RD AUDITORS

Super Soccer, Code Review and Security Analysis Report

Customer: Super Soccer
Prepared on: 4th June 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 Super Soccer
Platform	BSC / Solidity
File	SuperSoccer.sol
MD5 hash	C443F5A339FF9D3AD085C8B0609E044A
SHA256 hash	0A6DB90FB3222F3B322B590ABCE24ED2FD0CA3BC1C360C704F0EC016C225B959
Date	04/06/2022

Introduction

RD Auditors (Consultant) were contracted by Super Soccer (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report represents the findings of the security assessment of the customer`s smart contracts and its code review conducted between 1st - 4th June 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 **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 Super Soccer 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 not 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 Super Soccer code as a link:

https://bscscan.com/address/0xFfe8219A3B8b50A0C4E8d14e5bCCcD9e7d153240#code

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

File And Function Level Report

Contract: 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

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	WaiveOwnership	onlyowner	Passed	All Passed	No Issue	Passed
3	transferOwnership	onlyowner	Passed	All Passed	No Issue	Passed
4	getUnlockTime	read	Passed	All Passed	No Issue	Passed
5	getTime	read	Passed	All Passed	No Issue	Passed
6	lock	onlyowner	Passed	All Passed	No Issue	Passed
7	unlock	write	Passed	All Passed	No Issue	Passed

