0xCommit

Security Audit Report

Tronado - ERC20 Token Audit

Version: Final

Date: 17th March 2025

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

Introduction

Purpose of this report

0xCommit has been engaged by **Tronado Token Contact** to perform a security audit of several Solana Programs components.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine solana program bugs, which might lead to unexpected behaviour.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebases Submitted for the Audit

The audit has been performed on the following GitHub repositories:

Version	List of contracts	Source
1	0xC396b3198b5Bd60CF2cDaB9b34F646A58C029998 - On Polygon Network	https://polygonscan.com/address/0xC396b3198b5Bd60CF2cDaB9b34F646A58C029998#code#L1
2	0x238ad4b7b3883bf1946b6eefd396deee28824b12 - On Polygon network	[https://cardona-zkevm.polygonscan.com/address/0x238ad4b7b3883bf1946b6eefd396deee28824b12]

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending**, **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Overview

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Summary of Findings

Sr. No.	Description	Severity	Status
1	Missing SafeMath or Explicit Overflow Protection	Low	Resolved
2	Non-Compliance with ERC20 Standard	Low	Resolved
3	Missing Events during initialization	Low	Resolved
4	Missing documentation	Informational	Resolved
5	Gas Optimizations documentation	Informational	Resolved
6	Ancillary ERC20 Checks	Informational	

Detailed Findings

1. Missing SafeMath or Explicit Overflow Protection

Severity: Low

Description

While Solidity ^0.8.0 includes built-in overflow and underflow protection, explicit checks in critical functions provide an additional layer of security as a defensive programming practice.

Remediation

Add explicit checks for mathematical operations or implement SafeMath Library

Status

Resolved

2. Non-Compliance with ERC20 Standard

Severity: Low

Description

The contract does not fully comply with the ERC20 standard. It fails to implement the interface properly and is missing the `totalSupply()` function required by the standard. This might affect interoperability with other contracts and platforms.

Remediation

Explicitly implement the IERC20 interface and include all required functions. Or use OpenZeppelin's ERC20 Implementation.

Status

Resolved

3. Missing Events during initialization

Severity: Low

Description

The contract does not emit events for critical contract initialization and parameter changes. This makes it difficult for off-chain applications to track important contract state changes.

Remediation

Have transfer emitted during contract initialization.

Status

Resolved

4. Missing documentation

Severity: Low

Description

The contract has limited inline documentation for functions and lacks comprehensive NatSpec comments. This makes it difficult for reviewers, auditors, and developers to understand the contract's functionality and intent.

Remediation

Add Comprehensive NatSpec based documentation for all functions, events and variables in the contract.

Status

Resolved

5. Gas Optimizations documentation

Severity: Informational

Description

Several functions in the contract could be optimized for gas efficiency.

Remediation

1. Use `uint256` instead of `uint8` for decimals to save gas (though this is a common practice)
2. Mark constant values as `constant` or `immutable`:

```
string public constant name = "TRONADO";
string public constant symbol = "TRDO";
uint8 public constant decimals = 18;
```

3. Consider using memory variables in functions with multiple state changes:

```
function transferFrom(address from, address to, uint256 amount)
public returns (bool) {
    require(to != address(0), "Transfer to the zero address is
not allowed");
    uint256 currentAllowance = allowances[from][msg.sender];
    require(currentAllowance >= amount, "Transfer amount exceeds
allowance");
    unchecked {
        allowances[from][msg.sender] = currentAllowance -
amount;
    }
    _transfer(from, to, amount);
    return true;
}
```

4. Update function visibility for external-facing functions. (i.e. -Use external instead of public).

Status

Resolved

6. Ancillary Checks

Severity: Informational

Sr No	Checks	Status
1	Source Code Verified	Yes
2	Is Upgradeable	No
3	Token Contract Mintable	No
4	Admin Balance Change	No
5	Token Backdoor Identified	No
6	Is token contract self destructable	No
7	Is gas intensive contract	No
8	Does token contract has external call risk	No
9	Is Suspension of token feasible	No
10	Has trading cool down	No
11	Has Anti whale functions	No
12	Has any tax component	No
13	Has Blacklist	No
14	Has Whitelist	No