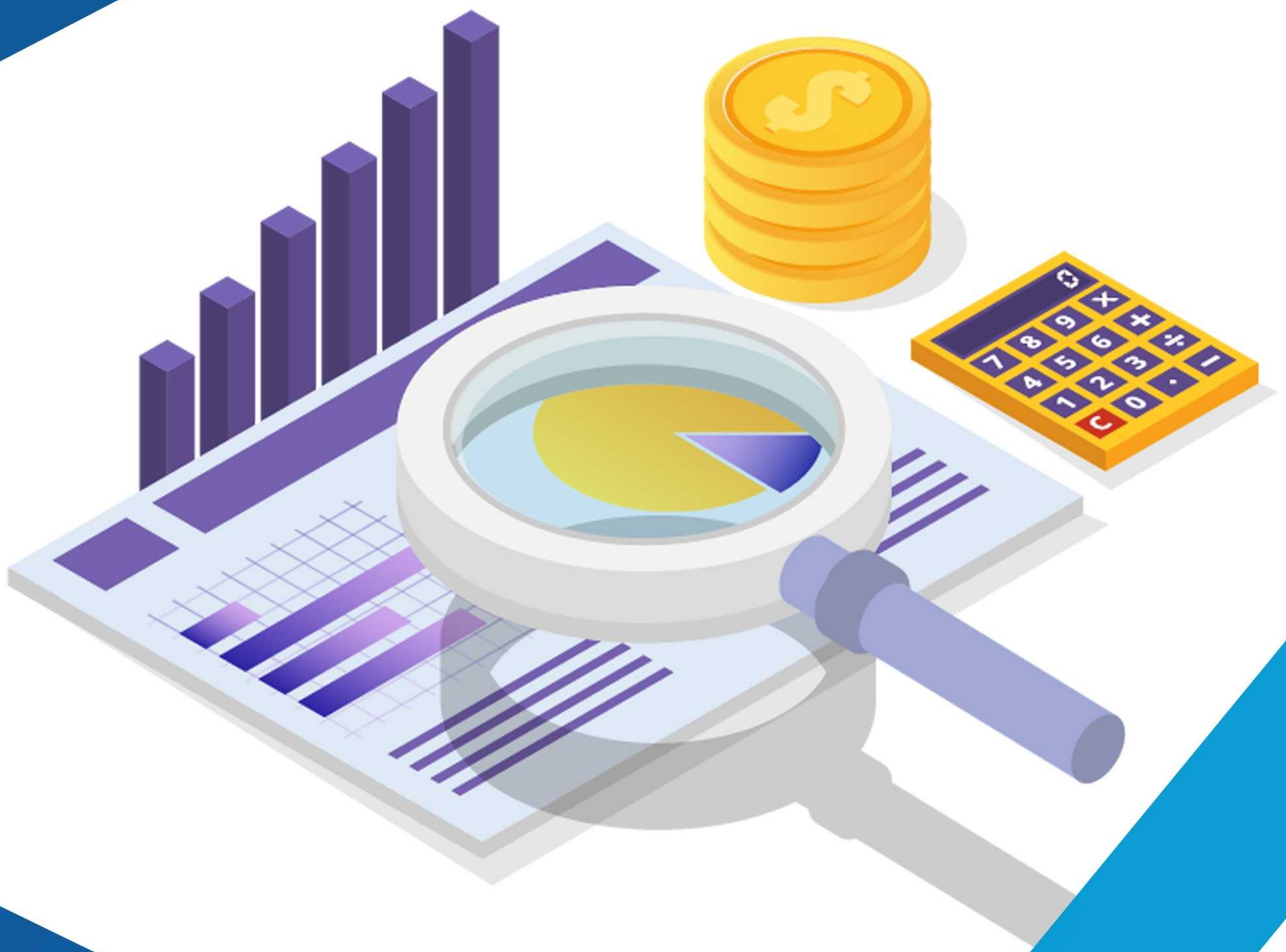
QuillAudits



Audit Report
September, 2021

K-ROOT

Contents

Scope of Audit

The scope of this audit was to analyze and document the K-Root Token smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.
SmartCheck.

Static Analysis
Static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step a series of automated tools are used to test security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of automated analysis were manually verified.

Gas Consumption
In this step we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

Every issue in this report has been assigned with a severity level. There are four levels of severity and each of them has been explained below.

High severity issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality and we recommend these issues to be fixed before moving to a live environment.

Medium level severity issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems and they should still be fixed.

Low level severity issues

Low level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity four issues which indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	1	0	0
Closed	0	0	1	2

Introduction

During the period of **August 25, 2021 to August 27, 2021** - QuillAudits Team performed a security audit for K-Root smart contracts.

The code for the audit was taken from the following official link: https://bscscan.com/address/0xe25d2b32749aa34708975ce949507ab723f71bbc#code

Revised code: https://bscscan.com/address/0x0cd3a8ce155a8d9daaf19e5caa642e72a2a24cd8#code

Issues Found – Code Review / Manual Testing

High severity issues

No issues were found.

Medium severity issues

1. User tokens can be burned by the owner

Line	Code

179	function burn(address from, uint amount) external {
	require(msg.sender == owner, "only admin");
	require(from != address(0), "No burn from zero address");
	_burn(from, amount);
	}

Description
The owner can burn any user address's tokens which do not give true freedom to users to use or hold their tokens.

Remediation
The owner should burn his tokens only, else it will be an issue of trust between the token holders or users.

Status: Acknowledged by the Auditee
Auditee Comments: "The burn feature is required as the token will exist on multiple chains."

Low level severity issues

2. Missing Check

Line	Code

179	function changeOwnership(address payable _newOwner) public onlyOwner {
	owner = _newOwner;
	}

Description
Missing Check for the new address which can create serious issues if wrong address passed.

Remediation
Check the new owner address before changing ownership of the contract.

Status: Fixed

Informational

3. Redundant code

Line	Code

165/179	function mint(address to, uint amount) external {
	require(msg.sender == owner, "only admin");
	require(to != address(0), "No mint to zero address");
	_mint(to, amount);
	}

Description
onlyOwner modifier can be used in both mint and burn functions to check owner address validations - to save gas.

Remediation
Use onlyOwner modifier for code optimization and code reusability.

Status: Fixed

4. Avoidable variable

Line	Code

129	uint256 accountBalance = balances[account];
	require(accountBalance >= amount, "BEP20: burn amount exceeds balance");
	balances[account] = accountBalance - amount;
	totalSupply -= amount;

Description
The use of extra variable can be avoided

Remediation
Directly use the balances[account] to reduce the extra variable.

Status: Fixed

Functional test

Function Names	Testing results
onlyOwner	Passed
changeOwnership	Passed
transfer	Passed
approve	Passed
transferFrom	Passed
allowance	Passed
_mint	Passed
_burn	Issue acknowledged by the auditee
_beforeTokenTransfer	Passed
mint	Passed
burn	Passed

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the K-Root platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the K-Root Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.

No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract, but the denial of service can impact the logic of the contract.

There are multiple issues found during the audit at various security levels that are highlighted in this report. Most of them are either fixed or acknowledged by K-Root team.

K-ROOT

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com