(address,address,uint256) (RB.sol#493-551):
        External calls:
        - Liquify(feeswap,currentTaxes) (RB.sol#539)
                - router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (RB.sol#61
5-622)
                - (success) = recipient.call{value: amount}() (RB.sol#317)
                - router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (
RB.sol#601-607)
                - address(marketingWallet).sendValue(marketingAmt) (RB.sol#586)
        External calls sending eth:
        - Liquify(feeswap,currentTaxes) (RB.sol#539)
                - router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (RB.sol#61
5-622)
                - (success) = recipient.call{value: amount}() (RB.sol#317)
        State variables written after the call(s):
        - super._transfer(sender,recipient,amount - fee) (RB.sol#542)
                - _balances[sender] = senderBalance - amount (RB.sol#270)
                - _balances[recipient] += amount (RB.sol#271)
        - super._transfer(sender,address(this),feeAmount) (RB.sol#547)
                - _balances[sender] = senderBalance - amount (RB.sol#270)
                - _balances[recipient] += amount (RB.sol#271)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
RB.rescueBSC20(address,uint256) (RB.sol#684-687) ignores return value by IBEP20(tokenAdd).transfer(owner(),amount) (RB.sol#686)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
```

```
INFO:Detectors:
RB.Liquify(uint256,RB.Taxes) (RB.sol#553-590) performs a multiplication on the result of a division:
        -unitBalance = deltaBalance / (denominator - swapTaxes.liquidity) (RB.sol#576)
        -ethToAddLiquidityWith = unitBalance * swapTaxes.liquidity (RB.sol#577)
RB.Liquify(uint256,RB.Taxes) (RB.sol#553-590) performs a multiplication on the result of a division:
        -unitBalance = deltaBalance / (denominator - swapTaxes.liquidity) (RB.sol#576)
        -marketingAmt = unitBalance * 2 * swapTaxes.marketing (RB.sol#584)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
RB._transfer(address,address,uint256).currentTaxes (RB.sol#507) is a local variable never initialized
RB._transfer(address,address,uint256).feeswap (RB.sol#504) is a local variable never initialized
RB._transfer(address,address,uint256).feesum (RB.sol#505) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
RB.addLiquidity(uint256,uint256) (RB.sol#610-623) ignores return value by router.addLiquidityETH{value: ethAmount}(address(this
),tokenAmount,0,0,deadWallet,block.timestamp) (RB.sol#615-622)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
RB._transfer(address,address,uint256).fee (RB.sol#506) is written in both
        fee = 0 (RB.sol#515)
        fee = (amount * feesum) / 100 (RB.sol#535)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#write-after-write
```

```
INFO:Detectors:
Modifier RB.lockTheSwap() (RB.sol#422-428) does not always execute _; or revertReference: https://github.com/crytic/slither/wik
i/Detector-Documentation#incorrect-modifier
INFO:Detectors:
Reentrancy in RB.Liquify(uint256,RB.Taxes) (RB.sol#553-590):
        External calls:
        - swapTokensForETH(toSwap) (RB.sol#573)
                - router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (
RB.sol#601-607)
        - addLiquidity(tokensToAddLiquidityWith,ethToAddLiquidityWith) (RB.sol#581)
                - router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (RB.sol#61
5-622)
        External calls sending eth:
        - addLiquidity(tokensToAddLiquidityWith,ethToAddLiquidityWith) (RB.sol#581)
                - router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (RB.sol#61
5-622)
        State variables written after the call(s):
        - addLiquidity(tokensToAddLiquidityWith,ethToAddLiquidityWith) (RB.sol#581)
                - _allowances[owner][spender] = amount (RB.sol#308)
Reentrancy in RB.constructor() (RB.sol#430-446):
        External calls:
        - _pair = IFactory(_router.factory()).createPair(address(this),_router.WETH()) (RB.sol#436)
        State variables written after the call(s):
        - exemptFee[address(this)] = true (RB.sol#440)
        - exemptFee[marketingWallet] = true (RB.sol#441)
        - exemptFee[deadWallet] = true (RB.sol#442)
        - exemptFee[0xD152f549545093347A162Dce210e7293f1452150] = true (RB.sol#443)
        - exemptFee[0x407993575c91ce7643a4d4cCACc9A98c36eE1BBE] = true (RB.sol#444)
        - pair = _pair (RB.sol#439)
        - router = _router (RB.sol#438)
Reentrancy in RB.transferFrom(address,address,uint256) (RB.sol#453-465):
        External calls:
        - _transfer(sender,recipient,amount) (RB.sol#458)
                - router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (RB.sol#61
5-622)
                - (success) = recipient.call{value: amount}() (RB.sol#317)
```

