QuillAudits

# Audit Report
# April, 2024

For

Covalent

# Table of Content

# Executive Summary

| | |
|---|---|
| **Project Name** | Covalent |
| **Overview** | The Covalent team plans to migrate from the current CQT token to their new CXT token on a 1:1 basis. The CQT token has an inflation mechanism where an account with **EMISSION_ROLE** is able to mint a number of new tokens depending on the time elapsed between the current time and lastMint.<br><br>There are a number of privileged users in the system granted privileges by roles:<br><br>- protocolCouncil<br>- emissionManager<br>- emergencyCouncil |
| **Timeline** | 27th March, 2024 - 4th April, 2024 |
| **Updated Code Received** | 9th April, 2024 |
| **Second Review** | 9th April, 2024 - 11th April, 2024 |
| **Method** | Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives. |
| **Audit Scope** | The scope of this audit was to analyse the Covalent codebase for quality, security, and correctness. |
| **Source Code** | *https://github.com/covalenthq/covalent-x-token/tree/main/src* |
| **Contracts In-Scope** | - CovalentMigration.sol<br>- CovalentXToken.sol<br>- DefaultEmissionManager.sol |
| **Branch** | **main** |

| | |
|---|---|
| **Contracts Out of Scope** | In-scope contracts have been audited by QuillAudits. However, these contracts inherit functionality from out-of-scope Smart contracts that were not audited. Vulnerabilities in unaudited contracts could impact in-scope Smart Contracts functionality. QuillAudits is not responsible for such vulnerabilities.<br><br>Below are Out of Scope Contracts:<br>- PowUtil.sol<br>- OpenZeppelin contracts: AccessControlUpgradeable, ERC20Permit, Ownable2StepUpgradeable, SafeERC20, etc. |
| **Fixed In** | *https://github.com/covalenthq/covalent-x-token* |
| **Commit** | 1f1e9af, 3c3db28 |

# Number of Security Issues per Severity


| | High | Medium | Low | Informational |
|---|---|---|---|---|
| Open Issues | 0 | 0 | 0 | 0 |
| Acknowledged Issues | 0 | 0 | 0 | 1 |
| Partially Resolved Issues | 0 | 0 | 0 | 0 |
| Resolved Issues | 0 | 1 | 0 | 1 |

# Checked Vulnerabilities

- 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
- 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 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's 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

Manual Review, Slither, Foundry.

## Types of Severity

Every issue in this report has been assigned to 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 be fixed before moving to a live environment.

### Medium 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 Severity Issues

Low-level severity issues can cause minor impact and 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 four 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.

## High Severity Issues

No issues were found.

## Medium Severity Issues

### 1. Higher token emission than the expected 5% inflation rate

**Path**

CovalentXToken.sol

**Function**

**#L17**: mintPerSecondCap

**Description**

After contract deployment 1 billion **CXT** tokens get minted to account 1:1 to the previous supply of the **CQT** token with an added twist of inflation to mint more tokens over time to the treasury address. If the mint rate is not taken into proper account, it could lead to hyperinflation with more tokens getting minted than the market can accommodate within that price range. The expected inflation rate is currently 5% compounded logarithmically yearly and cannot be updated, but the **mintPerSecondCap** can be adjusted by the **CAP_MANAGER_ROLE** address to reduce token emission (and subsequently inflation) to **0**.

To achieve a 5% yearly inflation rate, a token emission of **~2.3e18 CXT/second** is advisable but currently the contract has an emission rate of **10e18 CXT/second**.  The specifications document states that:

- Given the compounded rate, we've set **mintPerSecondCap = 2.5e18** (or 2.5 tokens per second) as a conservative cap.

**Recommendation**

Adjust the **mintPerSecondCap** variable to **2.5e18** to reduce risk of inflation. Also, ensure that changes effected with accounts that have the **CAP_MANAGER_ROLE** come from governance decisions to prevent a compromised user from gaming the protocol for personal gain.

**Status**

**Resolved**

## Low Severity Issues

No issues were found.

## Informational Issues / Recommendations

### 2. Unlocked pragma version

**Path**

CovalentXToken.sol

**Description**

It is best practice to select a specific solidity pragma version when deploying code to the blockchain to avoid compiler issues/breaking changes that could arise from newer versions being used.

**Recommendation**

Lock the solidity version.

**Status**

**Acknowledged**

### 3. Minting and transfer duplication

**Path**

DefaultEmissionManager.sol

**Function**

#L65: **mint()**

**Description**

The mint function of the DefaultEmissionManager contract after it performs the necessary supply checks will calculate the **amountToMint** variable to be minted to the DefaultEmissionManager contract. Immediately it mints those tokens they get transferred to the Treasury address as seen on #L65. This transfer is unnecessary because there's no further computation (token splitting between multiple addresses) required before the tokens go to the treasury. This incurs some extra costs on gas for calling this function than is required for the same execution without a transfer.

```
function mint() external {

    ICovalentXToken _token = token;

    _token.mint(address(this), amountToMint);

 ++ _token.mint(treasury, amountToMint);

    // @audit why mint to emissionManager and then transfer to treasury?

    // network development -- for rewards to node developers

 -- _token.safeTransfer(treasury, amountToMint);

}
```

**Recommendation**

Mint the tokens directly to the treasury address instead to optimize gas costs.

**Status**

**Resolved**

# Functional Tests Cases

- Should transfer tokens to other users
- Should transfer tokens using Permit2 functionality
- Should mint tokens approximately within the inflation range

# 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 Covalent codebase. We performed our audit according to the procedure described above.

Some issues of Medium, and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture. In The End, Covalent team,resolved one medium issue and one informational issue and acknowledged one remaining Information Issue.

# Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Covalent smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Covalent smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Covalent to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

# 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.

**1000+**
Audits Completed

**$30B**
Secured

**1M**
Lines of Code Audited

## Follow Our Journey

# Audit Report
# April, 2024

For

Covalent

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com