Quantstamp

Covalent X-Token

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	ERC20 Token
Timeline	2024-04-08 through 2024-04-12
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	spec.md
Source Code	• https://github.com/covalenthq/covalent-x-token #1f1e9af
Auditors	• Danny Aksenov Senior Auditing Engineer • Jennifer Wu Auditing Engineer • Nikita Belenkov Auditing Engineer

Documentation quality	Low
Test quality	Low
Total Findings	7 Fixed: 3 Acknowledged: 4
High severity findings	0
Medium severity findings	0
Low severity findings	2 Fixed: 1 Acknowledged: 1
Undetermined severity findings	1 Acknowledged: 1
Informational findings	4 Fixed: 2 Acknowledged: 2

Summary of Findings

The Covalent X Token (CXT) audit encompasses a suite of smart contracts designed for the management, migration, and emission of the Covalent Network Token on the Ethereum blockchain.

The audit revealed no critical security threats within these contracts. Most findings are informational, highlighting design and behavioral nuances of the reviewed contracts that could be relevant to the Covalent team. Furthermore, we recommend enhancing code coverage to bolster confidence in the code before launching into production.

Update: The Covalent team either acknowledged or addressed all issues highlighted in the report. We would still recommend adding additional code coverage to achieve higher production readiness.

ID	DESCRIPTION	SEVERITY	STATUS
COV-1	Unnecessary Token Approval	● Low	Acknowledged
COV-2	Missing Input Validation	● Low	Fixed
COV-3	Production Readiness Can Be Improved	● Informational	Fixed
COV-4	Limited Flexibility in Adjusting Interest Calculation Due to Constant Usage	● Informational	Acknowledged
COV-5	Upgrade Openzeppelin Dependencies	● Informational	Fixed
COV-6	Minting Allowance Gets Reset if Not Fully Used	● Informational	Acknowledged
COV-7	`permit2Enabled` Can Be Bypassed with Manual Approvals	● Undetermined	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer
Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
   1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

Repo: https://github.com/covalenthq/covalent-x-token(1f1e9afa221cb754e2fd26c7773be779418006b3) Files: src/CovalentMigration.sol src/CovalentXToken.sol src/DefaultEmissionManager.sol

Findings

COV-1 Unnecessary Token Approval

● Low ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client.
The client provided the following explanation:

Our contract will mint additional tokens in the future (inflationary supply or rewards), and these tokens need to be handled or distributed by the same or a different contract that was initially used, maintaining an approval mechanism is needed.

File(s) affected: `src/DefaultEmissionManager.sol`

Description: The `DefaultEmissionManager.initialize()` function includes a call to `token.safeApprove()` for the migration contract, which is designated to distribute the initial CXT token supply. However, this step is unnecessary because the `CovalentXToken.constructor()` already mints the initial token supply directly to the migration contract. The `safeApprove()` call is redundant and can be removed from the `initialize()` function to streamline the contract's setup process.

Recommendation: Remove the line `token.safeApprove(address(migration), type(uint256).max);` if it is not required for the intended functionality of the contract. If the approval is needed, provide a clear explanation of its purpose and ensure that it aligns with the contract's design and security considerations.

COV-2 Missing Input Validation

• Low ⓘ Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `89025dca19300daadcb50dc279685deece3d5d5e`.
The client provided the following explanation:

Added a check for "Set token _cxt should be validated to be a non-zero address"

File(s) affected: `src/lib/PowUtil.sol`, `src/CovalentMigration.sol`

Related Issue(s): SWC-123

Description: It is crucial to validate inputs, even if the inputs come from trusted addresses, to avoid human error. A lack of robust input validation can only increase the likelihood and impact in the event of mistakes.

Following is the list of places that can potentially benefit from stricter input validation:

- `PowUtil.sol`:
  - `exp2()`: The `x` value should be validated before proceeding with the shifting operation. The value of `x` should not exceed the limit of a `uint192` type, ensuring it remains within 192 bits to prevent data loss when shifting by 64 bits.
- `CovalentMigration.sol`:
  - `setToken()`: `_cxt` should be validated to be a non-zero address.

Recommendation: We recommend adding the relevant checks.

COV-3 Production Readiness Can Be Improved

• Informational ⓘ Fixed

Update

The Covalent team has addressed a majority of the concerns raised in this issue, opting to not use a package manager for the `PowUtil` library.

