QuillAudits

AUDIT REPORT

October 2025

For

SCOR™

Table of Content

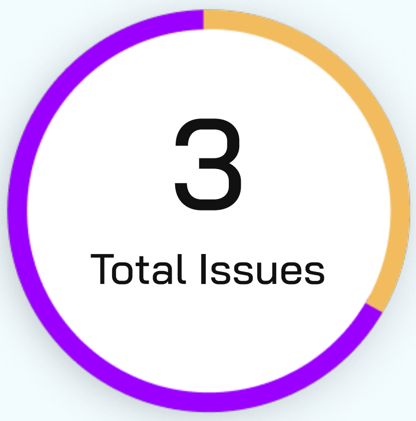
Executive Summary

Project Name	Scor Token
Protocol Type	ERC-20
Project URL	https://scor.io
Overview	ScorToken is an ERC20-based token built using OpenZeppelin standards, inheriting ERC20 and Ownable for reliability and security. It has a capped total supply of 4 billion tokens, minted in full at deployment. Ownership is assigned at contract creation, while all tokens are distributed immediately to the specified address. The contract does not include upgradeability or advanced control mechanisms, keeping it simple and predictable. No additional functionality beyond the ERC20 standard is implemented.
Audit Scope	The scope of this Audit was to analyze the Scor Token Smart Contracts for quality, security, and correctness.
Source Code Link	https://github.com/sweet-io-org/scor-token/blob/master/contracts/scor.sol
Branch	Master
Commit Hash	6cb9f16e7b08d4cb578329a2d971ef6c04d44e4a
Contracts in Scope	ScorToken.sol
Language	Solidity
Blockchain	Base Chain
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	23rd October 2025
Updated Code Received	24th October 2025
Review 2	24th October 2025
Fixed In	e2e2020a30e02a2085ca926783193fccdedabf8a

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	2 (66.6%)

Severity

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

Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Consider using Ownable2Step instead	Low	Resolved
2	Incorrect variable spelling	Informational	Resolved
3	Supply Cap Mechanism Redundancy	Informational	Resolved

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

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

Consider using Ownable2Step instead

Resolved

Path

Scor.sol

Description

The contract currently inherits from OpenZeppelin's Ownable, which uses a single-step ownership transfer pattern. This means that when the owner calls transferOwnership(newOwner), ownership is immediately transferred to the new address. If the new address is mis-typed or not controlled by the intended entity, ownership may be irretrievably lost.

Impact

Ownership could be permanently lost due to accidental mis-transfer.

Loss of administrative control over pausing/unpausing and other future privileged functions.

Likelihood

Low (requires a mistake during ownership transfer).

POC

```
// Owner accidentally calls:
contract.transferOwnership(0xDead…beef);

// Contract ownership is now irrecoverable.
```

Recommendation

Use OpenZeppelin's Ownable2Step, which requires the new owner to explicitly accept ownership via acceptOwnership(). This prevents accidental loss of control.

Informational Issues

Consider using named constant

Resolved

Path

ScorToken.sol

Description

The total supply is currently hardcoded inline using a numeric expression **`(4_000_000_000 * 10 ** decimals())`**. While functionally correct, this expression reduces readability and makes the code less intuitive for developers or auditors. It may also increase the chance of mistakes when modifying or reusing the code.

Impact

Low impact including readability risk

Recommendation

Define a named constant for clarity and reuse across the contract

Supply Cap Mechanism Redundancy

Resolved

Path

Scor.sol

Description

The contract inherits from ERC20Capped and sets a cap of 4,000,000,000 * 10^decimals(). However, the constructor immediately mints the full cap supply to msg.sender. This makes the cap enforcement redundant, since no further minting logic exists. The ERC20Capped extension adds unnecessary complexity without providing functional benefit.

Recommendation

Remove ERC20Capped inheritance and use plain ERC20 instead, since the supply is fixed at deployment.

Functional Tests

Some of the tests performed are mentioned below:

- ✓ Token name is “Scor” and symbol is “SCOR”
- ✓ Decimals is 18.
- ✓ Total supply is 4,000,000,000 SCOR.
- ✓ Successful minting to msg.sender
- ✓ balanceOf reflects correct balances.
- ✓ transfer works correctly.
- ✓ transfer reverts when exceeding balance.
- ✓ approve updates allowance correctly.
- ✓ transferFrom works within allowance.
- ✓ transferFrom reverts if exceeding allowance.
- ✓ Total supply cannot exceed 4,000,000,000 SCOR.
- ✓ Minting beyond cap (if possible) reverts.
- ✓ Ownership transfer works correctly.

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
ScorToken	_mint()	Allows contract to mint token **Intended Branches** • Should mint token to desired address **Negative Behavior** • Should mint on undesired address

Note to Users/Trust Assumptions

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.

Closing Summary

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

No critical issues in Scor token, just 3 issues of Low and Informational severity were found,which has been Fixed by Scor 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

SCOR™

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com