Quantstamp

Powerloom

Executive Summary

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

Type	ERC20
Timeline	2023-12-06 through 2023-12-15
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Technical Reference
Source Code	• https://github.com/PowerLoom/token-contract #2570df7
Auditors	• Julio Aguilar Auditing Engineer • Mostafa Yassin Auditing Engineer

Documentation quality	Medium
Test quality	High
Total Findings	4 Fixed: 4
High severity findings	0
Medium severity findings	0
Low severity findings	0
Undetermined severity findings	0
Informational findings	4 Fixed: 4

Summary of Findings

The contract `PowerloomToken` was created using OpenZeppelin's Wizard as a standard ERC-20 token with additional features. It inherits from OpenZeppelin's `ERC20Permit` for permit-based approvals, and `ERC20Votes` for voting delegation. The token's supply is determined at deployment with 1 billion tokens minted to the deploying address. Notably, tokens cannot be minted nor burned and the contract does not include any custom business logic or specific functionality beyond the standard ERC-20 and related features provided by the OpenZeppelin contracts. We have not found any significant security vulnerabilities. However, we still recommend addressing all the issues pointed out in the report.

2023-12-15 Update: The Powerloom team fixed all the issues in the report.

ID	DESCRIPTION	SEVERITY	STATUS
POW-1	**Missing Development Framework**	• Informational	Fixed
POW-2	**Centralized Design**	• Informational	Fixed
POW-3	**Unlocked Pragma**	• Informational	Fixed
POW-4	**Max Supply Inconsistency**	• Informational	Fixed

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/PowerLoom/token-contract(2570df7be7a85bf85ad95570e9a61fd53d0554cb) Files: PowerloomToken.sol

Files Excluded

Repo: https://github.com/PowerLoom/token-contract(2570df7be7a85bf85ad95570e9a61fd53d0554cb) Files: README.md LICENSE

Findings

POW-1 Missing Development Framework

● Informational ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `e6b7472d0983118cb5827e19dd7db3519b6bf2a5`. The client provided the following explanation:

> Added hardhat development framework to the project. Along with tests for ERC20, ERC20Votes, and ERC20Permit. You can use `npm test` to run the tests.

File(s) affected: `PowerloomToken.sol`

Description: Using development frameworks is essential in any kind of software endeavor. They often come with built-in testing and deployment tools. Automated testing is crucial for ensuring the reliability and security of smart contracts. Additionally, they also facilitate adding new features and cooperation. The client provided a single isolated token contract. There is no development framework or tests.

Recommendation: We recommend adding such a framework and tests.

POW-2 Centralized Design

● Informational ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `7326411bc1b60398dcb3b33c05930b3e7eb91e5b`. The client provided the following explanation:

> Instead of minting tokens to the account deploying the contract, we mint to a multi-sig address. This address is controlled by the team and will be used to set things up initially.

File(s) affected: `PowerLoomToken.sol`

Description: The current token contract mints the total supply to the deployer's address. Only the deployer can distribute the minted tokens. Additionally, if this is an EOA, it could open the possibility of a compromised deployer (i.e. stolen keys) sending tokens to unexpected addresses.

Recommendation: Consider having this address be a multi-sig to prevent such attacks.

POW-3 Unlocked Pragma

● Informational ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `856ef93b20ddc24daa02e005333bc1ce9d7cbe5f`. The client provided the following explanation:

> Removed the caret from the pragma. It is now locked to version 0.8.20.

File(s) affected: `PowerloomToken.sol`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format pragma solidity (^)0.8.*. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version and above, hence the term "unlocked". Using unlocked pragmas is discouraged due to compatibility risks with potential breaking changes in newer compiler versions. Specifically, locking the pragma to a fixed version enhances security by ensuring the code benefits from the latest security enhancements and bug fixes. Additionally, specifying a fixed compiler version promotes code reproducibility, reducing variations in compiled bytecode for greater consistency across environments and deployments.

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend removing the caret to lock the file onto a specific Solidity version.

POW-4 Max Supply Inconsistency

● Informational ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `5f06d8ba2746d3e17305e57487f9e566a14d558f`. The client provided the following explanation:

> We are now overrriding the `_maxSupply` function to set max Supply to 1 billion tokens.

File(s) affected: `PowerloomToken.sol`

Description: The max token supply is defined at deployment and set to 1 billion tokens. One of the inherited contracts, ERC20Votes, defines the internal function _maxSupply() which returns type(uint208).max. However, the PowerloomToken contract does not override this function to provide the real maximum supply.