Update

Marked as "Fixed" by the client.
Addressed in: `0932bd630d2576ecf592da86455b7513a938c793`, `3c3db2841fcd77415c42413feabe483f9b8a8881`.
The client provided the following explanation:

1. We removed console2.log where it's not necessary
2. We have added testcases for permit2 (commit hash - 3c3db2841fcd77415c42413feabe483f9b8a8881)
3. Acknowledged
4. We removed the todo comments as they are already done and reviewed it to streamline the code

File(s) affected: `src/CovalentXToken.sol`, `src/lib/PowUtil.sol`, `src/DefaultEmissionManager.sol`, `src/CovalentMigration.sol`

Description: Several aspects of the codebase need improvement to better prepare the project for production. In particular,

1. There are many areas in the codebase (e.g. the use of `console2.log`) that are only intended for testing and debugging. This greatly hinders the readability of the codebase and introduces unnecessary complexities.
2. The project can greatly benefit from additional test cases, given the amount of moving parts in the project. It may be helpful to have some integration tests to test the interaction between different steps in a more complex workflow.
3. The clone-and-own approach involves copying and adjusting open-source code at one's own discretion. From the development perspective, it is initially beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries. Rather than the clone-and-own approach, good industry practice is to use a package manager (e.g., npm) for handling library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as using libraries. If the file is cloned anyway, a comment including the repository, commit hash of the version cloned, and the summary of modifications (if any) should be added. This helps to improve the traceability of the file. These libraries include, but not limited to:
   1. `src/lib/PowUtil.sol`
4. The smart contract contains some `todo` comments. It is unclear whether this needs to be addressed or not.

Recommendation: To streamline your transition to production, consider addressing the points mentioned in the description.

COV-4 Limited Flexibility in Adjusting Interest Calculation Due to Constant Usage

● Informational ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

This approach is expected behaviour as per our design and needs, we want to minimize the changes from POL token contracts as they are thoroughly audited already.

File(s) affected: `src/DefaultEmissionManager.sol`

Description: Based on the client's tests in `ImplChange.t.sol` and the implementation of `DefaultEmissionManager.sol`, the client intends to change `INTEREST_PER_YEAR_LOG2` by upgrading the implementation contract of `DefaultEmissionManager.sol` whenever they need to modify `INTEREST_PER_YEAR_LOG2`, which is a constant. Constants are stored within the implementation contracts. This approach introduces unnecessary complexity because it relies on contract upgrades to change a constant, making the system less flexible.

Recommendation: If the `INTEREST_PER_YEAR_LOG2` value is intended to be mutable, we recommend changing this constant to a variable. Furthermore, a method should be established to allow modification of this variable, with access restricted to the contract owner. This approach would simplify the process of adjusting `INTEREST_PER_YEAR_LOG2` without the need to deploy a new implementation contract.

COV-5 Upgrade Openzeppelin Dependencies

● Informational ⓘ Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `41959f95b7f1fc84d8338aec1ffc72182997096a`.
The client provided the following explanation:

Changed the Openzepplin module to v4.9.6

File(s) affected: `All contracts in scope`

Description: The project currently uses OpenZeppelin contracts version `4.9.2`. Version 4.9.6 is available and is the latest safe version, offering improvements and bug fixes.

Recommendation: Update OpenZeppelin dependencies to version `4.9.6` to maintain code security and reliability.

COV-6 Minting Allowance Gets Reset if Not Fully Used

● Informational ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

Emission_role is strictly be given to Governance/trusted address only, and Emission_role can't mint desired amount of tokens, mint function will mint maxMint (calculated inflatory supply) directly.

File(s) affected: `src/DefaultEmissionManager.sol`

Description: Anyone with an `EMISSION_ROLE` can call the `mint()` function and mint a certain amount of tokens to a destination. There is a `mintPerSecondCap` that applies to the amount that is allowed to be minted. This is calculated based on the time difference between the last `mint()` action and the current timestamp. A user with `EMISSION_ROLE` can mint a number of tokens that is below the `maxMint`, and that would actually reset the allowance and not preserve it, as `mint()` action has just occurred.

Recommendation: Consider if this is an issue and potentially see if the whole allowance has been used up.

COV-7 `permit2Enabled` Can Be Bypassed with Manual Approvals

● Undetermined ⓘ Acknowledged

Update

> Marked as "Acknowledged" by the client.
> The client provided the following explanation:
>
> The permit2 contract has full approval by default. If the approval is revoked, it can still be manually approved by user.

File(s) affected: `/src/CovalentXToken.sol`

Description: `CovalentXToken` can enable full allowance to `PERMIT2` when `permit2Enabled` is set to true. It can also be disabled if `permit2Enabled` is set to false. It is unclear if `permit2Enabled==false` would mean that no `PERMIT2` allowance should be allowed, as currently, the user can manually call `increaseAllowance()` to `PERMIT2`.

