QuillAudits

Audit Report
May, 2021

b-cube.ai

Contents

Introduction

During the period of **May 3rd, 2021 to May 13th, 2021** – QuillHash Team performed security audit for **BCUBE** smart contracts. The code for audit was taken from the following link:

https://github.com/b-cube-ai/b-cube-ico/blob/feat/public-sale/contracts/BCubePublicSale.sol
Commit Hash - 8600cd67a86fc951030528fe540952d92ff3e6be

https://github.com/b-cube-ai/b-cube-ico/blob/feat/public-sale/contracts/PublicSaleTreasury.sol
Commit Hash: e494ba8b72ee64d1a86f246c18f5d7a3e632f9b2

Updated Contracts:

https://github.com/b-cube-ai/b-cube-ico/blob/feat/public-sale/contracts/BCubePublicSale.sol
Commit Hash: d23fcd340783eb553be2d890d61529f4deda3ddb

https://github.com/b-cube-ai/b-cube-ico/blob/feat/public-sale/contracts/PublicSaleTreasury.sol
Commit Hash: 062b17c675762cecb7721023629417a1be4bfded

Overview of BCUBE

BCUBE is a secure, efficient, and easy-to-use blockchain-based platform providing solutions for the common pitfalls faced by crypto traders.
An all-in-one AI-driven Platform that offers:

- Trading on autopilot for CEX & DEX, cutting-edge DeFI solutions
- In-house strategies using AI/ML dynamically adapting to the market
- Marketplace of premium quality Crypto Trading Signals & Bots
- Build your own AI/ML bot on top of our already working strategies
- Educational courses, webinars, community & consultancy

BCUBE Features

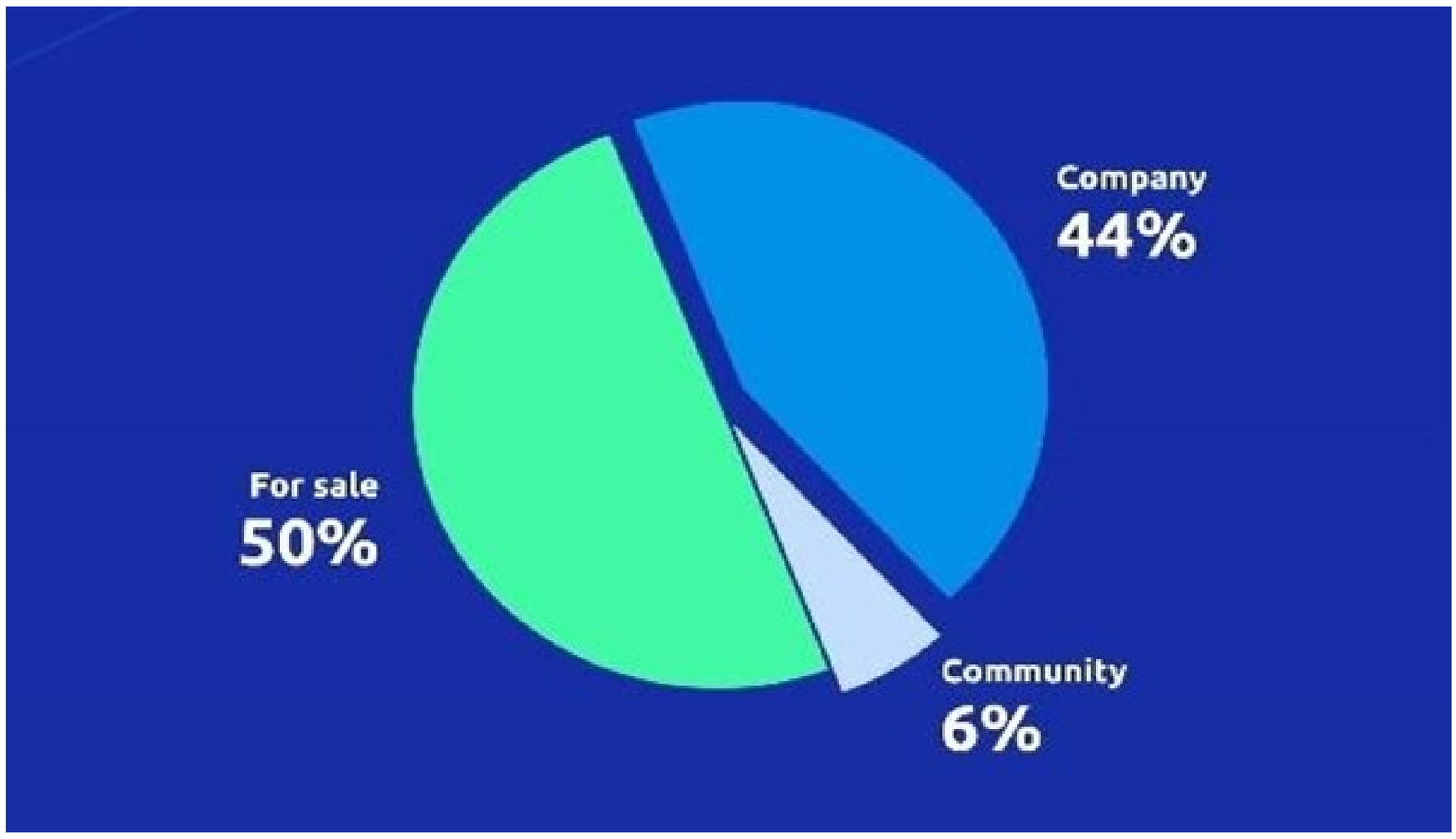
- Marketplace of best quality trading signals & bots
- AI-driven Quantitative Finance
- In-house institutional-grade execution engine
- In-house Sentiment Analysis engine
- Build your own AI/ML bots
- Non-custodial & regulated
- Profits-sharing Smart Contract
- High-quality educational contents
- Solutions for CeFI & DeFI
- Easy-to-use platform

