



CONTRACT
CHECKER

Blockchain Solutions



<https://t.me/contractchecker>

contact@contractchecker.app

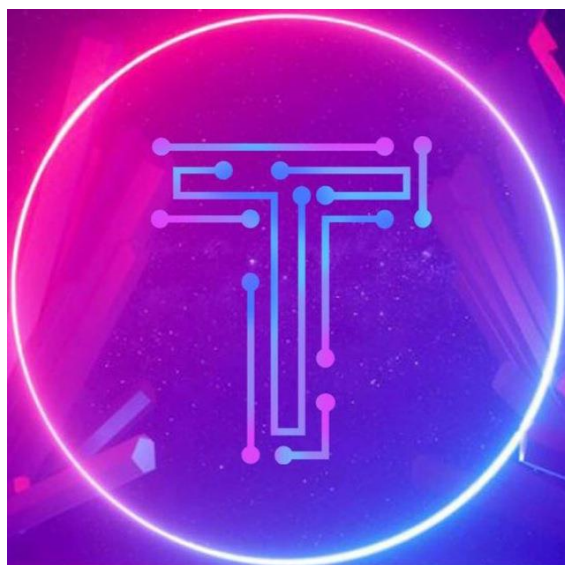
contractchecker.app

Anywhere on the Blockchain

Date: 13.08.2022

Smart Contract Security Audit

TECHNOLAND TOKEN



Harry K

Harry Kedelman
General Manager

Audit Result

🔑 TECHNOLAND TOKEN has successfully **PASSED** the smart contract security audit with below listed privileges.

🔑 **Smart contract meets SAFU requirements**

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

| | |
|-------------------|--|
| Audit Result: | PASSED |
| Ownership: | Not renounced yet |
| KYC Verification: | KYC Verified by ContractChecker |
| Audit Date: | August 13, 2022 |
| Audit Team: | CONTRACTCHECKER |

Findings_ Privileges of Ownership

- ⚠️ Owner can exclude an account from paying fees
- ⚠️ Owner can change the fees but with limit of 15% at max
- ⚠️ Trading must be enabled by the owner
- ⚠️ Owner can change swap settings

Important Notice for Investors

As Contract Checker team we are mainly auditing the contract code to find out how it will be functioning, and risks which are hidden in the code if any.

There are many factors must be taken into consideration before investing to a project, like: ownership status, project team approach, marketing, general market condition, liquidity, token holdings etc.

Investors must always do their own research and manage their risk considering different factors which can affect the success of a project.

Table of Contents

| | |
|---|---|
| Audit Result | 1 |
| Findings_ Privileges of Ownership | 1 |
| Important Notice for Investors | 1 |
| SUMMARY | 3 |
| Project Summary | 3 |
| OVERVIEW | 4 |
| Auditing Approach and Applied Methodologies | 4 |
| Security | 4 |
| Sound Architecture | 4 |
| Code Correctness and Quality | 4 |
| Risk Classification | 4 |
| High level vulnerability | 5 |
| Medium level vulnerability | 5 |
| Low level vulnerability | 5 |
| Vulnerability Checklist | 5 |
| Manual Audit: | 5 |
| Smart Contract SWC Attack Test | 6 |
| ➤ SWC-103: A floating pragma is set | 7 |
| ➤ SWC-108: State variable visibility is not set | 7 |
| ➤ SWC-115: Use of "tx.origin" as a part of authorization control | 7 |
| ➤ SWC-120: Potential use of "block.number" as source of randomness | 7 |
| Automated Audit | 7 |
| Remix Compiler Warnings | 7 |
| Disclaimer | 8 |

SUMMARY

CONTRACTCHECKER received an application for smart contract security audit of TECHNOLAND TOKEN on August 11, 2022, from the project team to discover if any vulnerability in the source codes of the TECHNOLAND TOKEN as well as any contract dependencies. Detailed test has been performed using Static Analysis and Manual Review techniques.

The auditing process focuses to the following considerations with collaboration of an expert team

- Functionality test of the Smart Contract to determine if proper logic has been followed throughout the whole process.
- Manually detailed examination of the code line by line by experts.
- Live test by multiple clients using Testnet.
- Analysing failure preparations to check how the Smart Contract performs in case of any bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analysing the security of the on-chain data.

