TOREKKO

SMART CONTRACT AUDIT

ZOKYO.

December 30th, 2021 | v. 1.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

SCORE

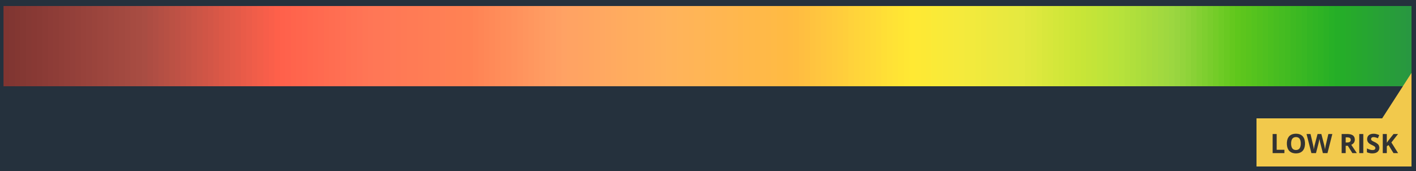
99

TECHNICAL SUMMARY

This document outlines the overall security of the Torekko smart contracts, evaluated by Zokyo's Blockchain Security team.

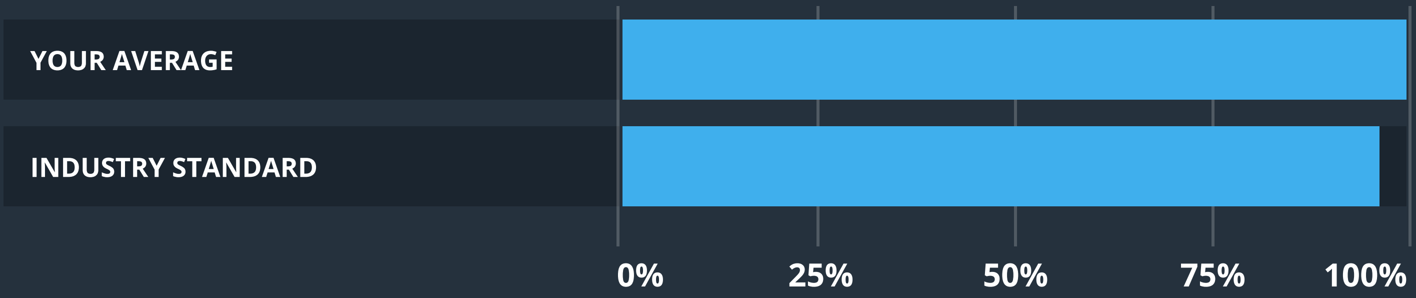
The scope of this audit was to analyze and document Torekko smart contract codebase for quality, security, and correctness.

Contract Status



There were some critical and medium issues found during the audit.

Testable Code



The testable code is 100%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a security of the contract we at Zokyo recommend that the Torekko team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Torekko smart contract's source code was taken from the repository provided by the Torekko team: https://github.com/codeforlife69/TRKContract

Audit commit: 7bcbc0493bce39264fa544c8e4f8941b0ad8eefe
Post-audit commit: 4f5cc91756b418cd0eb9eb733ca1c417698fc478

Within the scope of this audit Zokyo auditors have reviewed the following contract(s):

- TRKContractWithoutMintFn.sol

Throughout the review process, care was taken to ensure that the contract:

- Implements and adheres to existing standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Follows best practices in efficient use of resources, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of Torekko smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1 Due diligence in assessing the overall code quality of the codebase.

2 Cross-comparison with other, similar smart contracts by industry leaders.

3 Testing contract logic against common and uncommon attack vectors.

4 Thorough, manual review of the codebase, line-by-line.

EXECUTIVE SUMMARY

There were some critical and medium issues found during the audit, they relate to the following:

- Asset’s allowance vulnerability
- Allowance manipulation.

After recommendations by Zokyo auditors, all issues that influence security and efficiency were fixed by Torekko Team.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Issues tagged “Verified” contain unclear or suspicious functionality that either needs explanation from the Customer’s side or it is an issue that the Customer disregards as an issue. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

● Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

● High

The issue affects the ability of the contract to compile or operate in a significant way.

● Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

● Low

The issue has minimal impact on the contract’s ability to operate.

● Informational

The issue has no impact on the contract’s ability to operate.

LOW | VERIFIED

Code can be simplified

Line 94. If ownership transferring is disabled in a constructor forever, then the “else” condition is meaningless and will never be reached.

Recommendation:
Reload transferOwnership() method to do nothing instead of meaningless checks.

Post-audit:
Verified by the Client that transferOwnership() is disabled.

LOW | RESOLVED

Code can be simplified

Lines 41, 58. whenNotPaused modifier’s realisation can be replaced into _beforeTokenTransfer function. It will complete the check once every transfer happens. Also this modifier is not necessary in approval functions.

Recommendation:
Replace whenNotPaused modifier’s realisation into _beforeTokenTransfer function in order to save gas and simplify operations.

LOW | RESOLVED

Use standard ERC20 interface

Token does not utilize OpenZeppelin ERC20 token - instead it implements all functions by itself. Though in order to be compatible with most protocols, the token should inherit OpenZeppelin ERC20 functionality and implement the minimum amount of functionality. It is enough in order to be compatible with BEP20 standard (which is identical to ERC20).

Recommendation:
Consider usage of the standard token functionality.

INFORMATIONAL | RESOLVED

Update Solidity version

Standard recommendation from the audit is to use the latest stable Solidity version in the strict manner.

Recommendation:
Use Solidity 0.8.11 in a strict way (without ^).

	TRKContractWithoutMintFn.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Delegatecall Unexpected Ether	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Secured team

As part of our work assisting Torekko team in verifying the correctness of their contract code, our team was responsible for writing integration tests using Truffle testing framework.

Tests were based on the functionality of the code, as well as review of the Torekko contract requirements for details about issuance amounts and how the system handles these.

```
Contract: TRKContractWithoutMintFn
    Initialization->
        ✓ Mints proper amount (122ms)
    Methods->
        ✓ Transfers proper amount(309ms)
        ✓ Can't tranfer from address if not allowed (289ms)
        ✓ Transfers from address if allowed (334ms)
        ✓ Can't unpause when not paused (121ms)
        ✓ Can't pause when paused (109ms)
        ✓ Can't transfer if paused (137ms)
        ✓ Can't transfer from address if paused(145ms)
        ✓ Can't transfer ownership(130ms)
        ✓ Renounces ownrship(178ms)
    9 passing (4s)
```

FILE	% STMTS	% BRANCH	% FUNCS
TRKContractWithoutMintFn.sol	100.00	92	100.00
All files	**100**	**92**	**100**

We are grateful to have been given the opportunity to work with the Torekko team.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Zokyo's Security Team recommends that the Torekko team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.