



**RD
AUDITORS**

SuperCluster, Smart Contract, Code Review and Security Analysis Report

Customer: SuperCluster
Prepared on: 22nd May, 2023
Platform: Binance
Language: Solidity

rdauditors.com

Table of Contents

Disclaimer	1
Documentation	2
Introduction	2
Project Scope	3
Executive Summary	5
Code Quality	6
Documentation	7
Use of Dependencies	8
AS-IS Overview	8
Code Flow Diagram - SuperCluster.sol	12
Code Flow Diagram - Slither Results Log	12
Severity Definitions	12
Audit Findings	20
Conclusion	20
Note For Contract Users	21
Our Methodology	22
Disclaimers	25

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 SuperCluster
Platform	Binance/ Solidity
File 1	SuperCluster.sol
MD5 hash	187ffe832a9fcd1884a5694eb1a976af
SHA256 hash	31b32b7b317b60b7b5a5ae09edc01572916719b1e0986cb8a755ed252659f47e
Date	22/05/2023

Introduction

RD Auditors (Consultant) were contracted by SuperCluster(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 18th- 22nd May, 2023.

This contract consists of one file.

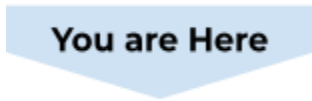
Project Scope

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 **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 SuperCluster 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 Moonbase Alpha smart contract code:

<https://bscscan.com/token/0x111d4ef5f450855ac0c41c9225215b7985efdbe1?a=0x10e8c94ae35757c60368218ab31b96408cc2420c>

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

SuperCluster.sol

File And Function Level Report

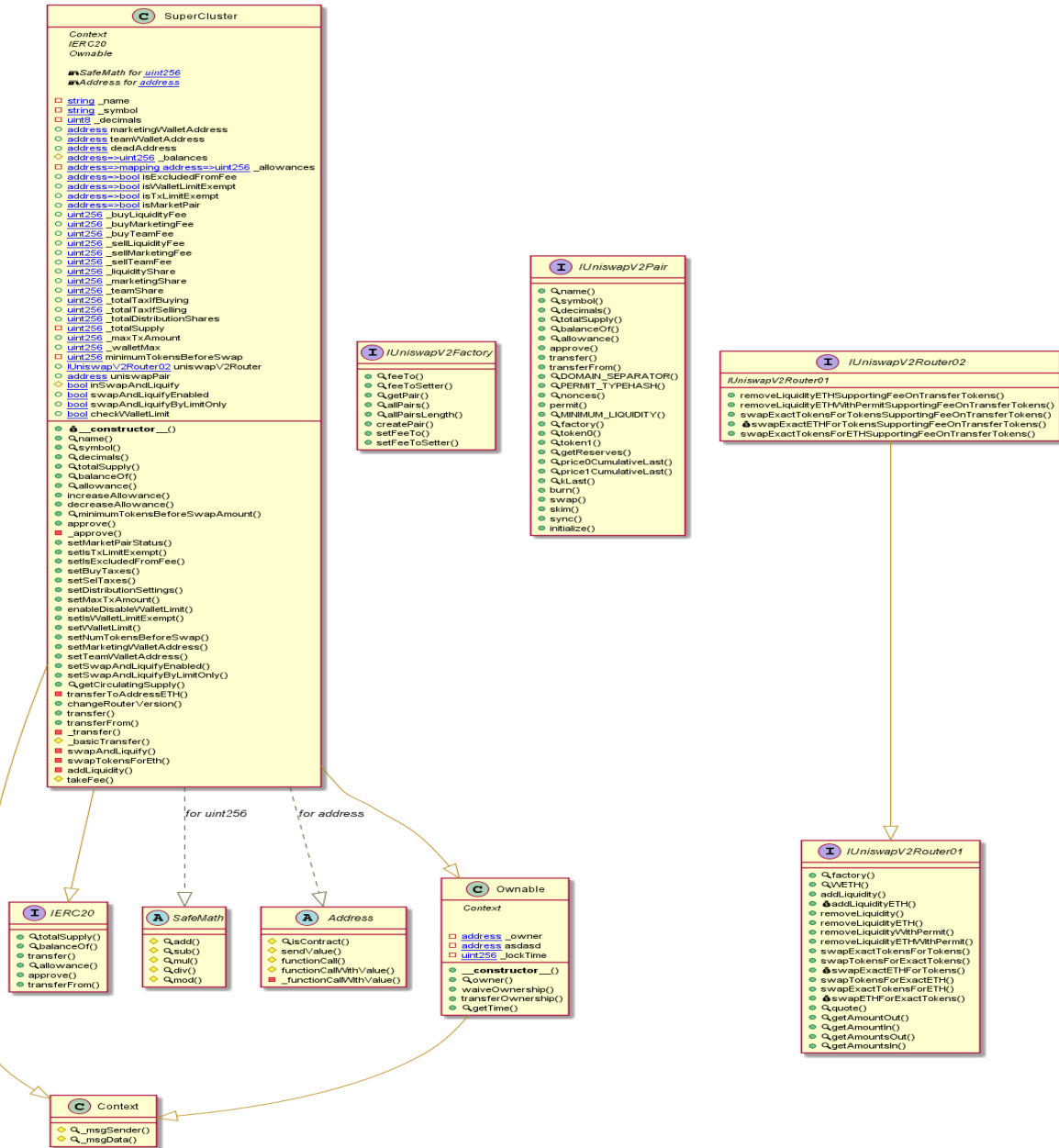
File: SuperCluster.sol
Contract: SuperCluster
Inherit: Context, IERC20, Ownable
Observation: Passed
Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	name	public	Passed	All Passed	No Issue	Passed
2	symbol	public	Passed	All Passed	No Issue	Passed
3	decimals	public	Passed	All Passed	No Issue	Passed
4	totalSupply	public	Passed	All Passed	No Issue	Passed
5	balanceOf	public	Passed	All Passed	No Issue	Passed
6	allowance	public	Passed	All Passed	No Issue	Passed
7	increaseAllowance	public	Passed	All Passed	No Issue	Passed
8	decreaseAllowance	public	Passed	All Passed	No Issue	Passed
9	minimumTokensBeforeSwapAmount	public	Passed	All Passed	No Issue	Passed
10	_approve	private	Passed	All Passed	No Issue	Passed

