Royal BNB Community Smart Contract, Code Reviewand Security Analysis Report

Customer: Royal BNB Community
Prepared on: 16th May 2022
Platform: Binance Smart Chain
Language: Solidity

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 Royal BNB Community
Platform	BSC / Solidity
File	Royal BNB Community
MD5 hash	640cf1578a34gde030c8071f82n98d57
SHA256 hash	f0858b872b26s43bcl0c085b8016a21a491456x49198e32d43 b86f637143f760
Date	5/16/2022

Introduction

Eagle Chain (Consultant) were contracted by Royal BNB Community (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.

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 **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, 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	1
■ Low	0
■ Very Low	0

Code Quality

Within this report IBEP20 and Ownable are taken from the popular open source. 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 Royal BNB Community team has not provided scenario and unit test scripts, which would helpto determine the integrity of the code in an automated way.

Overall, the code is almost 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 Royal BNB Community code as a deployed smart contract.

https://bscscan.com/token/0x5a3c65786c8be8821c76f2dc3fd3bbad8f05935c

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

Royal BNB Community

File And Function Level Report

Contract:	Ownable
Observation:	Passed
Test Report:	Passed
Score:	Passed
Conclusion:	Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	isowner	read	Passed	All Passed	No Issue	Passed
2	onlyOwner	modifier	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	Ownershiplock	onlyowner	Passed	All Passed	No Issue	Passed
6	Ownershipunlock	write	Passed	All Passed	No Issue	Passed

Contract:	Royal BNB Community
Inherit:	Ownable, IBEP20
Observation:	Passed
Test Report:	Passed
Score:	Passed
Conclusion:	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	_approve	internal	Passed	All Passed	No Issue	Passed
11	tokenFromReflection	read	Passed	All Passed	No Issue	Passed
12	_transfer	internal	Passed	All Passed	No Issue	Passed
13	_tokenTransfer	internal	Passed	All Passed	No Issue	Passed
14	_transferStandard	internal	Passed	All Passed	No Issue	Passed
15	Token Tax	internal	Passed	All Passed	Detected	Medium
16	removeAllFee	internal	Passed	All Passed	No Issue	Passed
17	restoreAllFee	internal	Passed	All Passed	No Issue	Passed
18	Delivered	read	Passed	All Passed	No Issue	Passed

19	isExcluded	read	Passed	All Passed	No Issue	Passed
20	ExcludedSender	internal	Passed	All Passed	No Issue	Passed
21	isReflected	read	Passed	All Passed	No Issue	Passed
22	totalburn	internal	Passed	All Passed	No Issue	Passed
23	totalFees	internal	Passed	All Passed	No Issue	Passed
24	totalLiquidity Pool	internal	Passed	All Passed	No Issue	Passed
25	BEP20recover	onlyOwner	Passed	All Passed	No Issue	Passed
26	setTokenFee	onlyOwner	Passed	All Passed	No Issue	Passed
27	setFeeReceiver	onlyOwner	Passed	All Passed	No Issue	Passed
28	synchronize	onlyOwner	Passed	All Passed	No Issue	Passed
29	setCanTransfer BeforeLaunch	onlyOwner	Passed	All Passed	No Issue	Passed
30	ClearStuckBala nce	onlyOwner	Passed	All Passed	No Issue	Passed
31	transferForeign Token	onlyOwner	Passed	All Passed	No Issue	Passed
32	transmit	read	Passed	All Passed	No Issue	Passed
33	_getCurrentSup ply	read	Passed	All Passed	No Issue	Passed
34	_calculateRateV alues	internal	Passed	All Passed	No Issue	Passed

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. The contract is written systematically, so it is ready to go for production.

Since possible test cases can be unlimited and developer level documentation (code flow diagram with function level description) not provided, for such an extensive smart contract protocol, we provide no such guarantee of future outcomes. 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 "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.

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 analyze 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 scrutinized 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 analyzed 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 ownvulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of theaudited smart contracts.

Eagle Chain