BCUBE TOKEN SALE

The BCUBE token is the fuel of our ecosystem for gaining access to free trading signals, bots, education courses, and various other benefits by staking the required number of BCUBE tokens, get extra benefits by holding for certain periods of time and earn interests (APR) by participating in the Liquidity Pool. The APR is paid on hourly basis.

Start of sale
2021-05-14 13:30:00 UTC
End of sale
2021-05-28 13:30:00 UTC
Listing Time (starting of vesting period)
2021-05-30 12:00:00 UTC
Number of tokens for sale
25.000.000 BCUBE (50% max.)
Tokens exchange rate
1 BCUBE = $0.15 in Private Sale
1 BCUBE = $0.20 in Public Sale
Acceptable currencies
ETH, USDT
Minimal transaction amount
$500

Tokenomics



Token holders	Initial locking period after listing	Vesting rate
Clients Round, Pre-Seed & Seed participant	1 month	25 % / month
Private Sale participant	6.25% available at TGE	6.25 % / week
Public Sale participant	8.33% available at TGE	8.33 % / week
Team	6 months	12.5 % / 6 months
Advisor	6 months	25 % / 6 months
Development Fund	6 months	25 % / 6 months

Change in Tokenomics Limits

Private Sale: Min 500 USD, Max per transaction: 25k USD, Global max per wallet: 50k USD
Public Sale: Min 500 USD, Max per transaction: 25k USD, Global max per wallet: 50k USD

Details: https://ico.b-cube.ai/

Scope of Audit

The scope of this audit was to analyse **BCUBE** smart contract codebase for quality, security, and correctness. Following is the list of smart contracts included in the scope of this audit:

- BCubePublicSale.sol
- PublicSaleTreasury.sol

OUT-OF-SCOPE: External contracts, External Oracles, other smart contracts in the repository or imported smart contracts, economic attacks.

Check Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with (Unexpected) revert
- 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
- Unsafe type inference
- Implicit visibility level
- Address hardcoded
- Using delete for arrays
- Integer overflow/underflow
- Locked money
- Private modifier
- Revert/require functions
- Using var
- Visibility
- Using blockhash
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of BCUBE smart contracts 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, Manticore, Slither.

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

	High	Medium	Low	Informational
Reported	0	0	9	4
Open	0	0	3	4
Closed	0	0	6	0

Issues Found – Code Review / Manual Testing

BCubePublicSale Contract

High severity issues

None.

Medium severity issues

None.

