QuillAudits

AUDIT REPORT

December, 2024

For

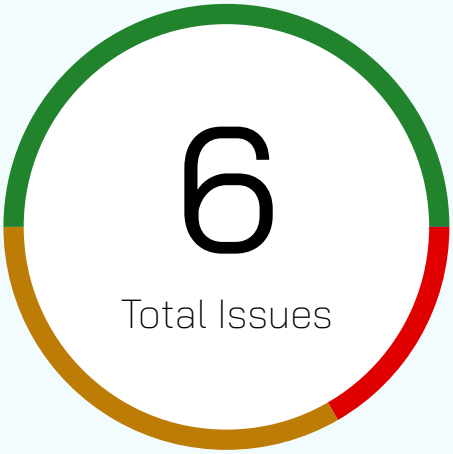
TRRUE

Table of Content

Executive Summary

Project Name	Trrue
Project URL	www.Trrue.io
Overview	Trrue is a blockchain-based platform focused on creating sustainable and compliant digital asset solutions.
Audit Scope	The scope of this audit was to analyse the Trrue Token Contract for quality, security, and correctness.
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Source Code	https://gitlab.com/trrue/trc20/-/blob/master/contracts/UpgradableToken.sol commit: 83d4998cd0a1cc632418e2281cf5b77dbb63a805
Branch	Master
Blockchain	EVM
Contracts_in_Scope	UpgradeableToken.sol
Timeline	9th December 2024 - 11th December 2024
Updated Code received	16th December 2024
Second review	16th December 2024
Fixed In	52494bfd23568dc7f46739ea0bb067800672deeb

Number of Issues per Severity



High	1 (16.67%)
Medium	2 (33.33%)
Low	3 (50.00%)
Informational	0 (0.00%)

Severity

Issues	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	**1**	**2**	**3**	0
Acknowledged	0	0	0	0
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- ☑ Re-entrancy
- ☑ Timestamp Dependence
- ☑ Gas Limit and Loops
- ☑ Transaction-Ordering Dependence
- ☑ DoS with Block Gas Limit
- ☑ Use of tx.origin
- ☑ Exception disorder
- ☑ Gasless send
- ☑ Balance equality
- ☑ Byte array
- ☑ Transfer forwards all gas
- ☑ Compiler version not fixed
- ☑ Redundant fallback function
- ☑ Style guide violation
- ☑ Unchecked external call
- ☑ Send instead of transfer
- ☑ Unchecked math
- ☑ Unsafe type inference
- ☑ Implicit visibility level
- ☑ Private modifier
- ☑ Revert/require functions
- ☑ Multiple Sends
- ☑ Using suicide
- ☑ Using delegatecall
- ☑ Upgradeable safety
- ☑ Using inline assembly
- ☑ Style guide violation
- ☑ Unsafe type inference
- ☑ Implicit visibility level
- ☑ Missing Zero Address Validation
- ☑ Integer overflow/underflow
- ☑ Divide before multiply

- ☑ Address hardcoded
- ☑ SWC Registry
- ☑ Race conditions/front running
- ☑ Arithmetic Computations Correctness
- ☑ Logical issues and flaws
- ☑ Centralization of control
- ☑ Arbitrary write to storage

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 Issues

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.

Issue Status

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.

High Severity Issues

UpgradeableToken missing critical upgrade functionality

Resolved

Path

UpgradeableToken.sol

Description

UpgradeableToken.sol imports multiple upgradeable OpenZeppelin contracts:
-ERC20Upgradeable
-OwnableUpgradeable,
-Initializable,
-ERC20BurnableUpgradeable,
-ERC20PausableUpgradeable, and
-ERC20VotesUpgradeable
but does not explicitly handle the upgradeability mechanism of its own contract. Unless the deployer intends to handle the proxy deployment and redeployment mechanism manually (proxy contract and deployment scripts are not included in the scope of this audit), you can make use of the industry standards available.

Without proper upgradeability patterns, the contract is at risk of:
being upgraded to a new malicious implementation by an attacker
introducing newer state variables and functions that modify core functionality
overriding the initializer and manipulating previous state

Recommendation

Import the UUPSUpgradeable proxy package, use the OpenZeppelin Contracts Wizard to work with a blueprint for safer building decisions.
Have only authorized addresses perform contract upgrades.

```
+import "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
+contract UpgradableToken is UUPSUpgradeable, … {}
+function _authorizeUpgrade(
address newImplementation
 )internal override onlyOwner {}
```

Medium Severity Issues

Contract can be bricked by renouncing ownership

Resolved

Path

UpgradeableToken.sol, OwnableUpgradeable.sol

Function

renounceOwnership()

Description

The current contract uses the OwnableUpgradeable library that provides the specified owner with administrative functionality over functions in the contract that the onlyOwner modifier is assigned. The issue lies in renouncing ownership without the contract being in a proper 'safe' state. Say for example the contracts were paused, every function that uses the internal _update function (transfer, transferFrom, mint and burn) will be halted causing a DOS to holders of the token.

Recommendation

-Override the revokeOwnership function if it is not intended to be used./n
-Implement two-step ownership verification before transfer by using Ownable2StepUpgradeable.

Token permits cannot be granted

Resolved

Path

UpgradeableToken.sol

Function

nonces(...)

Description

The nonces function returns the next nonce for an address to be used during the function execution. It is a critical part of the ERC20Permit specification. Currently, the nonces function is overridden in UpgradeableToken.sol and has no code in its function body making it to return 0 on every call. The returned value will be the same for every function call thereby defeating the purpose of creating a self-incrementing value per transaction that cannot be reused. On the second call to the nonce, this value would remain the same and halt other transactions because the signatures do not match as expected.

Recommendation

Implement the nonces function in the contract by including logic in the contract.

Low Severity Issues

Total supply can be less than 10^18 decimals

Resolved

Description

The token currently has no checks or scaling factors in place for the minting of tokens in the initializer. Generally tokens are scaled to 18 decimals as a best practice to handle fungibility efficiently but the current implementation does not guarantee this.

Recommendation

Ensure the deployment script / manual deployer is aware of this before deploying to mainnet.

Contract ownership can transferred to the wrong owner

Resolved

Path

UpgradeableToken.sol, OwnableUpgradeable.sol

Function

transferOwnership()

Description

Access control is a critical part of smart contract security. Within this contract, access control is handled by the Ownable library which allows the owner to relinquish their rights to a new owner in one function call, transferOwnership(). This is risky if the new owner address passed in is malicious or is a victim of an address poisoning attack where some bytes of the address are changed when copied to the clipboard.

Recommendation

Implement two-step ownership that includes confirmation of ownership transfer from the new owner before it is accepted.

Missing _disableInitializers() method and constructor

Resolved

Description

In upgradeable smart contracts using the OpenZeppelin Initializable pattern, it is crucial to prevent the implementation contract from being initialized multiple times. The _disableInitializers() method in the constructor provides a critical security mechanism to mitigate potential initialization attacks.

Without _disableInitializers(), the implementation contract's initialize() function remains vulnerable to:/n
-Multiple initialization attempts/n
-Potential state manipulation/n
-Proxy contract initialization bypass

Recommendation

-Confirm _disableInitializers() is called in constructor/n
-Validate that initialize() can only be called once

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

Some issues of High, Medium and Low severity were found. Some suggestions, gas optimizations and best practices are also provided in order to improve the code quality and security posture.In the End,Trrue Team Resolved all Issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Trrue. 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 Trrue. 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 Trrue to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

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

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

Follow Our Journey

AUDIT REPORT

December, 2024

For

TRRUE

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com