Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	0x Protocol Token
Website:	0x.org
Platform:	Ethereum
Language:	Solidity
Date:	February 24th, 2024

Table of contents

Email: audit@EtherAuthority.io

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the 0x Protocol Token smart contract from 0x.org was audited extensively. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on February 24th, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- The `ZRXToken` contract inherits from `UnlimitedAllowanceToken` and represents the actual ZRX token. It sets the token's parameters such as `name`, `symbol`, `decimals`, and `totalSupply`. The total supply is set to 1 billion tokens with 18 decimal places, and all tokens are initially assigned to the contract creator.
- This code provides a basic implementation of an ERC20 token contract for the ZRX Protocol Token, including functionality for transferring tokens, managing allowances, and emitting events.
- The contract is without any other custom functionality and without any ownership control, which makes it truly decentralized.

Audit scope

Name	**Code Review and Security Analysis Report for 0x Protocol Token Smart Contract**
Platform	**Ethereum**
Language	**Solidity**
File	ZRXToken.sol
Smart Contract Code	0xe41d2489571d322189246dafa5ebde1f4699f498
Audit Date	February 24th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: 0x Protocol Token • Symbol: ZRX • Decimals: 18 • Total Supply: 1 billion	**YES, This is valid.**
Ownership Control: • There are no owner functions, which makes it 100% decentralized.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **“Secured”**. This token contract does not have any ownership control, hence it is **100% decentralized**.

We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low, and 5 very low level issues.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version not specified	Passed
	Solidity version is too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Moderated
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Business Risk Analysis

Category	Result
Buy Tax	0%
Sell Tax	0%
Cannot Buy	No
Cannot Sell	No
Max Tax	0%
Modify Tax	No
Fee Check	Not Detected
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	Not Detected
Pause Transfer?	Not Detected
Max Tax?	No
Is it Anti-whale?	Not Detected
Is Anti-bot?	Not Detected
Is it a Blacklist?	No
Blacklist Check	No
Can Mint?	No
Is it a Proxy?	No
Can Take Ownership?	No
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract. Smart contract contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in 0x Protocol Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the 0x Protocol Token.

The EtherAuthority team has no scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are not well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a 0x Protocol Token smart contract code in the form of an Etherscan web link.

As mentioned above, code parts are not well commented on. but the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure are based on well-known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	ZRXToken	write	Passed	No Issue
3	transferFrom	write	Passed	No Issue
4	transfer	write	Passed	No Issue
5	transferFrom	write	Passed	No Issue
6	balanceOf	write	Passed	No Issue
7	approve	write	Passed	No Issue
8	allowance	write	Passed	No Issue
9	totalSupply	write	Unused local functions	Refer Audit Findings
10	balanceOf	write	Unused local functions	Refer Audit Findings
11	transfer	write	Unused local functions	Refer Audit Findings
12	transferFrom	write	Unused local functions	Refer Audit Findings
13	approve	write	Unused local functions	Refer Audit Findings
14	allowance	write	Unused local functions	Refer Audit Findings

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium Severity vulnerabilities were found.

Low

No Low Severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Use the latest solidity version:

```
pragma solidity 0.4.11;
```

Using the latest solidity will prevent any compiler-level bugs.

Resolution: We suggest using the latest solidity compiler version.

(2) Declare variables constant:

```
uint8 constant public decimals = 18;
uint public totalSupply = 10**27; // 1 billion tokens, 18 decimal places
```

These variables' values will remain unchanged. so, we suggest making them constant. It is best practice and it also saves some gas. Just add a constant keyword.

Resolution: Please suggest making variables constant.

(3) State variable shadowing:

```
    mapping (address => uint) balances;
    mapping (address => mapping (address => uint)) allowed;
    uint public totalSupply;
}

contract UnlimitedAllowanceToken is StandardToken {···
}

contract ZRXToken is UnlimitedAllowanceToken {

    uint8 constant public decimals = 18;
    uint public totalSupply = 10**27; // 1 billion tokens, 18 decimal places
    string constant public name = "0x Protocol Token";
    string constant public symbol = "ZRX";
```

The totalSupply variable is defined as multiplied in ZRXToken and StandardToken contracts. StandardToken inherited in ZRXToken

Resolution: We suggest removing the state variable shadowing.

(4) Unused local functions:

```
  /// @return total amount of tokens
  function totalSupply() constant returns (uint supply) {}

  /// @param _owner The address from which the balance will be retrieved
  /// @return The balance
  function balanceOf(address _owner) constant returns (uint balance) {}

  /// @notice send `_value` token to `_to` from `msg.sender`
  /// @param _to The address of the recipient
  /// @param _value The amount of token to be transferred
  /// @return Whether the transfer was successful or not
  function transfer(address _to, uint _value) returns (bool success) {}

  /// @notice send `_value` token to `_to` from `_from` on the condition it is approved by `_from`
  /// @param _from The address of the sender
  /// @param _to The address of the recipient
  /// @param _value The amount of token to be transferred
  /// @return Whether the transfer was successful or not
  function transferFrom(address _from, address _to, uint _value) returns (bool success) {}
```

```
function approve(address _spender, uint _value) returns (bool success) {}

/// @param _owner The address of the account owning tokens
/// @param _spender The address of the account able to transfer the tokens
/// @return Amount of remaining tokens allowed to spent
function allowance(address _owner, address _spender) constant returns (uint remaining) {}
```

These are all functions already defined in StandardToken.