11	setMarketPair Status	public	Passed	All Passed	No Issue	Passed
12	setIsTxLimitEx empt	external	Passed	All Passed	No Issue	Passed
13	setIsExcluded FromFee	public	Passed	All Passed	No Issue	Passed
14	setBuyTaxes	external	Passed	All Passed	No Issue	Passed
15	setSelTaxes	external	Passed	All Passed	No Issue	Passed
16	setDistributio nSettings	external	Passed	All Passed	No Issue	Passed
17	setMaxTxAmo unt	external	Passed	All Passed	No Issue	Passed
18	enableDisable WalletLimit	external	Passed	All Passed	No Issue	Passed
19	setIsWalletLi mitExempt	external	Passed	All Passed	No Issue	Passed
20	setWalletLimi t	external	Passed	All Passed	No Issue	Passed
21	setNumToken sBeforeSwap	external	Passed	All Passed	No Issue	Passed
22	setMarketing WalletAddress	external	Passed	All Passed	No Issue	Passed
23	setTeamWalle tAddress	external	Passed	All Passed	No Issue	Passed
24	setSwapAndLi quifyEnabled	public	Passed	All Passed	No Issue	Passed
25	setSwapAndLi quifyByLimitO nly	public	Passed	All Passed	No Issue	Passed
26	getCirculating Supply	public	Passed	All Passed	No Issue	Passed
27	transferToAdd ressETH	private	Passed	All Passed	No Issue	Passed
28	changeRouter Version	public	Passed	All Passed	No Issue	Passed

29	transfer	public	Passed	All Passed	No Issue	Passed
30	transferFrom	public	Passed	All Passed	No Issue	Passed
31	_transfer	private	Passed	All Passed	No Issue	Passed
32	_basicTransfer	internal	Passed	All Passed	No Issue	Passed
33	swapAndLiquify	private	Passed	All Passed	No Issue	Passed
34	swapTokensForEth	private	Passed	All Passed	No Issue	Passed
35	addLiquidity	private	Passed	All Passed	No Issue	Passed
36	takeFee	internal	Passed	All Passed	No Issue	Passed

