QuillAudits

AUDIT REPORT

January 2025

For

tria

Table of Content

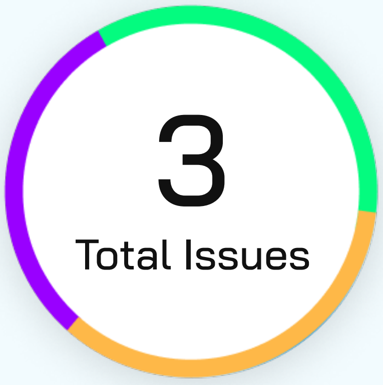
Executive Summary

Project Name	Tria
Project URL	https://www.tria.so/
Overview	The project creates a blockchain-based NFT sales platform with smart contracts that manage token creation, minting, and time-bound sales rounds, providing flexible pricing and robust access control mechanisms.
Audit Scope	The scope of this Audit was to analyze the Tria Smart Contracts for quality, security, and correctness.
Contracts in Scope	1. TriaSale.sol 2. TriaSaleNFT.sol 3. Rounds.sol 4. TokenRegistry.sol
Commit Hash	67e1c2be2fb6fc5257bfaacd7e7f1c814ce3361d
Language	solidity
Blockchain	Ethereum
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	16th January 2025 - 23rd January 2025
Updated Code Received	23rd January 2025
Review 2	23rd January 2025

Executive Summary

Fixed In	1c5414789633374a9b84b6b60126650ba03598cd
Review 3	30th January 2025 *https://github.com/TriaHQ/forge-contracts/tree/feat/final-updates* **Commit hash:** 23b1ee15151ee02f37b81298130a3bb667cd2297 Post Audit (after 2nd review) ,Tria team Made minor changes,ie added function to manually close the round and also removed an unused function. We reviewed it and no new issues found

Number of Issues per Severity



🟥	High	0 (0%)
🟧	Medium	1 (33%)
🟩	Low	1 (33%)
🟪	Informational	1 (33%)

Severity

Issues

	🟥 High	🟧 Medium	🟩 Low	🟪 Informational
Open	0	0	0	0
Resolved	0	1	0	1
Acknowledged	0	0	1	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

Checked Vulnerabilities

- ☑ 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.

Techniques and Methods

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 statistic 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

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

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.

Medium Severity Issues

1. Chainlink’s latestRoundData may return a stale or incorrect result

Resolved

Path

TriaSale.sol

Function Name

getLatestEthUSDPrice

Description

Chainlink’s `latestRoundData` is used here to retrieve price feed data; however, there is insufficient protection against price staleness.

Return arguments other than `int256 answer` are necessary to determine the validity of the returned price, as it is possible for an outdated price to be received.

Return value updatedAt contains the timestamp at which the received price was last updated, and can be used to ensure that the price is not outdated. Inaccurate price data can lead to functions not working as expected and/or loss of funds.

Recommendation

```
function getLatestEthUSDPrice() public view returns (uint256) {

(uint80 roundID, int256 answer, uint256 timestamp , uint256 updatedAt, uint80
answeredInRound) = priceFeed.latestRoundData();,

if (updatedAt + tokenHeartbeat[token] < block.timestamp) revert StalePrice();

require(timestamp != 0,”Round not complete”);

require(answer > 0, “Invalid Price”);
return price.toUint256();

}
```

Low Severity Issues

2. No duplicate Token ID check in `createNewRound` function

Acknowledged

Path

Rounds.sol

Function Name

createNewRound

Description

When the same token ID appears multiple times in the _tokenIds array, contract will overwrite the previous price with the latest occurrence. This means:

1. Only the last price for a duplicate token ID will be stored
2. Previous price entries for the same token ID are silently discarded
3. No warning or error is raised during this process

Recommendation

Add checks of a few things

- Implement a duplicate token ID check before processing
- Revert transaction if duplicate token IDs are found
- Ensure each token ID is unique within a round

Informational Issues

3. Unused state variable called startRound

Resolved

Path

Rounds.sol

Description

startRound is declared as an immutable variable in the Rounds contract;
Initialized to 0 in the constructor however Never used in any function or logic within contract Serves no functional purpose in the current implementation

Recommendation

Remove `startRound` state variable

Functional Tests

Some of the tests performed are mentioned below:

- ✓ Should purchase single NFT with exact ETH
- ✓ Should accept valid signature
- ✓ Should not accept wrong nonce
- ✓ Should not purchase more NFT’s than expected.
- ✓ Should not create round with past start time
- ✓ Should not create round with non-existent tokenIds
- ✓ Should not allow purchase before round starts
- ✓ Should transfer ownership correctly
- ✓ Should pause all operations

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

Some issues of low, medium and informational severity were found. Some suggestions, gas optimizations and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Tria. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Tria. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of Tria to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

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 six years of expertise, we've secured over 1000 projects globally, averting over $30 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

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

Follow Our Journey

AUDIT REPORT

January 2025

For

tria

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com auditsquillhash.com