```
                - router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (
RB.sol#601-607)
                - address(marketingWallet).sendValue(marketingAmt) (RB.sol#586)
        External calls sending eth:
        - _transfer(sender,recipient,amount) (RB.sol#458)
                - router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (RB.sol#61
5-622)
                - (success) = recipient.call{value: amount}() (RB.sol#317)
        State variables written after the call(s):
        - _approve(sender,_msgSender(),currentAllowance - amount) (RB.sol#462)
                - _allowances[owner][spender] = amount (RB.sol#308)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in RB.Liquify(uint256,RB.Taxes) (RB.sol#553-590):
        External calls:
        - swapTokensForETH(toSwap) (RB.sol#573)
                - router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (
RB.sol#601-607)
        - addLiquidity(tokensToAddLiquidityWith,ethToAddLiquidityWith) (RB.sol#581)
                - router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (RB.sol#61
5-622)
        External calls sending eth:
        - addLiquidity(tokensToAddLiquidityWith,ethToAddLiquidityWith) (RB.sol#581)
                - router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (RB.sol#61
5-622)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (RB.sol#309)
                - addLiquidity(tokensToAddLiquidityWith,ethToAddLiquidityWith) (RB.sol#581)
Reentrancy in RB._transfer(address,address,uint256) (RB.sol#493-551):
        External calls:
        - Liquify(feeswap,currentTaxes) (RB.sol#539)
                - router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (RB.sol#61
5-622)
                - (success) = recipient.call{value: amount}() (RB.sol#317)
                - router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (
RB.sol#601-607)
                - address(marketingWallet).sendValue(marketingAmt) (RB.sol#586)
```

```
        External calls sending eth:
        - Liquify(feeswap,currentTaxes) (RB.sol#539)
                - router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (RB.sol#61
5-622)
                - (success) = recipient.call{value: amount}() (RB.sol#317)
        Event emitted after the call(s):
        - Transfer(sender,recipient,amount) (RB.sol#273)
                - super._transfer(sender,address(this),feeAmount) (RB.sol#547)
        - Transfer(sender,recipient,amount) (RB.sol#273)
                - super._transfer(sender,recipient,amount - fee) (RB.sol#542)
Reentrancy in RB.transferFrom(address,address,uint256) (RB.sol#453-465):
        External calls:
        - _transfer(sender,recipient,amount) (RB.sol#458)
                - router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (RB.sol#61
5-622)
                - (success) = recipient.call{value: amount}() (RB.sol#317)
                - router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (
RB.sol#601-607)
                - address(marketingWallet).sendValue(marketingAmt) (RB.sol#586)
        External calls sending eth:
        - _transfer(sender,recipient,amount) (RB.sol#458)
                - router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (RB.sol#61
5-622)
                - (success) = recipient.call{value: amount}() (RB.sol#317)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (RB.sol#309)
                - _approve(sender,_msgSender(),currentAllowance - amount) (RB.sol#462)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Context._msgData() (RB.sol#14-17) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.4 (RB.sol#7) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

