QuillAudits

Audit Report
November, 2022

For

Table of Content

Executive Summary 01

Checked Vulnerabilities 03

Techniques and Methods 04

Manual Testing 05

A. Contract - CommonBEP20.sol 05

High Severity Issues 05

Medium Severity Issues 05

Low Severity Issues 05

A.1 Ownership Transfer must be a two-step process. 05

Informational Issues 06

A.2 Unlocked pragma (pragma solidity ^0.7.0) 06

A.3 Absence of Adequate Code Comments 06

A.3 General Recommendation 07

Functional Testing 08

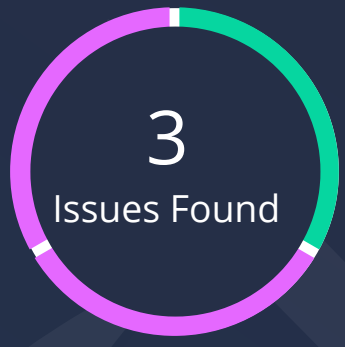
Automated Testing 08

Closing Summary 09

About QuillAudits 10

Executive Summary

Project Name	Realblock Report
Overview	Realblock token contract is a token created from the Binance Smart Chain token maker. The CommonBEP20.sol inherits a couple of BEP20 properties; BEP20Mintable, BEP20Burnable, BEP20Capped, and ServicePayer. These inherited properties aid in the mints, burns, and also limiting the total supply of the token. It also inherits the Ownable contract for management features and access control privileges.
Timeline	21 Nov,2022 - 24 November,2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse Realblock codebase for quality, security, and correctness. *https://bscscan.com/token/0x1068A279aEE90c40336604254066658f4474FE2b5*



High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	1	2
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	0

Types of Severities

High

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 be fixed before moving to a live environment.

Medium

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

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 issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

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 analysed 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.

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 the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the 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 the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - CommonBEP20.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

A.1 Ownership Transfer must be a Two-step Processes

Description

Contracts are integrated with the standard Openzeppelin ownable contract, however, when the owner mistakenly transfers ownership to an incorrect address, ownership is completely removed from the original owner and cannot be reverted. The transferOwnership() function in the ownable contract allows the current owner to transfer his privileges to another address. However, inside transferOwnership() , the newOwner is directly stored in the storage, owner, after validating the newOwner is a non-zero address, which may not be enough.

Remediation

It would be much safer if the transition is managed by implementing a two-step approach: _transferOwnership() and _updateOwnership() . Specifically, the _transferOwnership () function keeps the new address in the storage, _newOwner , instead of modifying the _owner() directly. The updateOwnership() function checks whether _newOwner is msg.sender, which means _newOwner signs the transaction and verifies himself as the new owner. After that, _newOwner could be set into _owner.

Status

Acknowledged

Informational Issues

A.2 Unlocked pragma (pragma solidity ^0.7.0)

Description

Contract has a floating solidity pragma version. This is present also in inherited contracts. Locking the pragma helps to ensure that the contract does not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively. The recent solidity pragma version also possesses its own unique bugs.

Remediation

Making the contract use a stable solidity pragma version prevents bugs occurrence that could be ushered in by prospective versions. It is recommended, therefore, to use a fixed solidity pragma version while deploying to avoid deployment with versions that could expose the contract to attack.

Status

Acknowledged

A.3 Absence of Adequate Code Comments

Description

CommonBEP20 inherits a ServicePayer abstract contract that interacts with the ServiceReceiver. This contract is characterized with some critical functions that have no adequate explanation that explains the motive of the functions. These code comments are necessary to help technical and non-technical people understand the business logic behind every function in the contracts. This establishes trust for the users.

Remediation

Provide adequate code comments for the ServiceReceiver/ServicePayer. Adopting the Natspec comment format is recommended due to how much clarity it gives to users.

Status

Acknowledged

A.4 General Recommendation

Privileges in contracts are important and for this reason, utmost caution must be taken when transferring ownership privileges among addresses. For precautionary reasons, we recommend a two-method transfer of ownership to avoid giving privileges to mistaken addresses. Also, code comments are necessary for the establishment of trust when users understand the business logic through the contract's comment. .

Functional Testing

CommonBEP20.sol

- ✓ Should get the name of the token
- ✓ should get the symbol of the token
- ✓ should get the decimals of the token
- ✓ should get the total supply of the token when deployed
- ✓ Should revert when the intended mint exceeds the token cap
- ✓ should get balance of the owner when contract is deployed
- ✓ should transfer tokens to other address
- ✓ should approve another account to spend token
- ✓ should burn the token by an holder
- ✓ Should burn and reduce approved token by calling the burnFrom function
- ✓ should revert when trying to burn beyond balance
- ✓ should mint to others address and increase total supply
- ✓ Should revert when non-owner tries to call the finishMinting

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the Realblock. We performed our audit according to the procedure described above.

Some issues of Low and Informational severity were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Realblock 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 Realblock Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

700+
Audits Completed

$15B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report
November, 2022

For

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com