IDEX 2.0 Audit Report

Version 1.0.0

Serial No. 2021101500012015

Presented by Fairyproof

October 15, 2021

FAIRYPROOF

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the IDEX2.0 project.

Audit Start Time:

October 11, 2021

Audit End Time:

October 14, 2021

Audited Source Files' Ethereum Onchain Addresses:

Governance: 0xC883C1774BC4e699dFd3ABD122FDB751702B7146

Exchange: 0xA36972E347E538E6C7Afb9f44FB10DDa7BBa9BA2

Custodian: 0xE5c405C5578d84c5231D3a9a29Ef4374423fA0c2

Audited Contract Files' Addresses:

https://etherscan.io/address/0xC883C1774BC4e699dFd3ABD122FDB751702B7146#code

https://etherscan.io/address/0xA36972E347E538E6C7Afb9f44FB10DDa7BBa9BA2#code

https://etherscan.io/address/0xE5c405C5578d84c5231D3a9a29Ef4374423fA0c2#code

The goal of this audit is to review IDEX's solidity implementation for its decentralized exchange, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the IDEX team for specified versions. Whenever the code, software, materials, settings, enviroment etc is changed, the comments of this audit will no longer apply.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from offchain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Fairyproof to make sure we understand the size, scope, and functionality of the project's source code.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Fairyproof describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run the test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.

3. Best practices review, which is a review of the source code to improve maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

For this audit, we used the following sources of truth about how the decentralized exchange system should work:

https://idex.io/

project docs

These were considered the specification.

— Comments from Auditor

No vulnerabilities with critical, high or medium-severity were found in the above source code.

One vulnerability with low-severity was found in the above source code.

Additional notice: 0.

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

03. Introduction to IDEX 2.0

IDEX is the first Hybrid Liquidity DEX, combining a high-performance order book and matching engine with Automated Market Making (AMM). The platform blends the best of centralized and decentralized exchanges, with the performance and features of a traditional order book and the security and liquidity of an AMM. IDEX allows traders to get the best spreads, avoid failed transactions, and easily provide liquidity, all with the power of real limit and stop-loss orders.

04. Major functions of audited code

The audited code mainly implements the following functions:

- Deposits of Crypto Assets
    - Users can deposit crypto assets to IDEX
    - Users can trade on IDEX with their deposited assets
    - Users can withdraw their deposited assets
    - Two ways to withdraw assets
        - Users can submit their withdrawal applications from offchain and sign their applications with their private keys. After the admin receives an application for withdrawal from a user, he/she will charge no more than 20% of the total amount as a transaction fee and send the remaining amount to that user
        - Users can withdraw their assets onchain without being charged by the application
- Order Book Exchange
    - Verifying a user's signature for submitting an order from offchain
    - The admin collects, matches and executes sell and buy orders
    - The admin sets the transaction fee of a transaction and the transaction fee cannot exceed 20% of an order's total amount.
- The Exchange's contracts can be upgraded

05. Admin rights

In this application the admin has the following previlleges:

- Upgrading the Exchange's contracts
- Collecting and executing users' submitted orders
- Matching users' submitted orders

06. Key points in audit

During the audit Fairyproof mainly worked on the following items:

- Integer Overflow/Underflow

We checked all the code sections, which had arithmetic operations and might introduce integer overflow or underflow if no safe libraries were used.

We found an issue. For more details please refer to "10. Issue descriptions".

- Access Control

We checked each of the functions that can modify a state, especially those functions that can only be accessed by "owner".

We didn't find issues or risks in these functions or areas at the time of writing.

- Admin Rights

We checked whether or not the admin had inappropriate access rights.

We didn't find issues or risks in these functions or areas at the time of writing.

- State Update

We checked some key state variables which should only be set at initialization.

We didn't find issues or risks in these functions or areas at the time of writing.

- Asset Security

We checked whether or not the assets deposited in the contracts were safe and secure.

We didn't find issues or risks in these functions or areas at the time of writing.

- Offchain Signature Security

We checked whether or not the offchain signatures submitted for orders were safely and securely handled.

We didn't find issues or risks in these functions or areas at the time of writing.

07. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Re-entrancy Attack
- DDos Attack
- Integer Overflow
- Function Visibility
- Logic Vulnerability
- Uninitialized Storage Pointer
- Arithmetic Precision
- Tx.origin
- Shadow Variable
- Design Vulnerability
- Token Issurance
- Asset Security
- Access Control

08. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

09. List of issues by severity

A. Critical

- N/A

B. High

- N/A

C. Medium

- N/A

D. Low

- Integer Overflow/Underflow: Low

10. Issue descriptions

- Integer Overflow/Underflow: Low

Source and Description:

In line 129 of `AssetRegistry.sol`, the `loadAssetBySymbol()` function has the following code section:

```
function loadAssetBySymbol(...) internal view returns (Structs.Asset memory) {
  //...
    for (uint8 i = 0; i < self.assetsBySymbol[symbol].length; i++) {
      if (
        self.assetsBySymbol[symbol][i].confirmedTimestampInMs <= timestampInMs
      ) {
        asset = self.assetsBySymbol[symbol][i];
      }
    }
  }
```

The gas consumption will exceed the `gasLimit` in the following two scenarios and may be vulnerable to DOS attacks

- An integer overflow occurs to `i` and the loop becomes infinite.

  The index `i` in the `for` loop is an `uint8` variable and its max value is 255. The upper limit of the loop is determined by `self.assetsBySymbol[symbol].length`. if the upper limit is greater than 256, `i` will overflow and be reset to 0 and the loop will never end, thus causing `out of gas`.
- Before the `for` loop ends the gas consumption already exceeds the `gasLimit` thus causing `out of gas`, and the corresponding trade to be stuck.

`assetsBySymbol` holds different types of tokens who share the same name. If the number of types is greater than 256 it will cause issues. Therefore we marked this risk with a low-severity.

Recommendation:

Consider validating the number of types that `assetsBySymbol` holds is less than 256 prior to adding a new type of token whose name is the same as the tokens held in `assetsBySymbol`.

11. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- Defining Frequently Used Numbers As Constants

`msInOneSecond` is a constant number `1000`, which is used in both `getOneDayFromNowInMs` and `getCurrentTimestampInMs`.

Consider defining it as a constant.