Project Summary

| | |
|-------------------|---|
| Token Name | TECHNOLAND TOKEN |
| Web Site | https://technoland.io/ |
| Twitter | https://twitter.com/Technolandio |
| Telegram | https://t.me/technolandofficial |
| Platform | Binance Smart Chain |
| Token Type | BEP20 |
| Language | Solidity |
| Platforms & Tools | Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Mythril, Contract Library |
| Contract Address | 0xEC671EEA2971Fc5E7159D8838431CdD24C6bdbCc |
| Contract Link | https://bscscan.com/token/0xEC671EEA2971Fc5E7159D8838431CdD24C6bdbCc |

OVERVIEW

This Audit Report mainly focuses on overall security of TECHNOLAND TOKEN smart contract. Contract Checker team scanned the contract and assessed overall system architecture and the smart contract codebase against vulnerabilities, exploitations, hacks, and back-doors to ensure its reliability and correctness.

Auditing Approach and Applied Methodologies

Contract Checker team has performed rigorous test procedures of the project

- Code design patterns analysis in which smart contract architecture is reviewed to ensure it is structured according to industry standards and safe use of third-party smart contracts and libraries.
- Line-by-line inspection of the Smart Contract to find any potential vulnerability like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.
- Unit testing Phase, we coded/conducted custom unit tests written for each function in the contract to verify that each function works as expected.
- Automated Test performed with our in-house developed tools to identify vulnerabilities and security flaws of the Smart Contract.

The focus of the audit was to verify that the Smart Contract System is secure, resilient, and working according to the specifications. The audit activities can be grouped in the following three categories:

Security

Identifying security related issues within each contract and the system of contract.

Sound Architecture

Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality

A full review of the contract source code. The primary areas of focus include:

- Accuracy
- Readability
- Sections of code with high complexity
- Quantity and quality of test coverage

Risk Classification

Vulnerabilities are classified in 3 main levels as below based on possible effect to the contract.

High level vulnerability

Vulnerabilities on this level must be fixed immediately as they might lead to fund and data loss and open to manipulation. Any High-level finding will be highlighted with **RED** text

Medium level vulnerability

Vulnerabilities on this level also important to fix as they have potential risk of future exploit and manipulation. Any Medium-level finding will be highlighted with **ORANGE** text

Low level vulnerability

Vulnerabilities on this level are minor and may not affect the smart contract execution. Any Low-level finding will be highlighted with **BLUE** text

Vulnerability Checklist

| Nº | Description. | Result |
|----|--|--------|
| 1 | Compiler warnings. | Passed |
| 2 | Race conditions and Re-entrancy. Cross-function race conditions. | Passed |
| 3 | Possible delays in data delivery. | Passed |
| 4 | Oracle calls. | Passed |
| 5 | Front running. | Passed |
| 6 | Timestamp dependence. | Passed |
| 7 | Integer Overflow and Underflow. | Passed |
| 8 | DoS with Revert. | Passed |
| 9 | DoS with block gas limit. | Passed |
| 10 | Methods execution permissions. | Passed |
| 11 | Economy model. | Passed |
| 12 | The impact of the exchange rate on the logic. | Passed |
| 13 | Private user data leaks. | Passed |
| 14 | Malicious Event log. | Passed |
| 15 | Scoping and Declarations. | Passed |
| 16 | Uninitialized storage pointers. | Passed |
| 17 | Arithmetic accuracy. | Passed |
| 18 | Design Logic. | Passed |
| 19 | Cross-function race conditions. | Passed |
| 20 | Safe Zeppelin module. | Passed |
| 21 | Fallback function security. | Passed |

Manual Audit:

For this section the code was tested/read line by line by our developers. Additionally, Remix IDE's JavaScript VM and Kovan networks used to test the contract functionality.