Recommendation: Consider if this is expected behavior.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Files

- `85b...381 ./src/CovalentMigration.sol`
- `66c...6c3 ./src/DefaultEmissionManager.sol`
- `946...8ef ./src/CovalentXToken.sol`
- `7c0...9c6 ./src/interfaces/ICovalentXToken.sol`
- `237...71f ./src/interfaces/IDefaultEmissionManager.sol`
- `f97...eca ./src/interfaces/ICovalentMigration.sol`
- `8bc...46a ./src/lib/PowUtil.sol`

Tests

- `1fe...bdc ./test/CovalentXToken.t.sol`
- `650...5cc ./test/SigUtils.t.sol`
- `8eb...9c5 ./test/DefaultEmissionManager.t.sol`
- `95b...307 ./test/CovalentMigration.t.sol`
- `4ee...18f ./test/ImplChange.t.sol`
- `553...63c ./test/util/calc.js`

Automated Analysis

N/A

Test Suite Results

All tests are passing.

```
Running 1 test for test/ImplChange.t.sol:ChangeImplementation
[PASS] testChangeImplementation() (gas: 5627797)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 7.96ms

Running 1 test for test/CovalentMigration.t.sol:CovalentMigrationTest
[PASS] test_Deployment() (gas: 119471)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 22.16s

Running 9 tests for test/DefaultEmissionManager.t.sol:DefaultEmissionManagerTest
[PASS] testRevert_Initialize() (gas: 18464)
[PASS] test_Deployment() (gas: 39402)
[PASS] test_ImplementationCannotBeInitialized() (gas: 24380)
[PASS] test_InflatedSupplyAfter(uint256) (runs: 256, μ: 82158, ~: 82186)
[PASS] test_InvalidDeployment() (gas: 1803660)
[PASS] test_Mint() (gas: 30134)
[PASS] test_MintDelay(uint128) (runs: 256, μ: 134270, ~: 137409)
[PASS] test_MintDelayAfterNCycles(uint128,uint8) (runs: 256, μ: 178156, ~: 3570)
[PASS] test_MintDelayTwice(uint128) (runs: 256, μ: 163563, ~: 163618)
Test result: ok. 9 passed; 0 failed; 0 skipped; finished in 22.16s




Running 11 tests for test/CovalentXToken.t.sol:CovalentXTokenTest
[PASS] testRevert_Mint(address,address,uint256) (runs: 256, μ: 37852, ~: 37852)
[PASS] testRevert_Permit2Revoke(address) (runs: 256, μ: 40335, ~: 40335)
[PASS] testRevert_UpdateMintCap(uint256,address) (runs: 256, μ: 38446, ~: 38446)
[PASS] test_Deployment(address) (runs: 256, μ: 85423, ~: 85423)
[PASS] test_InvalidDeployment() (gas: 436965)
[PASS] test_Mint(address,uint256) (runs: 256, μ: 52718, ~: 54403)
[PASS] test_MintMaxExceeded(address,uint256,uint256) (runs: 256, μ: 53978, ~: 56773)
[PASS] test_Permit2Transfer() (gas: 96409)
[PASS] test_RevertExtraAmountPermit2Transfer() (gas: 86472)
[PASS] test_RevertPermit2DuplicateSigTransfer() (gas: 123366)
[PASS] test_RevokePermit2Allowance(address) (runs: 256, μ: 25043, ~: 25043)
Test result: ok. 11 passed; 0 failed; 0 skipped; finished in 312.26s

Ran 4 test suites: 22 tests passed, 0 failed, 0 skipped (22 total tests)
```

Code Coverage

We would like to see improved branch coverage, at least 90% or higher, in `src/CovalentMigration.sol` and `src/DefaultEmissionManager.sol`.

Coverage

File	% Lines	% Statements	% Branches	% Funcs
src/CovalentMigration.sol	100.00% (**6**/6)	100.00% (**8**/8)	50.00% (**2**/4)	100.00% (**3**/3)
src/CovalentXToken.sol	76.92% (**10**/13)	84.21% (**16**/19)	100.00% (**4**/4)	83.33% (**5**/6)
src/DefaultEmissionManager.sol	94.44% (**17**/18)	96.55% (**28**/29)	87.50% (**7**/8)	75.00% (**3**/4)
src/lib/PowUtil.sol	100.00% (**132**/132)	100.00% (**212**/212)	100.00% (**144**/144)	100.00% (**1**/1)

Changelog

- 2024-04-12 - Initial report
- 2024-05-10 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Quantstamp

Covalent X-Token