SMART CONTRACT CODE REVIEW
AND
SECURITY ANALYSIS REPORT

Date: 12 June, 2025

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.
The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for CAROU Token
Approved By	Svyatoslav Nadozirny | Solidity SC Auditor
Auditor company	Coders Valley Ltd. 63-66 Hatton Garden Fifth Floor, Suite 23 EC1N 8LE - London London (GB) United Kingdom
Type	BEP-20 Utility / DeFi Token
Platform	Binance Smart Chain (BSC)
Language	Solidity ^0.8.30
Methodology	Referenced document for audit methodology
ChangeLog	June 12, 2025 - initial release

Table of contents

Introduction

The Customer engaged OpenAudit Labs to evaluate the **CAROU** smart-contract for security, code quality and compliance with BEP-20 best practices. This report summarises our findings and provides actionable recommendations.

Scope

The scope of the project includes the following smart contracts from the file:

Contracts: https://drive.google.com/file/d/1gRcWKdJ94bJqMqYUAwwSbN2fPVj9iCS_/view

- **`BEP20.sol`** – `Implementation of BEP-20 standard token logic`
- **`Context.sol`** – `Provides execution context information`
- **`IBEP20.sol`** – `BEP-20 interface`
- **`Ownable.sol`** – `Basic access control mechanism`
- **`SafeMath.sol`** – `Arithmetic operations with overflow checks`
- **`CAROU.sol`** – `Main token contract that mints the fixed supply of 358 764 814 CAROU tokens`

Live Code: Not provided

Technical Documentation: Not provided

Tests: Not provided

Environment: Not provided

Additionally, the assessment reviews the token ABI and considers external documentation including the project whitepaper and investor dashboard details.

SHA256 Hash

SHA256 hash of the source code provided:

f2b8b1adf35af9f94a5aa8880286ec36433bc5a8fa3b927fd177d8dc8addb79d ….. CAROU.zip

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.

High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality.

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality Score is 8 out of 10.

- **Functional requirements** are provided in https://docs.google.com/presentation/d/141xnaz2ZUCpPiMOoHHESP5UH6S9IHCPDK2wK4q9YHDY/edit?slide=id.g3619f6016fd_0_3#slide=id.g3619f6016fd_0_3

  The token implements standard BEP-20 functions. (Score: 5/5).

- **Technical Requirements**: Technical requirements & environment details are partially provided, deployment and testing procedures are only briefly mentioned. (Score: 3/5).

- **NatSpec Adherence**: NatSpec comments are not used, which reduces readability for auditors and developers.

Code quality

The total Code Quality Score is 6 out of 10.

- **Development Environment:** The contract code follows a standard BEP-20 implementation with established libraries (SafeMath, Ownable, Context, BEP20 etc.). There is no detailed environment configuration provided with the code. (Score: 2/5).
- **Solidity Style Guide Compliance**: The code adheres to Solidity best practices with clear structure and consistent formatting. (Score: 4/5).

Security score

The security Score is 10 out of 10.

No critical, high, or medium severity issues were found. The contract correctly implements BEP-20 functionality, and while the usage of SafeMath is redundant in Solidity ^0.8.29 (due to built-in overflow protection), it does not compromise security. (Score: 10/10).

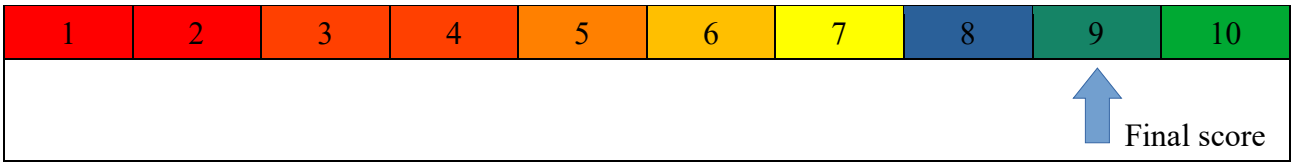
- **Critical Issues**: None

- **High Issues**: None
- **Medium Issues**: None
- **Low Issues**: 2. The use of SafeMath is redundant in Solidity ^0.8.30 due to built-in overflow checks; however, this does not impact security. Internal visibility of owner().

Summary

According to the assessment, the Customer's smart contract has the following score: **9.0**.

The system users should acknowledge all the risks summed up in the risks section of the report.



Breakdown:

- Documentation Quality: 8/10
- Code Quality: 6/10
- Security Level: 10/10
- Test Coverage: Not provided (requires unit tests for scoring).

