QuillAudits

AUDIT REPORT

March, 2025

For

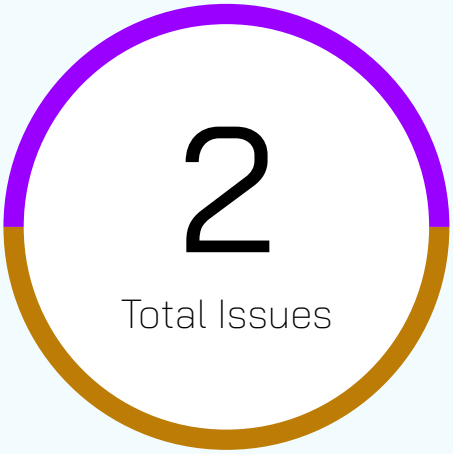
G.O.A.T.AI

Table of Content

Executive Summary

Project name	GOATAI
Overview	GOAT Athletics Inc. (GOATAI) is a community governance meme-token built on the ERC20 standard. The token supports a community-governed running team, and raises funds through buy/sell trading fees to support the Global Running Foundation's initiatives in health and opportunity. The GOATAI_ERC20 contract uses OpenZeppelin contracts, with custom fee-on-transfer logic.
Audit Scope	The Scope of the Audit is to analyse the security and Correctness of GOAT Token Contract.
Contracts in Scope	GOATAI_ERC20.sol
Language	Solidity
Blockchain	Base chain - Ethereum Layer 2
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	07 March 2025 - 11 March 2025
Updated Code Received	14th March 2025
Review 2	15th March 2025
Fixed In	https://github.com/GOAT-Athletics-Inc/contracts/pull/1

Number of Issues per Severity



High	0 (0.00%)
Medium	1 (50.00%)
Low	0 (0.00%)
Informational	1 (50.00%)

Severity

Issues	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	0	**1**	0	**1**
Acknowledged	0	0	0	0
Partially Resolved	0	0	0	0

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

- [x] Using inline assembly
- [x] Style guide violation
- [x] Unsafe type inference
- [x] Implicit visibility level
- [x] Architecture compatibility of GOAT_ERC20 with Governor + TimeLockController

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 statistical analysis.

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 Issues

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

Security vulnerabilities identified that must be resolved and are currently unresolved.

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.

Medium Severity Issues

Unintended Permanent Loss of Admin Control Through Role Renouncement

Resolved

Path

GOATAI_ERC20.sol

Function

NA

Description

The contract inherits OpenZeppelin's AccessControl but doesn't override renounceRole. This means the admin (who has DEFAULT_ADMIN_ROLE) can renounce their role.
If the admin renounces DEFAULT_ADMIN_ROLE.
Then no one can grant new roles,no one can set LP pairs, no one can set exemptions and Admin functions become permanently inaccessible

Recommendation

Override the renounceRole() function from AccessControl to prevent the admin from renouncing the DEFAULT_ADMIN_ROLE

```
function renounceRole(bytes32 role, address account)
        public
        virtual
        override
    {
        require(role != DEFAULT_ADMIN_ROLE, "AccessControl: cannot renounce Admin
role");
        super.renounceRole(role, account);
    }
```

Informational Severity Issues

Use Named Constants Instead of Magic Numbers for Exempt Accounts Limit

Resolved

Path

GOATAI_ERC20.sol

Function

setExempt()

Description

The setExempt function uses a hardcoded number 20 as the maximum accounts limit. Similar to how MAX_NUM_RECIPIENTS and MAX_FEE_BPS are defined as constants, this limit should also be defined as a named constant for maintainability.

Recommendation

Define the limit as a named constant

```
uint256 private constant MAX_EXEMPT_ACCOUNTS_LIMIT = 20;
```

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 GOAT. We performed our audit according to the procedure described above.

1 medium and 1 Informational Issues Found, which the GOAT team Resolved. Moreover, we reviewed the architecture of Governor + TimelockController contracts, keeping in mind that the contract GOATAI_ERC20.sol is compatible with that architecture. It appears to be in fine condition with no concerns.

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 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

7+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

March, 2025

For

G.O.A.T.AI

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com