RD AUDITORS

KICHICOIN SMART CONTRACT, CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: KichiCoin
Prepared on: 24 June 2021
Platform: Binance Smart Chain
Language: Solidity

TABLE OF CONTENTS

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 DECISION OF CUSTOMER.

Document

Name	Smart Contract Code Review and Security Analysis Report of KichiCoin
Platform	BSC / Solidity
File 1	Kichii.sol
MD5 hash	4E39C4A8C7F503FB26F58AE90FA5333D
SHA256 hash	0D2DDF65229831DC09F33B646A97D684401A4D13360F349F8E9E1FC7BC6CEE23
Date	21/06/2021

Introduction

RD Auditors (Consultant) were contracted by KichiCoin (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 20 - 24 June 2021.

This contract consists of twelve files.

Project Scope

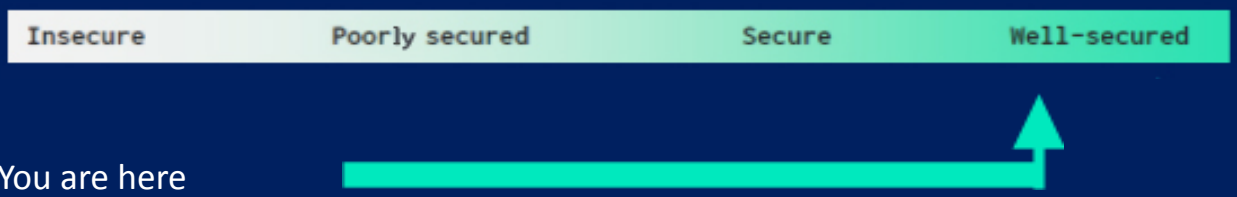
The scope of the project is a smart contract.

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.



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 0 critical, 0 high, 0 medium, 0 low and 0 very low level issues.

Code Quality

Please find a link that, within this report safeMath, IERC20, ownable 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 KichiCoin 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 not commented. Commenting can provide rich documentation for functions, return variables and more. Use of Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

We were given the KichiCoin contract as a github link:

https://testnet.bscscan.com/address/0xaD156cb49714113f414E660B028e65D02A782752#code

The hash of that file is mentioned in the table. As mentioned above, It's recommended to write comments in the smart contract code, so anyone can quickly understand the programming flow as well as complex code logic.

Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure. Those were based on well known industry standard open source projects and even core code blocks that are written well and systematically.

AS-IS Overview

KichiCoin

File And Function Level Report

File: Kichii.sol

Contract: KichiCoin
Import: Context, IBEP20, VRFConsumerBase
Inherit: Ownable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	transferOwnership	write	Passed	All Passed	No Issue	Passed
2	balanceOf	read	Passed	All Passed	No Issue	Passed
3	transfer	write	Passed	All Passed	No Issue	Passed
4	allowance	read	Passed	All Passed	No Issue	Passed
5	approveInternal	write	Passed	All Passed	No Issue	Passed
6	approve	read	Passed	All Passed	No Issue	Passed
7	transferFrom	write	Passed	All Passed	No Issue	Passed
8	increaseAllowance	write	Passed	All Passed	No Issue	Passed
9	decreaseAllowance	write	Passed	All Passed	No Issue	Passed
10	deliverReflectTokens	write	Passed	All Passed	No Issue	Passed
11	reflectionFromToken	read	Passed	All Passed	No Issue	Passed
12	tokenFromReflection	read	Passed	All Passed	No Issue	Passed
13	excludeFromReward	write	Passed	All Passed	No Issue	Passed
14	excludeBurnAddrFromReward	write	Passed	All Passed	No Issue	Passed
15	includeInReward	write	Passed	All Passed	No Issue	Passed
16	includeBurnAddrInReward	write	Passed	All Passed	No Issue	Passed
17	excludeFromFee	write	Passed	All Passed	No Issue	Passed
18	includeInFee	write	Passed	All Passed	No Issue	Passed

19	setTaxFeePercent	write	Passed	All Passed	No Issue	Passed
20	setCharityFeePercent	write	Passed	All Passed	No Issue	Passed
21	setBurnFeePercent	write	Passed	All Passed	No Issue	Passed
22	setLotteryFeePercent	write	Passed	All Passed	No Issue	Passed
23	setLiquidityFeePercent	write	Passed	All Passed	No Issue	Passed
24	setSwapAndLiquifyEnabled	write	Passed	All Passed	No Issue	Passed
25	takeReflectFee	write	Passed	All Passed	No Issue	Passed
26	getReflectRate	read	Passed	All Passed	No Issue	Passed
27	getCurrentSupplyTotals	read	Passed	All Passed	No Issue	Passed
28	takeLiquidityFee	write	Passed	All Passed	No Issue	Passed
29	takeCharityFee	write	Passed	All Passed	No Issue	Passed
30	takeBurnFee	write	Passed	All Passed	No Issue	Passed
31	takeLotteryFee	write	Passed	All Passed	No Issue	Passed
32	swapAndLiquify	write	Passed	All Passed	No Issue	Passed
33	swapTokensForEth	write	Passed	All Passed	No Issue	Passed
34	addLiquidity	write	Passed	All Passed	No Issue	Passed
35	removeAllFee	write	Passed	All Passed	No Issue	Passed
36	restoreAllFee	write	Passed	All Passed	No Issue	Passed
37	getTaxValues	write	Passed	All Passed	No Issue	Passed
38	getReflectionValues	read	Passed	All Passed	No Issue	Passed
39	getTaxAndReflectionValues	read	Passed	All Passed	No Issue	Passed
40	transferTokens	write	Passed	All Passed	No Issue	Passed
42	withdrawBNBSentToContractAddress	write	Passed	All Passed	No Issue	Passed
43	withdrawBEP20SentToContractAddress	write	Passed	All Passed	No Issue	Passed
44	rescueAllContractToken	write	Passed	All Passed	No Issue	Passed
45	setRouterAddress	write	Passed	All Passed	No Issue	Passed
46	weeklyLottery	write	Passed	All Passed	No Issue	Passed
47	lotteryDisperseFromDrawingWallet	write	Passed	All Passed	No Issue	Passed
48	lotteryDisperseFromDrawingWalletManual	write	Passed	All Passed	No Issue	Passed
49	transferTokensForLotteryToDrawingOrWinner	write	Passed	All Passed	No Issue	Passed
50	setMaxDrawingChances	write	Passed	All Passed	No Issue	Passed
51	setAmountNeededForDrawingChance	write	Passed	All Passed	No Issue	Passed
52	setPeriodsToDisperse	write	Passed	All Passed	No Issue	Passed
53	setHoursInPeriodToDisperse	write	Passed	All Passed	No Issue	Passed
54	setLotterySystemEnabled	write	Passed	All Passed	No Issue	Passed

55	setNumberOfTokensToSwapAndLiquify	write	Passed	All Passed	No Issue	Passed
56	excludeOrIncludeFromLottery	write	Passed	All Passed	No Issue	Passed
57	checkForLotteryParticipationOrRemoval	write	Passed	All Passed	No Issue	Passed
58	removeIndexFromLotteryArray	write	Passed	All Passed	No Issue	Passed
59	removeAddrFromLottoPoolCompletely	write	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.

Discussion

1. Loops may fail if the counter exceeds the limit.
2. Check for static values in the constructor.

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”

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.

RD
AUDITORS

Email: info@rdauditors.com

Website: www.rdauditors.com