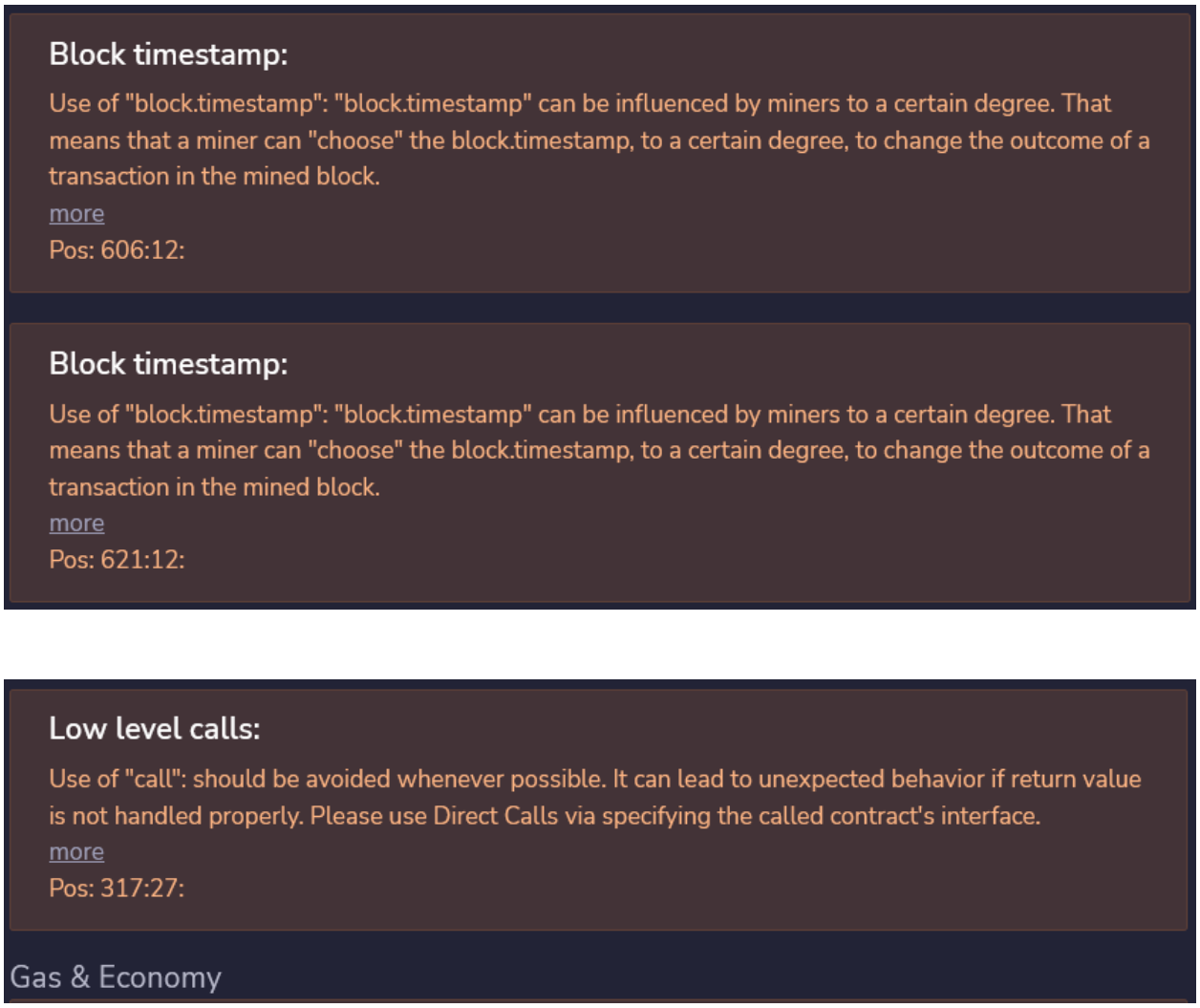
```
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (RB.sol#314-319):
        - (success) = recipient.call{value: amount}() (RB.sol#317)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
INFO:Detectors:
Redundant expression "this (RB.sol#15)" inContext (RB.sol#9-18)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
RB.slitherConstructorConstantVariables() (RB.sol#390-692) uses literals with too many digits:
        - deadWallet = 0x000000000000000000000000000000000000dEaD (RB.sol#407)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
RB._lastSell (RB.sol#420) is never used in RB (RB.sol#390-692)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
INFO:Detectors:
RB.launchtax (RB.sol#404) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
name() should be declared external:
        - BEP20.name() (RB.sol#86-88)
symbol() should be declared external:
        - BEP20.symbol() (RB.sol#94-96)
totalSupply() should be declared external:
        - BEP20.totalSupply() (RB.sol#118-120)
transfer(address,uint256) should be declared external:
        - BEP20.transfer(address,uint256) (RB.sol#137-145)
        - RB.transfer(address,uint256) (RB.sol#488-491)
allowance(address,address) should be declared external:
        - BEP20.allowance(address,address) (RB.sol#150-158)
approve(address,uint256) should be declared external:
        - BEP20.approve(address,uint256) (RB.sol#167-170)
        - RB.approve(address,uint256) (RB.sol#448-451)
transferFrom(address,address,uint256) should be declared external:
        - BEP20.transferFrom(address,address,uint256) (RB.sol#185-197)
        - RB.transferFrom(address,address,uint256) (RB.sol#453-465)
increaseAllowance(address,uint256) should be declared external:
        - BEP20.increaseAllowance(address,uint256) (RB.sol#211-218)
        - RB.increaseAllowance(address,uint256) (RB.sol#467-474)
decreaseAllowance(address,uint256) should be declared external:
        - BEP20.decreaseAllowance(address,uint256) (RB.sol#234-244)
        - RB.decreaseAllowance(address,uint256) (RB.sol#476-486)
```

```
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (RB.sol#340-342)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (RB.sol#344-347)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:RB.sol analyzed (9 contracts with 75 detectors), 52 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

File: RB.sol



Gas costs:

Gas requirement of function RB.bulkExemptFee 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: 674:4:

Gas costs:

Gas requirement of function RB.bulkExemptFee 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: 674:4:

Gas costs:

Gas requirement of function RB.rescueBNB 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: 680:4:

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: 675:8:

Miscellaneous

Constant/View/Pure functions:

IBEP20.transfer(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 25:4:

Constant/View/Pure functions:

IRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(uint256,uint256,address[],address,uint2
: Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 381:4:

Similar variable names:

RB.swapTokensForETH(uint256) : Variables have very similar names "pair" and "path". Note: Modifiers are currently not considered by this static analysis.
Pos: 596:8:

Similar variable names:

RB.swapTokensForETH(uint256) : Variables have very similar names "pair" and "path". Note: Modifiers are currently not considered by this static analysis.
Pos: 604:12:

No return:

IRouter.addLiquidityETH(address,uint256,uint256,uint256,address,uint256): Defines a return type but never explicitly returns a value.
Pos: 365:4:

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: 666:8:

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: 685:8:

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: 567:47:

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: 576:34:

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:

Re-entrancy possibility in line #586, use transfer instead or cap call by gas, or update an storage before transfer or apply non-reentrant modifier.