Contract: Supersoccer

Inherit: Context, IERC20, Ownable

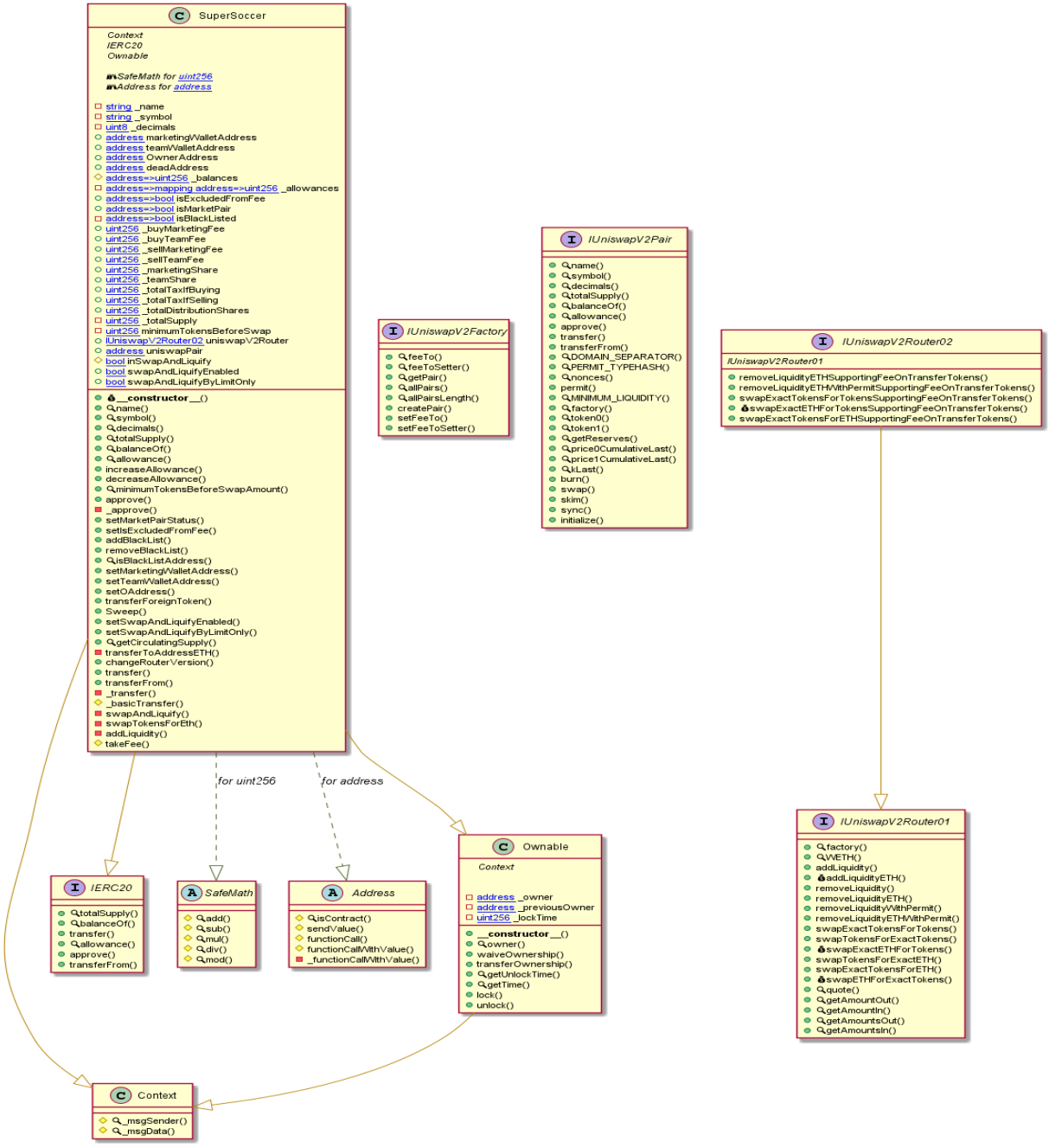
Observation: Passed

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	allowance	read	Passed	All Passed	No Issue	Passed
7	increaseAllowance	write	Passed	All Passed	No Issue	Passed
8	decreaseAllowance	write	Passed	All Passed	No Issue	Passed
9	minimumTokenBeforeSwapAmount	read	Passed	All Passed	No Issue	Passed
10	approve	write	Passed	All Passed	No Issue	Passed
11	_approve	private	Passed	All Passed	No Issue	Passed
12	SetMarketPairStatus	onlyowner	Passed	All Passed	No Issue	Passed
13	setIsExcludedFromFee	onlyowner	Passed	All Passed	No Issue	Passed
14	addBlackList	onlyowner	Passed	All Passed	No Issue	Passed
15	removeBlackList	onlyowner	Passed	All Passed	No Issue	Passed

16	isBlackListAddress	read	Passed	All Passed	No Issue	Passed
17	SetMarketingWalletAddress	onlyowner	Passed	All Passed	No Issue	Passed
18	setTeamWalletAddress	onlyowner	Passed	All Passed	No Issue	Passed
19	setOAddress	onlyowner	Passed	All Passed	No Issue	Passed
20	transferForeignToken	onlyowner	Passed	All Passed	No Issue	Passed
21	swap	onlyowner	Passed	All Passed	No Issue	Passed
22	SetSwapAndLiquifyEnabled		Passed	All Passed	No Issue	Passed
23	SetSwapAndLiquifyByLimitOnly	onlyowner	Passed	All Passed	No Issue	Passed
24	getCirculatingSupply	read	Passed	All Passed	No Issue	Passed
25	transferToAddressETH	private	Passed	All Passed	No Issue	Passed
26	ChangeRouterVersion	onlyowner	Passed	All Passed	No Issue	Passed
27	transfer	write	Passed	All Passed	No Issue	Passed
28	transferFrom	write	Passed	All Passed	No Issue	Passed
29	_transfer	private	Passed	All Passed	No Issue	Passed
30	_basicTransfer	internal	Passed	All Passed	No Issue	Passed
31	SwapAndLiquify	private	Passed	All Passed	No Issue	Passed
32	SwapTokensForETH	private	Passed	All Passed	No Issue	Passed
33	addLiquidity	private	Passed	All Passed	No Issue	Passed
34	takeFee	internal	Passed	All Passed	No Issue	Passed