Low severity issues

1. **Consider using the latest stable solidity version**

   The smart contract is using an old version of solidity. It is highly recommended to use the latest stable version of solidity.
   Code Lines: 1

Auditors Remarks: Not Fixed [Won't be fixed]

2. **Coding Style Issues**

   Coding style issues influence code readability and, in some cases, may lead to bugs in future. Smart Contracts have a naming convention, indentation and code layout issues. It's recommended to use Solidity Style Guide to fix all the issues. Consider following the Solidity guidelines on formatting the code and commenting for all the files. It can improve the overall code quality and readability.

```
BCubePublicSale.sol
   21:2  error  Line length must be no more than 120 but current length is 122  max-line-length
  301:2  error  Line length must be no more than 120 but current length is 133  max-line-length
  305:2  error  Line length must be no more than 120 but current length is 166  max-line-length
  307:2  error  Line length must be no more than 120 but current length is 164  max-line-length
  315:2  error  Line length must be no more than 120 but current length is 146  max-line-length
  320:2  error  Line length must be no more than 120 but current length is 144  max-line-length

✖ 6 problems (6 errors, 0 warnings)
```

Auditors Remarks: Fixed

3. Order of layout

The order of functions as well as the rest of the code layout does not follow the solidity style guide.
Layout contract elements in the following order:

- Pragma statements
- Import statements
- Interfaces
- Libraries
- Contracts

Inside each contract, library or interface, use the following order:

- Type declarations
- State variables
- Events
- Functions

Please read following documentation links to understand the correct order:
https://solidity.readthedocs.io/en/v0.5.17/style-guide.html#order-of-layout

Auditors Remarks: Fixed.

4. Use external function modifier instead of public

The public functions that are never called by contract should be declared external to save gas. Following functions can be declared external:
- setAdmin(address) [#112-116]

Auditors Remarks: Not Fixed [Won't be fixed]

5. Reentrancy

One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the

calling function wasn't expecting. It's recommended that the calls to external functions/events should happen after any changes to state variables in your contract so your contract is not vulnerable to a reentrancy exploit. When control is transferred to recipient, care must be taken to not create reentrancy vulnerabilities. OpenZeppelin has its own mutex implementation you can use called ReentrancyGuard. This library provides a modifier you can apply to any function called nonReentrant that guards the function with a mutex.
Consider using ReentrancyGuard or the checks-effects-interactions pattern.

Following link explains more about Reentrancy and ReentrancyGuard
https://docs.openzeppelin.com/contracts/2.x/api/utils#ReentrancyGuard
https://blog.openzeppelin.com/reentrancy-after-istanbul/

Most Recent DeFi Reentrancy Attack: DFORCE

Code Lines:
- buyBcubeUsingUSDT(uint256) [#249-262]
- buyBcubeUsingETH() [#231-245]

Event is emitted after an external function call.

Auditors Remarks: Fixed.

Informational severity issues

1. Use of block.timestamp should be avoided

Do not use block.timestamp, now or blockhash as a source of randomness. Malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. Contracts often need access to the current timestamp to trigger time-dependent events. As Ethereum is decentralized, nodes can synchronize time only to some degree. It is risky to use block.timestamp, as block.timestamp can be manipulated by
miners. Therefore block.timestamp is recommended to be supplemented with some other strategy in the case of high-value/risk applications.

Remediations:

- https://consensys.github.io/smart-contract-best-practices/recommendations/#timestamp-dependence
- https://consensys.github.io/smart-contract-best-practices/recommendations/#avoid-using-blocknumber-as-a-timestamp

Code Lines: 125, 157

Auditors Remarks: Not Fixed [Won't be fixed]

7. Approve function of ERC-20 is vulnerable

A security issue called "Multiple Withdrawal Attack" - originates from two methods in the ERC20 standard for approving and transferring tokens. The use of these functions in an adverse environment (e.g., front-running) could result in more tokens being spent than what was intended. This issue is still open on the GitHub and several solutions have been made to mitigate it.
For more details on how to resolve the multiple withdrawal attack check the following document for details:

- https://drive.google.com/file/d/1skR4BpZ0VBSQICIC_eqRBgGnACf2li5X/view?usp=sharing
- https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit
- https://blog.smartdec.net/erc20-approve-issue-in-simple-words-a41aaf47bca6
- https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729

Auditors Remarks: Not Fixed [Won't be fixed]

PublicSaleTreasury Contract

High severity issues

None.

