QuillAudits

AUDIT REPORT

January 2026

For

DeFinix

Table of Content

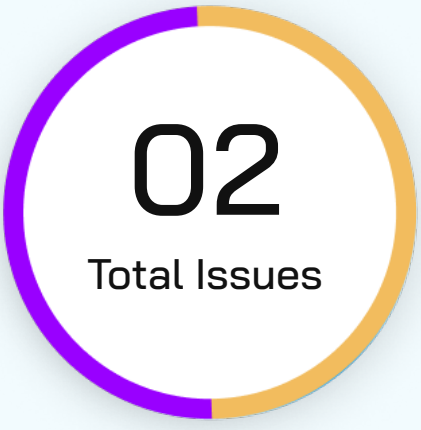
Executive Summary

Project Name	DFX
Protocol Type	Token
Project URL	https://definix.net/en/
Overview	DefinixToken is an ERC20-based token built using OpenZeppelin standards, ensuring reliability and security. It has a fixed total supply, all minted to the deployer at contract creation.
Audit Scope	The scope of this Audit was to analyze the DFX Token Smart Contracts for quality, security, and correctness.
Source Code link	https://bscscan.com/address/0xba509fde406f399852e71e8d9635887616d29f2d#code
Contracts in Scope	DefinixToken.sol
Branch	main
Commit Hash	7e8467865e4a86d36b710ebc34fedd0fbae0bfb0
Language	Solidity
Blockchain	Binance Smart Chain
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	8th January 2026
Updated Code Received	8th January 2026
Review 2	8th January 2026

Verify the Authenticity of Report on QuillAudits Leaderboard:

https://www.quillaudits.com/leaderboard

Number of Issues per Severity



Critical	0(0.0%)
High	0(0.0%)
Medium	0(0.0%)
Low	1 (50.0%)
Informational	1 (50.0%)

Severity

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

Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Use latest version of Solidity	Low	Acknowledged
2	Use of Magic Numbers Instead of Named Constants	Informational	Acknowledged

Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Computations Correctness
- Race conditions/front running
- SWC Registry
- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC's conformance
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls

- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Multiple Sends
- Using suicide
- Using delegatecall
- Upgradeable safety
- Using throw
- Using inline assembly
- Style guide violation
- 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 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 Static Analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are five levels of severity, and each of them has been explained below.

Critical: Immediate and Catastrophic Impact

Critical issues are the ones that an attacker could exploit with relative ease, potentially leading to an immediate and complete loss of user funds, a total takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

High (H): Significant Risk of Major Loss or Compromise

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

Medium (M): Potential for Moderate Harm Under Specific Circumstances

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

Low (L): Minor Imperfections with Limited Repercussions

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

Informational (I): Opportunities for Improvement, Not Immediate Risks

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.

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.

Severity Matrix

Impact

Likelihood	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

Low Severity Issues

Use latest version of Solidity

Acknowledged

Path

DefinixToken.sol

Function Name

`constructor()`

Description

The contract is written using Solidity ^0.5.1, which is a deprecated compiler version. Solidity versions prior to 0.8.x lack several important safety features, most notably built-in overflow and underflow checks. Additionally, older versions receive no security patches and are increasingly unsupported by modern tooling, auditors, and static analyzers.

Impact

While this specific contract is simple and does not currently perform arithmetic that is user-controlled, using an outdated compiler increases long-term maintenance risk and makes the contract more error-prone if extended in the future.

Likelihood

The issue is guaranteed to exist as long as the compiler version remains unchanged.

Recommendation

Upgrade to the latest stable Solidity version (^0.8.x). This will:

- Enable automatic arithmetic overflow/underflow protection
- Improve compiler diagnostics
- Ensure compatibility with modern tooling and security practices

Informational Issues

Use of Magic Numbers Instead of Named Constants

Acknowledged

Path

DefinixToken.sol

Function

`constructor()`

Description

The total token supply is defined using a hard-coded literal value
This is a classic example of a magic number, which reduces readability and makes future audits, refactors, or supply changes more error-prone.

Impact

This does not introduce a direct security vulnerability but negatively affects code clarity and maintainability.

Likelihood

The pattern is already present and will persist unless refactored.

Recommendation

Define the token supply using a clearly named constant to improve readability and reduce cognitive load for auditors and developers.

Functional Tests

Some of the tests performed are mentioned below:

- **Deployer receives full initial token supply**
- **Constructor sets correct token name, symbol and decimal**
- **No additional minting is possible after deployment**

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.

Threat Model

1. External Dependencies & Trust Boundaries

This section enumerates everything the protocol does not fully control.

1.1 External Dependencies Table

Dependency	Type	How It Is Used	Trust Assumption	Associated Risks
ERC20.sol	Library/Base Contract	Core Token logic (balances,	Correct implementation of ERC20 invariants	N/A
ERC20Detailed.sol	Library/ Metadata	Token name, symbol and decimals	Metadata is static and correct	UI integration
Solidity Compiler	Tool	Compilation	Compiler has no known critical bugs	Older version can lead to unsafe defaults

2. Entry & Exit Points Analysis (Function-Level)

This is the core of the threat model. Every externally callable function must appear here.

2.1 Function Entry / Exit Table

Each function gets one table.

Contract Name	Function	Category
DefinixToken.sol	constructor()	**What this function can do** Mint the full token supply and assign it to msg.sender **What this function cannot/should not do** Mint again after deployment, assign tokens to arbitrary addresses **Main invariant(s)** Total supply is fixed at deployment **Access level** Implicit restriction (once at deployment)

3. Asset Flow Mapping (Critical Contracts)

This section identifies where money actually lives and how it moves.

3.1 Asset-Holding Contracts

List only contracts that custody value.

Contract	Asset Type	Custodied Assets
DefinixToken.sol	ERC20	Entire Token Supply

3.2 Asset Entry & Exit Functions

This table maps money movement, which is where most exploits happen.

Contract	Function	Asset In	Asset Out	Caller	Risk Notes
DefinixToken .sol	Constructor	-	DFN	Deployer	Centralized initial distribution
Vault.sol	transfer	DFN	DFN	Public	N/A
Strategy.sol	transferFrom	DFN	DFN	Public	Allowance Risk

Note to Users/Trust Assumptions

- **Centralization Risk**

single-address control (high): Because the entire supply was minted to one recipient address at deployment, that address controls the entire circulating supply until tokens are distributed. If that address is a team, exchange, or a private wallet, they can move or sell tokens at any time. Treat tokens with a single initial holder as high-risk for sudden dumps. Verify who controls the recipient.

Closing Summary

In this report, we have considered the security of Definix Token. We performed our audit according to the procedure described above.

No critical issues in Definix token, just 2 issues of Low and Informational severity were found. Definix Token acknowledged all the issues mentioned above.

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

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 seven years of expertise, we've secured over 1400 projects globally, averting over $3 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.

7+ Years of Expertise	**1M+** Lines of Code Audited
50+ Chains Supported	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

January 2026

For

DeFinix

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com