Code Flow Diagram - Super Soccer



Code Flow Diagram - Slither Results Log

```
INFO:Detectors:
SuperSoccer.allowance(address,address).owner (SuperSoccer.sol#509) shadows:
        - Ownable.owner() (SuperSoccer.sol#156-158) (function)
SuperSoccer._approve(address,address,uint256).owner (SuperSoccer.sol#532) shadows:
        - Ownable.owner() (SuperSoccer.sol#156-158) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
SuperSoccer.setMarketingWalletAddress(address).newAddress (SuperSoccer.sol#564) lacks a zero-check on :
                - marketingWalletAddress = address(newAddress) (SuperSoccer.sol#565)
SuperSoccer.setTeamWalletAddress(address).newAddress (SuperSoccer.sol#568) lacks a zero-check on :
                - teamWalletAddress = address(newAddress) (SuperSoccer.sol#569)
SuperSoccer.setOAddress(address).newAddress (SuperSoccer.sol#572) lacks a zero-check on :
                - OwnerAddress = newAddress (SuperSoccer.sol#573)
SuperSoccer.Sweep(address).account (SuperSoccer.sol#582) lacks a zero-check on :
                - address(account).transfer(balance) (SuperSoccer.sol#584)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in SuperSoccer.changeRouterVersion(address) (SuperSoccer.sol#604-620):
        External calls:
        - newPairAddress = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (Su
perSoccer.sol#612-613)
        State variables written after the call(s):
        - isMarketPair[address(uniswapPair)] = true (SuperSoccer.sol#619)
        - uniswapPair = newPairAddress (SuperSoccer.sol#616)
        - uniswapV2Router = _uniswapV2Router (SuperSoccer.sol#617)
Reentrancy in SuperSoccer.constructor() (SuperSoccer.sol#466-487):
        External calls:
        - uniswapPair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (Super
Soccer.sol#470-471)
        State variables written after the call(s):
        - _allowances[address(this)][address(uniswapV2Router)] = _totalSupply (SuperSoccer.sol#474)
        - _balances[_msgSender()] = _totalSupply (SuperSoccer.sol#485)
        - _totalDistributionShares = _marketingShare.add(_teamShare) (SuperSoccer.sol#481)
        - _totalTaxIfBuying = _buyMarketingFee.add(_buyTeamFee) (SuperSoccer.sol#479)
        - _totalTaxIfSelling = _sellMarketingFee.add(_sellTeamFee) (SuperSoccer.sol#480)
        - isExcludedFromFee[owner()] = true (SuperSoccer.sol#476)
        - isExcludedFromFee[address(this)] = true (SuperSoccer.sol#477)
```

```
INFO:Detectors:
Ownable.unlock() (SuperSoccer.sol#191-196) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > _lockTime,Contract is locked until 7 days) (SuperSoccer.sol#193)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (SuperSoccer.sol#87-96) uses assembly
        - INLINE ASM (SuperSoccer.sol#94)
Address._functionCallWithValue(address,bytes,uint256,string) (SuperSoccer.sol#123-140) uses assembly
        - INLINE ASM (SuperSoccer.sol#132-135)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address._functionCallWithValue(address,bytes,uint256,string) (SuperSoccer.sol#123-140) is never used and should be removed
Address.functionCall(address,bytes) (SuperSoccer.sol#106-108) is never used and should be removed
Address.functionCall(address,bytes,string) (SuperSoccer.sol#110-112) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (SuperSoccer.sol#114-116) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (SuperSoccer.sol#118-121) is never used and should be removed
Address.isContract(address) (SuperSoccer.sol#87-96) is never used and should be removed
Address.sendValue(address,uint256) (SuperSoccer.sol#98-104) is never used and should be removed
Context._msgData() (SuperSoccer.sol#14-17) is never used and should be removed
SafeMath.mod(uint256,uint256) (SuperSoccer.sol#75-77) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (SuperSoccer.sol#79-82) is never used and should be removed
SuperSoccer.addLiquidity(uint256,uint256) (SuperSoccer.sol#717-730) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
SuperSoccer._totalSupply (SuperSoccer.sol#433) is set pre-construction with a non-constant function or state variable:
        - 2022 * 10 ** 4 * 10 ** _decimals
SuperSoccer.minimumTokensBeforeSwap (SuperSoccer.sol#434) is set pre-construction with a non-constant function or state variab
le:
        - 2022 * 10 ** _decimals
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state-variables
INFO:Detectors:
Pragma version^0.8.4 (SuperSoccer.sol#6) 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:
Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (Super
rSoccer.sol#271) is too similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uin
t256).amountBDesired (SuperSoccer.sol#272)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
SuperSoccer.slitherConstructorVariables() (SuperSoccer.sol#399-752) uses literals with too many digits:
        - deadAddress = 0x000000000000000000000000000000000000dEaD (SuperSoccer.sol#411)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
SuperSoccer._buyMarketingFee (SuperSoccer.sol#420) should be constant
SuperSoccer._buyTeamFee (SuperSoccer.sol#421) should be constant
SuperSoccer._decimals (SuperSoccer.sol#406) should be constant
SuperSoccer._marketingShare (SuperSoccer.sol#426) should be constant
SuperSoccer._name (SuperSoccer.sol#404) should be constant
SuperSoccer._sellMarketingFee (SuperSoccer.sol#423) should be constant
SuperSoccer._sellTeamFee (SuperSoccer.sol#424) should be constant
SuperSoccer._symbol (SuperSoccer.sol#405) should be constant
SuperSoccer._teamShare (SuperSoccer.sol#427) should be constant
SuperSoccer.deadAddress (SuperSoccer.sol#411) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
waiveOwnership() should be declared external:
        - Ownable.waiveOwnership() (SuperSoccer.sol#165-168)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (SuperSoccer.sol#170-174)
getUnlockTime() should be declared external:
        - Ownable.getUnlockTime() (SuperSoccer.sol#176-178)
getTime() should be declared external:
        - Ownable.getTime() (SuperSoccer.sol#180-182)
lock(uint256) should be declared external:
        - Ownable.lock(uint256) (SuperSoccer.sol#184-189)
```