Medium severity issues

None.

Low severity issues

1. **Consider using the latest stable solidity version**

   The smart contract is using an old version of solidity. It is highly recommended to use the latest stable version of solidity.
   Code Lines: 2

   Auditors Remarks: Not Fixed [Won’t be fixed]

2. **Coding Style Issues**

   Coding style issues influence code readability and, in some cases, may lead to bugs in future. Smart Contracts have a naming convention, indentation and code layout issues. It’s recommended to use Solidity Style Guide to fix all the issues. Consider following the Solidity guidelines on formatting the code and commenting for all the files. It can improve the overall code quality and readability.

   Auditors Remarks: Fixed

3. **Order of layout**

   The order of functions as well as the rest of the code layout does not follow solidity style guide.
   Layout contract elements in the following order:

   - Pragma statements
   - Import statements
   - Interfaces
   - Libraries
   - Contracts

Inside each contract, library or interface, use the following order:

- Type declarations
- State variables
- Events
- Functions

Please read following documentation links to understand the correct order:
https://solidity.readthedocs.io/en/v0.5.17/style-guide.html#order-of-layout

Auditors Remarks: Fixed.

4. Reentrancy

One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the calling function wasn't expecting. It's recommended that the calls to external functions/events should happen after any changes to state variables in your contract so your contract is not vulnerable to a reentrancy exploit. When control is transferred to recipient, care must be taken to not create reentrancy vulnerabilities. OpenZeppelin has its own mutex implementation you can use called ReentrancyGuard. This library provides a modifier you can apply to any function called nonReentrant that guards the function with a mutex.
Consider using ReentrancyGuard or the checks-effects-interactions pattern.

Following link explains more about Reentrancy and ReentrancyGuard
https://docs.openzeppelin.com/contracts/2.x/api/utils#ReentrancyGuard
https://blog.openzeppelin.com/reentrancy-after-istanbul/

Most Recent DeFi Reentrancy Attack: DFORCE

Code Lines:
- shareWithdraw(uint256) [#135-161]
  Event is emitted after an external function call.

Auditors Remarks: Fixed.

Informational severity issues

1. **BokkyPooBahsDateTimeLibrary can be used instead of multiple if .. else if statements**

   To make contract more gas efficient BokkyPooBahsDateTimeLibrary can be used instead of multiple if .. else if statements. The diffDays() returns number of days between two timestamps.
   Details: BokkyPooBahsDateTimeContract.sol
   Code Lines: 80-130

   Auditors Remarks: Not Fixed [Won't be fixed]

2. **Approve function of ERC-20 is vulnerable**

   A security issue called "Multiple Withdrawal Attack" - originates from two methods in the ERC20 standard for approving and transferring tokens. The use of these functions in an adverse environment (e.g., front-running) could result in more tokens being spent than what was intended. This issue is still open on the GitHub and several solutions have been made to mitigate it.
   For more details on how to resolve the multiple withdrawal attack check the following document for details:

   - **https://drive.google.com/file/d/1skR4BpZ0VBSQICIC_eqRBgGnACf2li5X/view?usp=sharing**
   - **https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit**
   - **https://blog.smartdec.net/erc20-approve-issue-in-simple-words-a41aaf47bca6**
   - **https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729**

   Auditors Remarks: Not Fixed [Won't be fixed]

Automated Testing

Slither

Slither is an open-source Solidity static analysis framework. This tool provides rich information about Ethereum smart contracts and has the critical properties. It runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses.

