QuillAudits

AUDIT REPORT

September 2025

For

CoinQuant

Table of Content

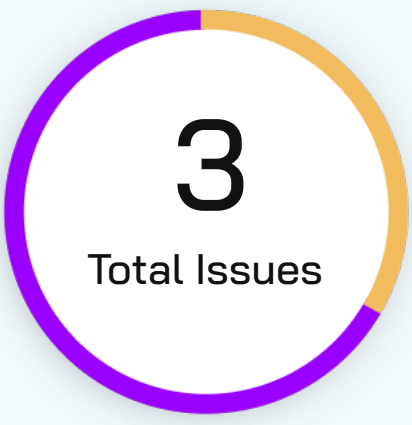
Executive Summary

Project Name	CQX Token
Protocol Type	ERC20 Token
Project URL	https://www.coinquant.ai/
Overview	CQXToken is a ERC-20 token implementation for "CoinQuant X" (CQX) that combines OpenZeppelin's standard token extensions for enhanced functionality. The contract mints a fixed supply of 500 million tokens to a designated recipient address upon deployment, positioning it as a presale or initial distribution token with no ongoing minting capabilities.
Audit Scope	The scope of this Audit was to analyze the CoinQuant X Token Smart Contract for quality, security, and correctness.
Source Code link	https://sepolia.etherscan.io/token/0xb40baf5546bef9ab04713119e59669781a0e5f76#code
Contracts in Scope	CQX Token
Language	Solidity
Blockchain	Ethereum
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	8th September 2025

Verify the Authenticity of Report on QuillAudits Leaderboard:

https://www.quillaudits.com/leaderboard

Number of Issues per Severity



Critical	0 (0.0%)
High	0 (0.0%)
Medium	0 (0.0%)
Low	1 (33.3%)
Informational	3 (66.6%)

Severity

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

Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Missing Zero Address Validation	Low	Acknowledged
2	Code Readability Enhancement	Informational	Acknowledged
3	Use of Floating Solidity Pragma Version	Informational	Acknowledged

Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Computations Correctness
- Race conditions/front running
- SWC Registry
- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC's conformance
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls

- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Multiple Sends
- Using suicide
- Using delegatecall
- Upgradeable safety
- Using throw
- Using inline assembly
- Style guide violation
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

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

- The overall quality of code
- Use of best practices
- Code documentation and comments, match logic and expected behavior
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper
- Implementation of ERC 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.

Static Analysis

A 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 analyzed, 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 behavior 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, F oundry, Solhint, Mythril, Slither, Solidity Static Analysis.

Types of Severity

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

Critical: Immediate and Catastrophic Impact

Critical issues are the ones that an attacker could exploit with relative ease, potentially leading to an immediate and complete loss of user funds, a total takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

High (H): Significant Risk of Major Loss or Compromise

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

Medium (M): Potential for Moderate Harm Under Specific Circumstances

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

Low (L): Minor Imperfections with Limited Repercussions

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

Informational (I): Opportunities for Improvement, Not Immediate Risks

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.

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.

Severity Matrix

Impact

Likelihood	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

Low Severity Issues

Missing Zero Address Validation

Acknowledged

Path

CQX Token

Description

The constructor lacks validation to ensure the recipient parameter is not the zero address.

The constructor accepts a recipient parameter to determine where the initial token supply should be minted, but it does not validate that this address is not the zero address (0x0000000000000000000000000000000000000000).

In Ethereum, sending tokens to the zero address is equivalent to burning them permanently, as no one controls the private keys to that address.

Impact

If the zero address is accidentally passed as the recipient during deployment, the entire initial supply of 100 million tokens would be minted to address(0), effectively burning them and making them permanently inaccessible. This would result in a total loss of the initial token supply.

Recommendation

Add zero address validation in the constructor

Informational Issues

Code Readability Enhancement

Acknowledged

Path

CQX Token

Description

The expression 100_000_000 * 10 ** decimals() could be more readable. The current implementation uses 100_000_000 * 10 ** decimals() to calculate the initial token supply. While this is functionally correct, it can be improved for better code readability and maintainability. Clear, readable code reduces the likelihood of errors during development and makes the contract easier to audit and understand.

Recommendation

Consider using a more explicit and readable format

```
uint256 constant INITIAL_SUPPLY = 100_000_000; // 100M tokens
_mint(recipient, INITIAL_SUPPLY * 10 ** decimals());
```

Use of Floating Solidity Pragma Version

Acknowledged

Path

CQX Token

Description

The contract currently uses a floating Solidity pragma (^0.8.0). This allows compilation with any compiler version >=0.8.0 <0.9.0. Such flexibility can introduce risks:

- Future compiler releases within 0.8.x may introduce behavior changes, optimizer differences, or stricter checks that could alter contract bytecode or functionality.
- OpenZeppelin dependencies may have been tested against specific compiler versions; compiling with an unsupported version could cause unexpected issues.
- Audit reproducibility is affected, as the contract may compile to different bytecode across environments.

Recommendation

Pin the Solidity compiler to a fixed, stable version that matches the OpenZeppelin release used in the project (e.g., pragma solidity 0.8.20;). This ensures deterministic compilation, maintains consistency across testing, deployment, and verification, and aligns with industry best practices.

Functional Tests

Some of the tests performed are mentioned below:

- ✔ should return name
- ✔ should return symbol
- ✔ should return decimals 18
- ✔ should have correct total supply
- ✔ should assign the entire initial supply to recipient
- ✔ should transfer tokens between accounts
- ✔ should fail when transferring more than balance
- ✔ should approve and allow transferFrom
- ✔ should increase and decrease allowance

Trust & assumptions / Notes to the user

- **Centralization risk — single-address control (high):** Because the entire supply was minted to one recipient address at deployment, that address controls the entire circulating supply until tokens are distributed. If that address is a team, exchange, or a private wallet, they can move or sell tokens at any time. Treat tokens with a single initial holder as high-risk for sudden dumps. Verify who controls the recipient.

- **Permit (gasless approvals):** permit is present — convenient UX,but always confirm front-end uses it safely (signing UX is still off-chain/UX risk).

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 CQX Token. We performed our audit according to the procedure described above.

No critical issues in CQX token, just 1 low and 2 issues of Informational severity were found.CQX Token team acknowledged all the issues mentioned.

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With seven years of expertise, we've secured over 1400 projects globally, averting over $3 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

7+ Years of Expertise	**1M+** Lines of Code Audited
50+ Chains Supported	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

September 2025

For

CoinQuant

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com