Note: The final score is weighted according to the methodology (Documentation weighted at 1.0, Code Quality at 2.0, Security at 7.0), and the absence of unit tests impacts the overall score.

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
12 June, 2025	2	0	0	0

Risks

No significant risks or vulnerabilities were identified in the contract. The implementation strictly follows BEP-20 standards.

Risks include general operational risks inherent in blockchain projects and potential external attack vectors, which are not specific to the token contract.

System Overview

The CAROU token is a BEP-20 token deployed on the Binance Smart Chain with a fixed total supply of **358,764,814 CAROU**. It implements standard BEP-20 functionalities, such as token transfers, balance inquiries, and allowance mechanisms. The token is intended as a utility asset inside the CAROU ecosystem, all ecosystem logic lives off-chain or in separate contracts and is therefore out of audit scope.

Privileged roles

The CAROU token contract does not assign any privileged roles post-deployment. The minting operation occurs once during deployment in the constructor, and there are no functions available that allow the owner to alter token balances or mint additional tokens. This design reinforces decentralization and security.

Recommendations

To further enhance the quality and maintainability of the CAROU token contract, the following recommendations are made:

- **Gas optimisation – remove SafeMath.** Solidity ≥ 0.8.x has built-in overflow protection, omitting the library reduces byte-code size and gas usage.
- **NatSpec comments.** Add full NatSpec for every public/external function to improve maintainability and future auditability.
- **Automated test-suite.** Implement 100 % positive & negative coverage (Hardhat + Chai/Mocha). Include fuzz-tests for edge cases (e.g., max allowance, zero-address transfers).
- **Multisig / Time-lock ownership.** Transfer owner privileges to a multi-signature wallet or a 24-hour time-lock contract to mitigate single-key risk and provide transparency for governance actions.
- **Public owner() accessor.** Exposing the standard owner() view aids block-explorer and analytics tooling.
- **Continuous integration.** Integrate Solidity-static-analysis (Slither) and gas-reporting into the CI pipeline to catch issues before deployment.

While our examination found no critical security risks or vulnerabilities in the current contract, implementing these recommendations would enhance the contract's robustness, facilitate future updates, and ensure ongoing safe operation.

Checked Items

The contract was audited for commonly known and specific vulnerabilities. Here is a summary of the items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly.	Passed
Integer Overflow and Underflow	SWC-101	Solidity ^0.8.0 includes built-in overflow and underflow protection.	Not relevant
Outdated Compiler Version	SWC-102	Uses recent Solidity version ^0.8.28.	Passed
Floating Pragma	SWC-103	Contracts should deploy with a fixed compiler version.	Passed
Unchecked Call Return Value	SWC-104	Ensures the return value of calls is checked.	Passed
Access Control & Authorization	CWE-284	Properly implemented without unauthorized access to protected functions.	Passed
SELFDESTRUCT Instruction	SWC-106	Contract does not contain self-destruct functionality.	Not Relevant
Check-Effect-Interaction	SWC-107	Follows the pattern to prevent reentrancy attacks..	Passed
Assert Violation	SWC-110	Proper code execution prevents reaching a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	No deprecated functions are used.	Passed
Delegatecall to Untrusted Callee	SWC-112	No delegatecall usage to untrusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	No risks of DoS attacks through contract design.	Passed
Race Conditions	SWC-114	No race conditions or transaction order dependencies identified.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers are not used as time proxies.	Passed
Signature Unique Id	SWC-117 SWC-121	Not applicable, as the contract does not use message signatures..	Not Relevant

	SWC-122 EIP-155		
Shadowing State Variable	SWC-119	State variables are not shadowed.	Passed
Weak Sources of Randomness	SWC-120	Randomness is not generated using block attributes.	Not Relevant
Incorrect Inheritance Order	SWC-125	Inheritance order is carefully specified.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	External calls are only performed to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design. No unused variables found, ensuring efficient code.	Passed
EIP standards violation	EIP	The contract adheres to EIP standards, particularly ERC-20.	Passed
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Not Relevant
Gas Limit and Loops	**Custom**	Code is optimized to avoid high gas usage and unbounded loops.	Passed
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements	**Custom**	The code should be compliant with the	Passed

Compliance		requirements provided by the Customer.	
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Not Relevant
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Not Relevant
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

Findings

Critical

No issues

High

No issues

Medium

No issues

Low

2

- SafeMath redundant for Solidity ≥0.8,
- owner() declared internal, external tools expect public

Disclaimers

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with 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).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications.

Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

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 the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.