Recommendation: We do not see a security risk with this issue. We would just like to point out a way to make the token implementation more understandable and consistent.

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).

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:
- Slither v0.10.0

Steps taken to run the tools:
1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

All the Slither results were either identified as false positives or included in the findings of this report.

Test Suite Results

```
ERC20
  PowerloomToken
    ✔ total supply: returns the total token value
    ✔ has a name
    ✔ has a symbol
    ✔ has 18 decimals
    balanceOf
      ✔ returns zero when the requested account has no tokens
      ✔ returns the total token value when the requested account has some tokens
    transfer
      ✔ reverts when the recipient is the zero address (40ms)
      when the recipient is not the zero address
        ✔ reverts when the sender does not have enough balance
        when the sender transfers all balance
          ✔ transfers the requested value
          ✔ emits a transfer event
        when the sender transfers zero tokens
          ✔ transfers the requested value
          ✔ emits a transfer event
    transfer from
      ✔ reverts when the token owner is the zero address
      when the token owner is not the zero address
        ✔ reverts when the recipient is the zero address
        when the recipient is not the zero address
          when the spender has enough allowance
            ✔ reverts when the token owner does not have enough balance
            when the token owner has enough balance
              ✔ transfers the requested value
              ✔ decreases the spender allowance
              ✔ emits a transfer event
              ✔ does not emit an approval event
          when the spender does not have enough allowance
            ✔ reverts when the token owner has enough balance
            ✔ reverts when the token owner does not have enough balance
          when the spender has unlimited allowance
            ✔ does not decrease the spender allowance
            ✔ does not emit an approval event
    approve
      ✔ reverts when the spender is the zero address
      when the spender is not the zero address
        when the sender has enough balance
          ✔ emits an approval event
          ✔ approves the requested value when there was no approved value before
          ✔ approves the requested value and replaces the previous one when the spender had an approved value
        when the sender does not have enough balance
          ✔ emits an approval event
          ✔ approves the requested value when there was no approved value before
          ✔ approves the requested value and replaces the previous one when the spender had an approved value

Contract: ERC20Permit
  ✔ initial nonce is 0
  ✔ domain separator
  permit
    ✔ accepts owner signature
    ✔ rejects reused signature
    ✔ rejects other signature
    ✔ rejects expired permit

Contract: ERC20Votes
  vote with blocknumber
    ✔ initial nonce is 0
    ✔ recent checkpoints
    should implement ERC6372
      ✔ clock is correct
      ✔ CLOCK_MODE is correct
    run votes workflow
```

```
      ✔ initial nonce is 0
      delegation with signature
        ✔ delegation without tokens
        ✔ delegation with tokens
        ✔ delegation update (44ms)
        with signature
          ✔ accept signed delegation
          ✔ rejects reused signature
          ✔ rejects bad delegatee
          ✔ rejects bad nonce
          ✔ rejects expired permit
      getPastTotalSupply
        ✔ reverts if block number >= current block
        ✔ returns 0 if there are no checkpoints
      Compound test suite
        getPastVotes
          ✔ reverts if block number >= current block
          ✔ returns 0 if there are no checkpoints
          ✔ returns the latest block if >= last checkpoint block
          ✔ returns zero if < first checkpoint block
    set delegation
      call
        ✔ delegation with balance
        ✔ delegation without balance
      with signature
        ✔ accept signed delegation
        ✔ rejects reused signature
        ✔ rejects bad delegatee
        ✔ rejects bad nonce
        ✔ rejects expired permit
    change delegation
      ✔ call
    transfers
      ✔ no delegation
      ✔ sender delegation
      ✔ receiver delegation
      ✔ full delegation
    Compound test suite
      balanceOf
        ✔ grants to initial account
      numCheckpoints
        ✔ returns the number of checkpoints for a delegate (60ms)
        ✔ does not add more than one checkpoint in a block (1048ms)
      getPastVotes
        ✔ reverts if block number >= current block
        ✔ returns 0 if there are no checkpoints
        ✔ returns the latest block if >= last checkpoint block
        ✔ returns zero if < first checkpoint block
        ✔ generally returns the voting balance at the appropriate checkpoint (56ms)


  75 passing (3s)
```

Code Coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
PowerloomToken.sol	100	100	100	100	
All files	100	100	100	100	

Changelog

- 2023-12-06 - Initial report
- 2023-12-15 - 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