```
553     function Liquify(uint256 feeswap, Taxes memory swapTaxes) private lockTheSwap {
554
555         if(feeswap == 0){
556             return;
557         }
558
559         uint256 contractBalance = balanceOf(address(this));
560         if (contractBalance >= tokenLiquidityThreshold) {
561             if (tokenLiquidityThreshold > 1) {
562                 contractBalance = tokenLiquidityThreshold;
563             }
564
565             // Split the contract balance into halves
566             uint256 denominator = feeswap * 2;
567             uint256 tokensToAddLiquidityWith = (contractBalance * swapTaxes.liquidity) /
568                 denominator;
569             uint256 toSwap = contractBalance - tokensToAddLiquidityWith;
570
571             uint256 initialBalance = address(this).balance;
572
573             swapTokensForETH(toSwap);
574
575             uint256 deltaBalance = address(this).balance - initialBalance;
576             uint256 unitBalance = deltaBalance / (denominator - swapTaxes.liquidity);
577             uint256 ethToAddLiquidityWith = unitBalance * swapTaxes.liquidity;
578
579             if (ethToAddLiquidityWith > 0) {
580                 // Add liquidity to pancake
581                 addLiquidity(tokensToAddLiquidityWith, ethToAddLiquidityWith);
582             }
583
584             uint256 marketingAmt = unitBalance * 2 * swapTaxes.marketing;
585             if (marketingAmt > 0) {
586                 payable(marketingWallet).sendValue(marketingAmt);
```

High:

No high severity vulnerabilities were found.

Medium:

No medium severity vulnerabilities were found.

Low:

No low severity vulnerabilities were found.

Very Low:

Discussion

1) Unused functions exist in the form of abstract, for example _msgData line 14 these can be removed to reduce code size.

2)Two step transfer of ownership recommended to avoid transfer at wrong address.

3)Many hard coded values, advised to double check before going to production.

Conclusion

We were given a contract file and have used all possible tests based on the given object. So it is now ready for mainnet deployment. 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 “**Poorly**-**Secured**”.

Note For Contract Users

There are several owner only functions. Those can be called by the owner's wallet only. So, if the owner's wallet is compromised, then it carries the risk of the contract becoming vulnerable.

rescueBSC20: The owner can't claim the contract's balance of its own token but can take another token from the contract.

```
684     function rescueBSC20(address tokenAdd, uint256 amount) external onlyOwner {
685         require(tokenAdd != address(this), "Owner can't claim contract's balance of its own tokens");
686         IBEP20(tokenAdd).transfer(owner(), amount);
687     }
688
```

rescueBNB: This function can be used to rescue BNB by the owner. By calling this function the owner can remove the contract balance.

```
679
680     function rescueBNB(uint256 weiAmount) external onlyOwner {
681         payable(owner()).transfer(weiAmount);
682     }
```

updateExemptFee: This function can be called by the admin to set state.
updateMarketingWallet: The owner can set a new marketingWallet address.
updateDeadline: The owner can update the deadline.
EnableTrading: The owner can enable trading.
setSellTaxes: The owner can set a marketing and liquidity tax percentage on any sells.
SetBuyTaxes: The owner can set marketing and liquidity tax percentage on any buys.
UpdateLiquidityTreshhold: The owner can update the threshold.

updateLiquidityProvide: The owner can update the liquidity providing state.
transferOwnership: The owner can transfer ownership.
renounceOwnership: The owner can renounceOwnership.

The owner has full control over the smart contract. Thus, 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