Smart Contract SWC Attack Test

| SWC ID | Description | Test Result |
|---------|---|-------------|
| SWC-100 | Function Visibility | Passed |
| SWC-101 | Integer Overflow and Underflow | Passed |
| SWC-102 | Outdated Compiler Version | Passed |
| SWC-103 | Floating Pragma | LOW |
| SWC-104 | Unchecked Call Return Value | Passed |
| SWC-105 | Unprotected Ether Withdrawal | Passed |
| SWC-106 | Unprotected SELFDESTRUCT Instruction | Passed |
| SWC-107 | Re-entrancy | Passed |
| SWC-108 | State Variable Default Visibility | LOW |
| SWC-109 | Uninitialized Storage Pointer | Passed |
| SWC-110 | Assert Violation | Passed |
| SWC-111 | Use of Deprecated Solidity Functions | Passed |
| SWC-112 | Delegate Call to Untrusted Callee | Passed |
| SWC-113 | DoS with Failed Call | Passed |
| SWC-114 | Transaction Order Dependence | Passed |
| SWC-115 | Authorization through tx.origin | LOW |
| SWC-116 | Block values as a proxy for time | Passed |
| SWC-117 | Signature Malleability | Passed |
| SWC-118 | Incorrect Constructor Name | Passed |
| SWC-119 | Shadowing State Variables | Passed |
| SWC-120 | Weak Sources of Randomness from Chain Attributes | LOW |
| SWC-121 | Missing Protection against Signature Replay Attacks | Passed |
| SWC-122 | Lack of Proper Signature Verification | Passed |
| SWC-123 | Requirement Violation | Passed |
| SWC-124 | Write to Arbitrary Storage Location | Passed |
| SWC-125 | Incorrect Inheritance Order | Passed |
| SWC-126 | Insufficient Gas Griefing | Passed |
| SWC-127 | Arbitrary Jump with Function Type Variable | Passed |
| SWC-128 | DoS With Block Gas Limit | Passed |
| SWC-129 | Typographical Error | Passed |
| SWC-130 | Right-To-Left-Override control character (U+202E) | Passed |
| SWC-131 | Presence of unused variables | Passed |
| SWC-132 | Unexpected Ether balance | Passed |
| SWC-133 | Hash Collisions with Multiple Variable Length Arguments | Passed |
| SWC-134 | Message call with hardcoded gas amount | Passed |
| SWC-135 | Code With No Effects (Irrelevant/Dead Code) | Passed |
| SWC-136 | Unencrypted Private Data On-Chain | Passed |

➤ SWC-103: A floating pragma is set

The current pragma Solidity directive is `">=0.6.0<0.9.0"`. It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

```
5 // SPDX-License-Identifier: MIT
6 pragma solidity >=0.6.0 <0.9.0;
7
8 interface IERC20 {
```

➤ SWC-108: State variable visibility is not set

It is best practice to set the visibility of state variables explicitly. The default visibility for `"lpPairs"` is internal. Other possible visibility settings are public and private.

```
111 mapping (address => uint256) private _owned;
112 mapping (address => bool) lpPairs;
113 uint256 private timeSinceLastPair = 0;
```

The default visibility for `"inSwap"` is internal. Other possible visibility settings are public and private.

```
165 bool inSwap;
166 bool public contractSwapEnabled = false;
167 uint256 public swapThreshold;
```

The default visibility for `"antiSnipe"` is internal. Other possible visibility settings are public and private.

```
172 bool public tradingEnabled = false;
173 bool public _hasLiqBeenAdded = false;
174 AntiSnipe antiSnipe;
```

➤ SWC-115: Use of `"tx.origin"` as a part of authorization control

The `tx.origin` environment variable has been found to influence a control flow decision. Note that using `"tx.origin"` as a security control might cause a situation where a user inadvertently authorizes a smart contract to perform an action on their behalf. It is recommended to use `"msg.sender"` instead.

```
449 if to != _owner
450     if tx.origin != _owner
451         !_liquidityHolders[to]
```

➤ SWC-120: Potential use of `"block.number"` as source of randomness

The environment variable `"block.number"` looks like it might be used as a source of randomness. Note that the values of variables like `coinbase`, `gaslimit`, `block number` and `timestamp` are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

```
544 try antiSnipe.setLaunch(lpPair, uint32(block.number), uint64(block.timestamp), _decimals) {} catch {}
545 tradingEnabled = true;
546 allowedPresaleExclusion = false;
```

Automated Audit

Remix Compiler Warnings

It throws warnings by Solidity's compiler. No issues found.

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. To get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us based on what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and ContractChecker and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (ContractChecker) owe no duty of care towards you or any other person, nor does ContractChecker make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and ContractChecker hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, ContractChecker hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against ContractChecker, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report.

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. If you have any doubt about the Genuity for this document, please check QR code:

