QuillAudits

AUDIT REPORT

October 2025

For

CHICAGO

Table of Content

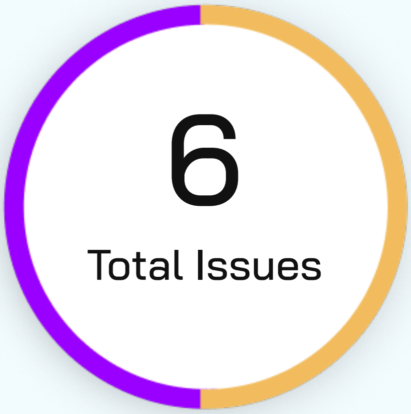
Executive Summary

Project Name	Chicago Coin
Protocol Type	E-Sports
Project URL	https://chicagocoin.io
Overview	ChicagoCoin is ERC20 token contract. It implements basic token functionality including transfers, approvals, and allowances, with an ownership mechanism inherited from OpenZeppelin's Ownable pattern. The contract mints a fixed supply of 1 billion tokens (with 18 decimals) to a specific address upon deployment. It includes standard features like balance checking, token burning, and allowance management (increase/decrease).
Audit Scope	The scope of this Audit was to analyze the Chicago Coin Smart Contracts for quality, security, and correctness.
Source Code Link	https://etherscan.io/token/0xae1e1b4d8f590371b77bee27257ef038d4b835a1#code
Language	Solidity
Blockchain	Ethereum
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	24th October 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	3 (50.0%)
🟪	Informational	3 (50.0%)

Severity

Issues

	Critical	High	Medium	Low	Informational
Open	0	0	0	0	0
Acknowledged	0	0	0	**3**	**3**
Partially Resolved	0	0	0	0	0
Resolved	0	0	0	0	0

Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Two-Step Ownership Transfer Missing	Low	Acknowledged
2	Incorrect Operation Order in transferFrom()	Low	Acknowledged
3	Redundant Constant Supply	Low	Acknowledged
4	Hardcoded Pre-Mint Recipient Address	Informational	Acknowledged
5	Unused Ownable Inheritance	Informational	Acknowledged
6	Non-Standard Parameter Naming	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, Foundry, 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

		🟥 High	🟧 Medium	🟨 Low
Likelihood	🟥 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

Two-Step Ownership Transfer Missing

Acknowledged

Path

ChicagoCoin.sol

Description

Ownership transfer occurs in a single step. If the owner mistakenly transfers ownership to an incorrect or non-receivable address, control of the contract may be permanently lost.

Impact

Potential permanent loss of ownership control if the new owner address is invalid.

Likelihood

Low

Recommendation

Implement a two-step ownership pattern.

Incorrect Operation Order in transferFrom()

Acknowledged

Path

ChicagoCoin.sol

Function Name

`transferFrom`

Description

The transferFrom() function violates the standard checks-effects-interactions pattern by calling _transfer() before validating and updating the allowance. While this doesn’t create a reentrancy vulnerability (due to the lack of external calls), it does emit the Transfer event before confirming the caller has sufficient allowance.

```
function transferFrom(address sender, address recipient, uint256 amount) public
virtual override returns (bool) {
    _transfer(sender, recipient, amount);  // ❌ Transfer happens first

    uint256 currentAllowance = _allowances[sender][_msgSender()];
    require(currentAllowance >= amount, ‘ERC20: transfer amount exceeds
allowance’); // ✓ Check happens second
    unchecked {
        _approve(sender, _msgSender(), currentAllowance - amount);
    }
    return true;
}
```

Impact

May confuse off-chain monitoring tools expecting checks before effects
Deviates from standard ERC-20 implementation patterns

Likelihood

Low

Recommendation

Reorder operations to follow checks-effects-interactions

```
function transferFrom(address sender, address recipient, uint256 amount) public
virtual override returns (bool) {
    // CHECK: Verify allowance first
    uint256 currentAllowance = _allowances[sender][_msgSender()];
    require(currentAllowance >= amount, ‘ERC20: transfer amount exceeds
allowance’);

    // EFFECT: Update allowance
    unchecked {
        _approve(sender, _msgSender(), currentAllowance - amount);
    }

    // INTERACTION: Perform transfer
    _transfer(sender, recipient, amount);

    return true;
}
```

Redundant Constant Supply

Acknowledged

Path

ChicagoCoin.sol

Description

The contract defines a public constant Supply = 1_000_000_000 * (10 ** _decimals) that duplicates the functionality of the standard totalSupply() function. After tokens are burned, these two values will diverge, creating confusion:

Supply: Always returns 1,000,000,000 tokens (initial/maximum supply)
totalSupply(): Returns current circulating supply (decreases after burns)

Impact

User confusion: Block explorers and integrators may be confused about the actual circulating supply
Misleading information: The name “Supply” suggests current supply, but it represents initial/maximum supply

Likelihood

Low

Recommendation

Option1(Recommended): Remove the constant entirely
Option 2 : Rename to clarify it represents initial/maximum supply

Informational Issues

Hardcoded Pre-Mint Recipient Address

Acknowledged

Path

ChicagoCoin.sol

Function Name

`Constructor`

Description

The constructor mints the entire token supply to a hardcoded external address (0x47Be8603CC37C2f64124c7D321127026EC0a1242) instead of using msg.sender or accepting the recipient as a parameter.

Current Implementation:

```
constructor(string memory name_, string memory symbol_) {
    _name = name_;
    _symbol = symbol_;
    _mint(0x47Be8603CC37C2f64124c7D321127026EC0a1242, Supply); // ❌ Hardcoded
}
```

Impact

Reduced deployment flexibility: Cannot easily deploy to different addresses
Testing complications: Difficult to test with different deployment scenarios
Security concern: If private key for hardcoded address is lost, all tokens are lost
Configuration risk: Wrong address cannot be corrected post-deployment

Likelihood

Low

Recommendation

Mint to deployer

Unused Ownable Inheritance

Acknowledged

Path

ChicagoCoin.sol

Description

The contract inherits from Ownable but does not implement any onlyOwner modifiers or owner-restricted functions. This inheritance adds unnecessary code complexity and deployment costs without providing any functionality.

```
constructor(string memory name_, string memory symbol_) {
    _name = name_;
    _symbol = symbol_;
    _mint(0x47Be8603CC37C2f64124c7D321127026EC0a1242, Supply); // ❌ Hardcoded
}
```

Impact

Increased gas costs: Deployment costs are higher due to unused code

Likelihood

Low

Recommendation

Remove Ownable if no owner functions are planned

Non-Standard Parameter Naming

Acknowledged

Path

ChicagoCoin.sol

Function Name

`allowance()` `_approve()`

Description

Several functions use non-standard parameter names that deviate from the ERC-20 specification and common conventions:

```
function allowance(address from, address to) // ❌Should be: owner, spender
function _approve(address from, address to, uint256 amount) // ❌ Should be:
owner, spender
```

Impact

Reduced code readability: Developers familiar with ERC-20 may be confused

Likelihood

Low

Recommendation

Rename parameters to match ERC-20 conventions

Functional Tests

Some of the tests performed are mentioned below:

✓ name() returns constructor value

✓ symbol() returns constructor value

✓ decimals() returns 18

✓ totalSupply() equals initial mint amount

✓ Initial supply minted to 0x47Be8603...1242 (correct recipient/balance)

✓ Reverts when transferring more than balance

✓ Reverts when sender is zero address (via _transfer)

✓ Reverts when recipient is zero address

✓ (Spec-compat) Zero-amount transfer behavior:
- If keeping current code: reverts with “ERC20: transfer amount zero”
- If patched for ERC-20: succeeds and emits Transfer(from, to, 0)

✓ approve(spender, amount) sets allowance correctly

✓ allowance(owner, spender) returns set amount

✓ transferFrom(owner, to, amount) moves tokens and updates balances

✓ transferFrom reduces allowance accordingly (post-call equals previous minus amount)

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.

Threat Model

Contract	Function	Threats
ChicagoCoin	constructor(string name_, string symbol_)	Centralization Risk: Entire supply minted to hardcoded address 0x47Be...1242 — single-key control risk. Misconfiguration Risk: Wrong address leads to permanent token loss.
	name() / symbol() / decimals()	Low risk — static metadata. No external calls or dynamic logic.
	totalSupply()	Low risk — may diverge from Supply constant after burns; could cause UI desync if frontends use constant instead of totalSupply().
	balanceOf(address)	Low risk — direct mapping read; no logic threats.
	transfer(address to, uint256 amount)	Standards Violation: Disallows zero-amount transfers (require(amount > 0)), which may break DEX/bridge/tooling integrations. Safety:Correct zero-address checks.
	allowance(address owner, address spender)	Low risk — naming inconsistency (from, to) only affects readability.
	approve(address spender, uint256 amount)	Allowance Race Condition: Known ERC20 issue — spender can front-run allowance updates. Minor Compatibility Issue: Reverts on zero spender; some systems allow address(0) as sentinel.