Solidity Static Analysis

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Address._functionCallWithValue(address,bytes,uint256,string): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 123:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in SuperSoccer.swapTokensForEth(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 697:4:

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: 728:12:

Gas costs:

Gas requirement of function SuperSoccer.allowance 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: 509:4:

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: 549:8:

ERC

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type
more
Pos: 221:4:

Miscellaneous

Constant/View/Pure functions:

IUniswapV2Router02.swapExactTokensForETHSupportingFeeOnTransferTokens(uint256,uint256,add
: Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 390:4:

Similar variable names:

SuperSoccer.Sweep(address) : Variables have very similar names "_balances" and "balance". Note: Modifiers are currently not considered by this static analysis.
Pos: 584:34:

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: 640:8:

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: 69:20:

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 medium 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.

Discussion

1) Hardcoded addresses should be checked carefully before deployment.

2) There is no need of safemath in >= 0.8 solidity version.

3) _name, _symbol, _decimals and other similar things that are not being changed anywhere can be declared as constant, it will save gas.

Conclusion

We were given a contract file and have used all possible tests based on the given object. The contract is written systematically, 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 now “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.

SetMarketPairStatus: The owner can set market pairs.

SetIsExcludedFromFee: The owner can exclude any address from taking a fee.

AddBlackList: The owner can set any address as a blacklist.

RemoveBlackList: The owner can remove any addresses from the blacklist.

SetMarketingWalletAddress: The owner can set a marketing wallet address.

SetTeamWalletAddress: The owner can set the team wallet address.

TransferForeignToken: The owner can take all tokens except SPS from contract via this function.

```
599
600     function transferToAddressETH(address payable recipient, uint256 amount) private {
601         recipient.transfer(amount);
602     }
```

Sweep: Owner can withdraw all the coins from the contract.

```
582     function Sweep(address account) external onlyOwner {
583         uint256 balance = address(this).balance;
584         payable(account).transfer(balance);
585     }
```

SetSwapAndLiquifyEnabled: The owner can enable this function.

ChangeRouterVersion: The owner can change the router version.

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