BCubePublicSale.sol:

```
INFO:Printers:
Compiled with solc
Number of lines: 1272 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 3
Number of contracts: 15 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 10
Number of informational issues: 31
Number of low issues: 4
Number of medium issues: 1
Number of high issues: 0

ERCs: ERC20

+-------------------------+-------------+-------+----------------------+--------------+---------------------+
|          Name           | # functions | ERCS  |      ERC20 info      | Complex code |      Features       |
+-------------------------+-------------+-------+----------------------+--------------+---------------------+
| AggregatorV3Interface   |      5      |       |                      |      No      |                     |
|          Roles          |      3      |       |                      |      No      |                     |
|     SignedSafeMath      |      5      |       |                      |      No      |                     |
|        Pausable         |      13     |       |                      |      No      |                     |
|        SafeMath         |      8      |       |                      |      No      |                     |
|         IERC20          |      6      | ERC20 |      No Minting      |      No      |                     |
|                         |             |       | Approve Race Cond.   |              |                     |
|                         |             |       |                      |              |                     |
|       SafeERC20         |      6      |       |                      |      No      | Tokens interaction  |
|        Address          |      3      |       |                      |      No      |      Send ETH       |
|                         |             |       |                      |              |      Assembly       |
|        SafeCast         |      5      |       |                      |      No      |                     |
|    BCubePublicSale      |      37     |       |                      |      No      |     Receive ETH     |
|                         |             |       |                      |              |      Send ETH       |
+-------------------------+-------------+-------+----------------------+--------------+---------------------+
INFO:Slither:BCubePublicSale02.sol analyzed (15 contracts)
```

PublicSaleTreasury.sol:

```
INFO:Printers:
Compiled with solc
Number of lines: 1352 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 3
Number of contracts: 15 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 8
Number of informational issues: 30
Number of low issues: 7
Number of medium issues: 12
Number of high issues: 0

ERCs: ERC20

+-------------------------+-------------+-------+----------------------+--------------+---------------------+
|          Name           | # functions | ERCS  |      ERC20 info      | Complex code |      Features       |
+-------------------------+-------------+-------+----------------------+--------------+---------------------+
| AggregatorV3Interface   |      5      |       |                      |      No      |                     |
|          Roles          |      3      |       |                      |      No      |                     |
|     SignedSafeMath      |      5      |       |                      |      No      |                     |
|        Pausable         |      13     |       |                      |      No      |                     |
|        SafeMath         |      8      |       |                      |      No      |                     |
|         IERC20          |      6      | ERC20 |     No Minting       |      No      |                     |
|                         |             |       | Approve Race Cond.   |              |                     |
|                         |             |       |                      |              |                     |
|       SafeERC20         |      6      |       |                      |      No      | Tokens interaction  |
|        Address          |      3      |       |                      |      No      |      Send ETH       |
|                         |             |       |                      |              |      Assembly       |
|        SafeCast         |      5      |       |                      |      No      |                     |
|   PublicSaleTreasury    |      39     |       |                      |     Yes      |     Receive ETH     |
|                         |             |       |                      |              |      Send ETH       |
+-------------------------+-------------+-------+----------------------+--------------+---------------------+
INFO:Slither:PublicSaleTreasury.sol analyzed (15 contracts)
```

Slither didn't raise any critical issue with smart contracts. The smart contracts were well tested and all the minor issues that were raised have been documented in the report. Also, all other vulnerabilities of importance have already been covered in the Findings and Tech Details section of the report.

Mythril

Mythril is a security analysis tool for EVM bytecode. It detects security vulnerabilities in smart contracts built for Ethereum. It uses symbolic execution, SMT solving and taint analysis to detect a variety of security vulnerabilities.

Mythril did not detect any high severity issue. All the considerable issues raised by Mythril are already covered in the **Findings and Tech Details** section of this report.

Manticore

Manticore is a symbolic execution tool for analysis of smart contracts and binaries. During Symbolic Execution / EVM bytecode security assessment did not detect any high severity issue. All the considerable issues are already covered in the **Findings and Tech Details** of this report.

Closing Summary

Overall, the smart contract code is extremely well documented, follows a high-quality software development standard, contains many utilities and automation scripts to support continuous deployment / testing / integration, and does NOT contain any obvious exploitation vectors that QuillHash was able to leverage within the timeframe of testing allotted.
Overall, the smart contracts adhered to **ERC20** guidelines. No critical or major vulnerabilities were found in the audit. Several issues of **low severity were found and reported during the audit.**
The outcome of this security audit is satisfactory; due to time and resource constraints, only testing and verification of essential properties were performed to achieve objectives and deliverables set in the scope. QuillHash recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts.

No critical or major vulnerabilities were found in the reaudit of BCubePublicSale.sol. The new logic for token limits has been implemented correctly. The test cases cover all scenarios.

Disclaimer

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

b-cube.ai

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com