- balanceOf(address _owner),
- transfer(address _to, uint _value),
- transferFrom(address _from, address _to, uint _value),
- approve(address _spender, uint _value),
- allowance(address _owner, address _spender)

Resolution: We suggest removing unused local functions from the Token contract.

(5) Visibility can be external over the public:
Any functions which are not called internally should be declared as external. This saves some gas and is considered a good practice.
https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices

Centralization Risk

The 0x Protocol Token smart contract does not have any ownership control, **hence it is 100% decentralized.**

Therefore, there is **no** centralization risk.

Conclusion

We were given a contract code in the form of Etherscan web links. And we have used all possible tests based on given objects as files. We observed 5 Informational issues in the smart contracts. but those are not critical. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Email: audit@EtherAuthority.io

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's website to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

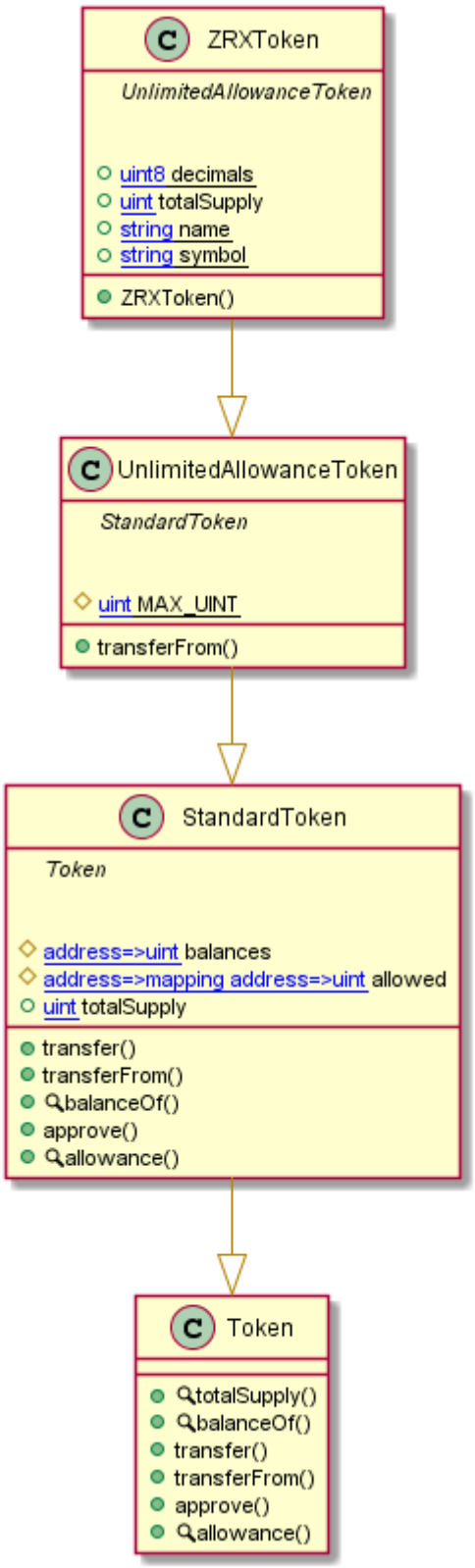
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - 0x Protocol Token



Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

Slither Log >> ZRXToken.sol

```
INFO:Detectors:
ZRXToken.totalSupply (ZRXToken.sol#137) shadows:
        - StandardToken.totalSupply (ZRXToken.sol#100)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
UnlimitedAllowanceToken.transferFrom(address,address,uint256).allowance (ZRXToken.sol#116) shadows:
        - StandardToken.allowance(address,address) (ZRXToken.sol#94-96) (function)
        - Token.allowance(address,address) (ZRXToken.sol#56) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Pragma version0.4.11 (ZRXToken.sol#23) allows old versions
solc-0.4.11 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter StandardToken.transfer(address,uint256)._to (ZRXToken.sol#64) is not in mixedCase
Parameter StandardToken.transfer(address,uint256)._value (ZRXToken.sol#64) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._from (ZRXToken.sol#74) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._to (ZRXToken.sol#74) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._value (ZRXToken.sol#74) is not in mixedCase
Parameter StandardToken.balanceOf(address)._owner (ZRXToken.sol#84) is not in mixedCase
Parameter StandardToken.approve(address,uint256)._spender (ZRXToken.sol#88) is not in mixedCase
Parameter StandardToken.approve(address,uint256)._value (ZRXToken.sol#88) is not in mixedCase
Parameter StandardToken.allowance(address,address)._owner (ZRXToken.sol#94) is not in mixedCase
Parameter StandardToken.allowance(address,address)._spender (ZRXToken.sol#94) is not in mixedCase
Parameter UnlimitedAllowanceToken.transferFrom(address,address,uint256)._from (ZRXToken.sol#112) is not in mixedCase
Parameter UnlimitedAllowanceToken.transferFrom(address,address,uint256)._to (ZRXToken.sol#112) is not in mixedCase
Parameter UnlimitedAllowanceToken.transferFrom(address,address,uint256)._value (ZRXToken.sol#112) is not in mixedCase

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
ZRXToken.totalSupply (ZRXToken.sol#137) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-const
ant
INFO:Slither:ZRXToken.sol analyzed (4 contracts with 93 detectors), 18 result(s) found
```

Software analysis result:

This software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io