Code Flow Diagram - SuperCluster.sol



Code Flow Diagram - Slither Results Log

SuperCluster.sol

```
SuperCluster.allowance(address,address).owner (SuperCluster.sol#503) shadows:
- Ownable.owner() (SuperCluster.sol#159-161) (function)
SuperCluster._approve(address,address,uint256).owner (SuperCluster.sol#526) shadows:
- Ownable.owner() (SuperCluster.sol#159-161) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

SuperCluster.setBuyTaxes(uint256,uint256,uint256) (SuperCluster.sol#544-550) should emit an event for:
- _totalTaxIfBuying = buyLiquidityFee.add(buyMarketingFee).add(buyTeamFee) (SuperCluster.sol#549)
SuperCluster.setSellTaxes(uint256,uint256,uint256) (SuperCluster.sol#552-558) should emit an event for:
- _totalTaxIfSelling = sellLiquidityFee.add(sellMarketingFee).add(sellTeamFee) (SuperCluster.sol#557)
SuperCluster.setDistributionSettings(uint256,uint256,uint256) (SuperCluster.sol#560-566) should emit an event for:
- _liquidityShare = newLiquidityShare (SuperCluster.sol#561)
- _teamShare = newTeamShare (SuperCluster.sol#563)
- _totalDistributionShares = liquidityShare.add(marketingShare).add(teamShare) (SuperCluster.sol#565)
SuperCluster.setMaxTxAmount(uint256) (SuperCluster.sol#568-570) should emit an event for:
- _maxTxAmount = maxTxAmount (SuperCluster.sol#569)
SuperCluster.setWalletLimit(uint256) (SuperCluster.sol#580-582) should emit an event for:
- _walletMax = newLimit (SuperCluster.sol#581)
SuperCluster.setNumTokensBeforeSwap(uint256) (SuperCluster.sol#584-586) should emit an event for:
- _minimumTokensBeforeSwap = newLimit (SuperCluster.sol#585)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

SuperCluster.setMarketingWalletAddress(address).newAddress (SuperCluster.sol#588) lacks a zero-check on :
- marketingWalletAddress = address(newAddress) (SuperCluster.sol#589)
SuperCluster.setTeamWalletAddress(address).newAddress (SuperCluster.sol#592) lacks a zero-check on :
- teamWalletAddress = address(newAddress) (SuperCluster.sol#593)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in SuperCluster.swapAndLiquify(uint256) (SuperCluster.sol#691-713):
External calls:
- transferToAddressETH(marketingWalletAddress,amountBNBMarketing) (SuperCluster.sol#706)
  - recipient.transfer(amount) (SuperCluster.sol#610)
- transferToAddressETH(teamWalletAddress,amountBNBTeam) (SuperCluster.sol#709)
  - recipient.transfer(amount) (SuperCluster.sol#610)
External calls sending eth:
- transferToAddressETH(marketingWalletAddress,amountBNBMarketing) (SuperCluster.sol#706)
  - recipient.transfer(amount) (SuperCluster.sol#610)
- transferToAddressETH(teamWalletAddress,amountBNBTeam) (SuperCluster.sol#709)
  - recipient.transfer(amount) (SuperCluster.sol#610)
- addLiquidity(tokensForLP,amountBNBLiquidity) (SuperCluster.sol#712)
  - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (SuperCluster.sol#738-745)
State variables written after the call(s):
- addLiquidity(tokensForLP,amountBNBLiquidity) (SuperCluster.sol#712)
- _allowances[owner][spender] = amount (SuperCluster.sol#530)
Event emitted after the call(s):
- Approval(owner,spender,amount) (SuperCluster.sol#531)
  - addLiquidity(tokensForLP,amountBNBLiquidity) (SuperCluster.sol#712)
Reentrancy in SuperCluster.transferFrom(address,address,uint256) (SuperCluster.sol#640-644):
External calls:
- _transfer(sender,recipient,amount) (SuperCluster.sol#641)
  - recipient.transfer(amount) (SuperCluster.sol#610)
External calls sending eth:
- _transfer(sender,recipient,amount) (SuperCluster.sol#641)
  - recipient.transfer(amount) (SuperCluster.sol#610)
- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (SuperCluster.sol#738-745)
```

```

SuperCluster._totalSupply (SuperCluster.sol#419) is set pre-construction with a non-constant function or state variable:
- 100000000 * 10 ** decimals
SuperCluster._maxTxAmount (SuperCluster.sol#420) is set pre-construction with a non-constant function or state variable:
- 2000000 * 10 ** decimals
SuperCluster._walletMax (SuperCluster.sol#421) is set pre-construction with a non-constant function or state variable:
- 3000000 * 10 ** decimals
SuperCluster.minimumTokensBeforeSwap (SuperCluster.sol#422) is set pre-construction with a non-constant function or state variable:
- 1000 * 10 ** decimals
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state

Pragma version^0.8.4 (SuperCluster.sol#5) allows old versions
solc-0.8.4 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) (SuperCluster.sol#103-109):
- (success) = recipient.call{value: amount}() (SuperCluster.sol#107)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (SuperCluster.sol#126-143):
- (success,returndata) = target.call{value: weiValue}(data) (SuperCluster.sol#129)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