Contract	Function	Threats
	transferFrom(address from, address to, uint256 amount)	Check-Effects Order Issue: Decreases allowance after _transfer; may cause confusion and early event emission before allowance check. Integration Risk: Zero-amount disallowed due to _transfer logic.
	increaseAllowance(address spender, uint256 addedValue)	Subject to same race condition risk if spender front-runs allowance changes. Otherwise safe.
	decreaseAllowance(address spender, uint256 subtractedValue)	Correctly reverts on underflow, but same front-running risk applies if spender uses allowance before tx confirmation.
	_transfer(address sender, address recipient, uint256 amount)	Integration DoS: amount > 0 requirement deviates from ERC-20 standard; may revert valid transfers. No Reentrancy: Safe as no external calls.
	_mint(address account, uint256 amount)	Used only in constructor. Fork Risk: If reused without proper access control → infinite minting vulnerability.
	_burn(address account, uint256 amount)	Reduces total supply — ensure dependent contracts handle dynamic supply. No reentrancy risk.
	burn(uint256 amount)	Economic Manipulation Risk: Token holders can burn to affect FDV/ supply-based metrics; acceptable if intended.

Contract	Function	Threats
	_approve(address owner, address spender, uint256 amount)	Minor Deviation: Requires both owner and spender to be non-zero — not standard in all ERC20s; may cause compatibility issues with some integrators.
Ownable	owner()	Low risk — provides owner visibility only.
	transferOwnership(address newOwner)	Currently unused but may introduce admin centralization if future owner-only functions are added.
	renounceOwnership()	If used in future forks with owner-only logic, could cause admin loss / irreversibility.
Context	_msgSender() / _msgData()	Standard utility — no risk.

Note to Users/Trust Assumptions

- The restriction on zero-amount transfers (require(amount > 0) in _transfer) is an intentional design decisionto optimize gas usage and prevent spam or bot-triggered empty transfers. While it deviates from the ERC-20 specification (which permits zero-value transfers), this choice does not pose a security risk but may cause compatibility issues with third-party protocols, DEXs, or bridges that assume full ERC-20 compliance.

- The entire token supply is pre-minted at deployment to a single externally owned address (EOA)0x47Be8603CC37C2f64124c7D321127026EC0a1242.
  - Users must trust the custodian of this address to manage the supply responsibly.
  - Loss or compromise of this private key could result in permanent loss of all tokens.
  - It is strongly recommended to migrate token custody to a multi-signature or time-locked wallet for better decentralization and fund security.

- The contract includes Ownable functionality but currently has no owner-restricted functions. Ownership transfer or renouncement has no functional effect today, but users should be aware that if new features are added in the future (e.g., admin minting, pausing, or governance), the owner's privileges could introduce centralization risks.

- The constant Supply represents the initial minted amount, not the live circulating supply.
  - Because tokens can be burned, totalSupply() may differ from the Supply constant.
  - Integrators and explorers should rely on totalSupply() for accurate on-chain data.

- The burn() function allows any holder to destroy their own tokens, reducing the total supply.
  - This can be used intentionally (e.g., deflationary mechanisms) but may also be exploited to manipulate token metrics such as market capitalization or Fully Diluted Valuation (FDV).
  - Downstream systems relying on fixed supply should account for dynamic supply reduction.
  - The token does not include reentrancy vulnerabilities or privileged minting paths. The _mint() function is internal and only invoked during contract deployment.

- Integration assumptions:
  - External platforms interacting with this token must handle reverts on zero-value transfers gracefully.
  - Automated systems, price feeds, or analytics dashboards should not depend on the constant Supply variable for supply metrics.
  - Wallets, bridges, or DEXs should verify compatibility before integration, as some may reject non-compliant ERC-20 tokens.

User operational assumption:

- Token holders assume that the deploying team will not redeploy or wrap the token in a new contract without formal migration notice.
- Since the contract is immutable and non-upgradeable, future functionality cannot be modified, meaning bugs or policy changes would require redeployment.
- Token holders assume that the vesting and lockup mechanism for the team follows a standard linear vesting schedule, where tokens are gradually released over time and no tokens are burned or revoked once allocated.

-

Closing Summary

In this report, we have considered the security of ChicagoCoin. We performed our audit according to the procedure described above.

No critical issues in ChicagoCoin token, 3 Low severity and 3 issues of Informational severity were found,which was acknowledged by Chicago team

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.

QuillAudits

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

Follow Our Journey

AUDIT REPORT

October 2025

For

CHICAGO

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com