QuillAudits

Audit Report
August, 2024

For

Table of Content

Executive Summary

Project Name	Triskel
Overview	The Triskel project implements two key components: a Utility Token compliant with the ERC20 standard for general-purpose token use and a Security Token adhering to the ERC-3643 standard for permissioned token management. The ERC-3643-based Security Token enables secure and regulated issuance, management, and transfer of tokens, leveraging advanced identity verification and modular compliance features to ensure robust asset management.
Timeline	25th July 2024 - 31st July 2024
Updated Code Received	8th August 2024
Second Review	09th August 2024
Method	The scope of this audit was to analyze the Demex Contracts for quality, security, and correctness.
Audit Scope	The scope of this audit was to analyse the Triskel Contracts for quality, security, and correctness.
Source Code	https://github.com/TheRavneet/Triskel-Smartcontract/tree/master
Contracts In-Scope	**Commit:** aaa28882d7e75d0d3608903ba160577b1b2f1ae5 files: └── contracts ├── Master.sol ├── factory │ └── IdFactory.sol └── UtilityToken.sol
Branch	Master
Fixed In	NA

Number of Security Issues per Severity

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	2	5
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	0

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- 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's 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

Hardhat, Foundry.

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.

Low Severity Issues

1. Unrestricted createTokenIdentity Function Susceptible to Front-Running Attacks

Path

factory/IdFactory.sol

Function

createTokenIdentity

Description

The createTokenIdentity function in the IdFactory contract lacks access control, allowing anyone to call it and potentially front-run legitimate users. During the token creation process, the token owner provides a unique salt for the deployed token. An attacker could exploit this lack of access control by front-running the legitimate user's transaction to create a token identity with the same salt, causing the original transaction (e.g., deployTREXSuite) to fail. This results in a temporary Denial of Service (DoS), as the token creation process will be disrupted. While the issue is temporary and the owner can use a different salt to deploy the token, this front-running vulnerability poses a risk to the intended token deployment process.

Recommendation

Implement access control for the createTokenIdentity function to restrict its usage to authorized addresses or roles. This will prevent unauthorized actors from interfering with token creation and mitigate the risk of front-running attacks, ensuring a more secure and reliable deployment process.

Status

Acknowledged

2. Unrestricted createIdentity Function Vulnerable to Front-Running and Denial of Service

Path

factory/IdFactory.sol

Function

createIdentity

Description

The createIdentity function in the IdFactory contract is susceptible to front-running attacks due to the lack of access control. Anyone can call this function and create an identity for any user. If the salt used for identity creation is predictable, a malicious actor can exploit this vulnerability by front-running legitimate identity creation requests. The attacker could preemptively create an identity with the same salt, causing subsequent attempts by legitimate users or protocol agents to register identities with the same salt to fail. This can lead to a Denial of Service (DoS) for legitimate operations, particularly if identity registration is automated and the salt is generated in a predictable manner.

Recommendation

Implement access control for the createIdentity function to restrict its use to authorized addresses or roles.

Status

Acknowledged

Informational Issues

3. Mint Function in UtilityToken is Redundant

Path

UtilityToken.sol

Function

mint

Description

The mint function in the UtilityToken contract is redundant because all tokens (500,000,000 * 10^18) are already distributed during contract construction. As a result, the mint function and the Ownable functionality, which restricts minting to the owner, are unnecessary and do not serve any purpose. The token supply is fully allocated at deployment, making additional minting operations redundant and potentially increasing the attack surface for unintended behavior.

Recommendation

Remove the mint function from the UtilityToken contract to simplify the contract and eliminate unnecessary functionality. Additionally, since the Ownable functionality is only used to restrict access to the mint function, consider removing Ownable as well to further streamline the contract.

Status

Acknowledged

4. Temporary DoS Risk from Unrestricted TREX Factory Setup

Path

Master.sol

Function

setTrexFact

Description

There is a potential for a temporary Denial of Service (DoS) due to the unrestricted setup of the TREX factory in the Master contract. A malicious actor could set the trexFactory address to a malicious contract immediately after the Master contract deployment. This would require the contract owner to notice and update the trexFactory address to a legitimate one before creating any tokens. Failure to do so could result in token creation through a malicious contract, compromising the integrity of the token deployment process.

Recommendation

To mitigate this risk, consider implementing one of the following solutions:

- **Access Control for TREX Factory Setup:** Restrict the ability to set the trexFactory address to specific roles or addresses, preventing unauthorized changes.
- **Set TREX Factory in Constructor:** Define the trexFactory address during the contract deployment in the constructor, ensuring it is set to a legitimate address from the outset and reducing the risk of manipulation.

Status

Acknowledged

5. Security Token Supply Limit Potentially Unverified Due to Unknown Compliance Module Implementation

Path

Master.sol

Function

mintTokens

Description

The Master contract relies on an external compliance module to enforce the maximum supply limit of 100,000 security tokens, as specified in the whitepaper. However, this compliance module was not included in the audit and its implementation cannot be verified. As a result, there is a potential risk that the supply limit could be bypassed if the compliance module is not properly integrated or if its functionality is compromised. The lack of verification for this module introduces uncertainty regarding the enforcement of the supply cap.

Recommendation

Ensure that the compliance module responsible for enforcing the token supply limit is included in the audit process for full verification or consider implementing a direct supply cap enforcement mechanism within the Master contract itself to ensure compliance with the supply limit, regardless of the external module.

Status

Acknowledged

6. Missing Development Environment Setup and Tests

Description

The project currently lacks a documented development environment setup and comprehensive test coverage. Without proper environment setup instructions, developers may face difficulties replicating the development conditions or running the project locally. Additionally, the absence of detailed test cases or automated tests increases the risk of undetected bugs and vulnerabilities, which could compromise the stability and security of the smart contracts. This deficiency hinders the ability to verify the correctness and reliability of the code through consistent and repeatable testing.

Recommendation

Provide Development Environment Setup: Utilize tools such as Foundry or Hardhat to set up the development environment. Provide detailed instructions and configurations for using these tools to ensure that developers can easily replicate the environment and start working on the project.

Create Automated Tests: Develop automated tests using Foundry or Hardhat to cover all critical functions and components of the smart contracts. Ensure these tests include various scenarios and edge cases to thoroughly verify the correctness and security of the code. Integrate these tests into a continuous integration (CI) pipeline to regularly check code quality and detect issues early.

Status

Acknowledged

7. Inconsistent Naming Conventions for Events and Functions

Path

Master.sol

Description

The project does not fully adhere to Ethereum's naming conventions for events and functions. According to Ethereum code standards, event names should start with a capital letter and function names should use camelCase. However, in the provided contracts, event names such as tokenMint and deployed do not follow the recommended capitalization. Similarly, function names like batchdeployIdentity do not adhere to the camelCase convention. This inconsistency can lead to confusion and reduced readability of the code.

Recommendation

It is recommended to fix aforementioned function and event names.

Status

Acknowledged

Functional Tests Cases

Best practices in T-REX framework

Ensure that the best practices according to T-REX framework are kept.

Best practices in ERC-20 standard

Ensure that the best practices according to ERC-20 standard are kept.

Assessing differences in IdFactory.sol

Ensure that the modifications in IdFactory.sol contracts are not breaking other parts of T-REX framework.

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 the Triskel codebase. We performed our audit according to the procedure described above.

Some issues of Low and informational severity were found, Some suggestions 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 Triskel smart contracts. 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 Triskel smart contracts. 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 the Triskel 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.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
August, 2024

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com