```

```

Function IUniswapV2Pair.DOMAIN_SEPARATOR() (SuperCluster.sol#214) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (SuperCluster.sol#215) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (SuperCluster.sol#231) is not in mixedCase
Function IUniswapV2Router01.WETH() (SuperCluster.sol#250) is not in mixedCase
Parameter SuperCluster.setSwapAndLiquifyEnabled(bool)._enabled (SuperCluster.sol#596) is not in mixedCase
Variable SuperCluster._balances (SuperCluster.sol#395) is not in mixedCase
Variable SuperCluster._buyLiquidityFee (SuperCluster.sol#403) is not in mixedCase
Variable SuperCluster._buyMarketingFee (SuperCluster.sol#404) is not in mixedCase
Variable SuperCluster._buyTeamFee (SuperCluster.sol#405) is not in mixedCase
Variable SuperCluster._sellLiquidityFee (SuperCluster.sol#407) is not in mixedCase
Variable SuperCluster._sellMarketingFee (SuperCluster.sol#408) is not in mixedCase
Variable SuperCluster._sellTeamFee (SuperCluster.sol#409) is not in mixedCase
Variable SuperCluster._liquidityShare (SuperCluster.sol#411) is not in mixedCase
Variable SuperCluster._marketingShare (SuperCluster.sol#412) is not in mixedCase
Variable SuperCluster._teamShare (SuperCluster.sol#413) is not in mixedCase
Variable SuperCluster._totalTaxIfBuying (SuperCluster.sol#415) is not in mixedCase
Variable SuperCluster._totalTaxIfSelling (SuperCluster.sol#416) is not in mixedCase
Variable SuperCluster._totalDistributionShares (SuperCluster.sol#417) is not in mixedCase
Variable SuperCluster._maxTxAmount (SuperCluster.sol#420) is not in mixedCase
Variable SuperCluster._walletMax (SuperCluster.sol#421) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (SuperCluster.sol#20)" inContext (SuperCluster.sol#13-23)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

```

```

- _allowances[owner][spender] = amount (SuperCluster.sol#530)
Event emitted after the call(s):
- Approval(owner,spender,amount) (SuperCluster.sol#531)
- approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,ERC20: transfer amount exceeds allowance)) (SuperCluster.sol#642)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4

Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (SuperCluster.sol#255) is too similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (SuperCluster.sol#256)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

SuperCluster.slitherConstructorVariables() (SuperCluster.sol#382-768) uses literals with too many digits:
- _totalSupply = 100000000 * 10 ** decimals (SuperCluster.sol#419)
SuperCluster.slitherConstructorVariables() (SuperCluster.sol#382-768) uses literals with too many digits:
- _maxTxAmount = 2000000 * 10 ** decimals (SuperCluster.sol#420)
SuperCluster.slitherConstructorVariables() (SuperCluster.sol#382-768) uses literals with too many digits:
- _walletMax = 3000000 * 10 ** decimals (SuperCluster.sol#421)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

SuperCluster._decimals (SuperCluster.sol#389) should be constant
SuperCluster._name (SuperCluster.sol#387) should be constant
SuperCluster._symbol (SuperCluster.sol#388) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

SuperCluster._totalSupply (SuperCluster.sol#419) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
SuperCluster.sol analyzed (10 contracts with 84 detectors), 72 result(s) found

```

Solidity Static Analysis

SuperCluster.sol

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: 121:4:

Check-effects-interaction:

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

[more](#)

Pos: 710:4:

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: 739:12:

Gas & Economy

Gas costs:

Gas requirement of function SuperCluster.transferFrom 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: 635:4:

ERC

ERC20:

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

[more](#)

Pos: 202:4:

Miscellaneous

No return:

IUniswapV2Router02.removeLiquidityETHWithPermitSupportingFeeOnTran
Defines a return type but never explicitly returns a value.

Pos: 345:4:

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: 670:16:

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:

Constant/View/Pure functions:

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

[more](#)

Pos: 368:4:

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 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 "**Well-Secured**".

Note For Contract Users

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

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

