Security Audit Report

OneDex liquidity-pools
smart contract

by ARDA

on May 31, 2023

Table of Content

Disclaimer

The report makes no statements or warranties, either expressed or implied, regarding the security of the code, the information herein or its usage. It also cannot be considered as a sufficient assessment regarding the utility, safety and bugfree status of the code, or any other statements.

This report does not constitute legal or investment advice. It is for informational purposes only and is provided on an "as-is" basis. You acknowledge that any use of this report and the information contained herein is at your own risk. The authors of this report shall not be liable to you or any third parties for any acts or omissions undertaken by you or any third parties based on the information contained herein.

Terminology

Inherent risk: A risk for users that comes from a behavior inherent to the smart contract design.

Inherent risks only represent the risks inherent to the smart contract design, which are a subset of all the possible risks. **No inherent risk doesn't mean no risk.** It only means that no risk inherent to the smart contract design has been identified. Other kind of risks could still be present. For example, the issues not fixed incur risks for the users, or the smart contracts deployed as upgradeable also incur risks for the users.

Issue: A behavior unexpected by the users or by the project, or a practice that increases the chances of unexpected behaviors to appear.

Critical issue: An issue intolerable for the users or the project, that must be addressed.

Major issue: An issue undesirable for the users or the project, that we strongly recommend to address.

Medium issue: An issue uncomfortable for the users or the project, that we recommend to address.

Minor issue: An issue imperceptible for the users or the project, that we advise to address for the overall project security.

Audit Summary

Scope of initial audit

- **Repository:** https://github.com/SynchronicSoftware/onedex-sc
- **Commit:** `2ebb9cf0eb8adf376918a56f4008673b2eb3b6ac`
- **Path to Smart contract:** `./onedex-sc/`

Scope of final audit

- **Repository:** https://github.com/SynchronicSoftware/onedex-sc
- **Commit:** `efa4a68f9389a96ba2215ad5531632d1250dd8ca`
- **Path to Smart contract:** `./onedex-sc/`

Report objectives

1. Reporting all **inherent risks** of the smart contract.
2. Reporting all **issues** in the smart contract **code**.
3. Reporting all **issues** in the smart contract **test**.
4. Reporting all **issues** in the **other** parts of the smart contract.
5. Proposing **recommendations** to address all issues reported.

2 inherent risks in the final commit

1 issue in the final commit

27 issues reported from the initial commit and 1 remaining in the final commit:

Severity	Reported			Remaining		
	Code	Test	Other	Code	Test	Other
Critical	3	0	0	0	0	0
Major	4	0	0	0	0	0
Medium	6	1	0	0	1	0
Minor	12	1	0	0	0	0

Inherent Risks

R1: A user might earn less by providing his tokens as liquidity than by simply holding them in his wallet.

This is because an “impermanent loss” occurs as soon as the current price of the pool differs from the initial price (at deposit time), but it can be counterbalanced by swap fees earned by liquidity providers.

What is impermanent loss? If we don’t take into account the swap fees, when a user buys tokens from the pool, the liquidity provider effectively sells his tokens at all intermediate prices from the initial price to the current price. From the perspective of the liquidity provider, this will be worst than holding all his tokens and selling them at the current price.

Computing the impermanent loss. In OneDex, given a price ratio `x` between the current price and the initial price, the impermanent loss is: `1 - 2 * sqrt(x) / (1 + x)`. From this, it can be seen that the bigger the price variation, the higher the impermanent loss: when `x = 1`, the impermanent loss is `0`, and as `x` departs from `1`, the impermanent loss gets closer to `1`, i.e. a 100% loss.

R2: A liquidity provider is very unlikely to withdraw his tokens in the same amounts as he initially deposited.

This is because:

- When a liquidity provider deposits tokens, the amounts must be provided according to the ratio of the two token’s reserves of the pool. In return, the liquidity provider now owns a share of the pool, proportional to the amounts of deposited tokens.
- When users swap, the ratio of the pool’s reserves changes.

- When the liquidity provider withdraws his share, tokens are taken from the pool according to the ratio of the pool's reserves, which might differ from the initial ratio due to users' swaps.

Code Issues & Recommendations

Since the smart contract code is not open-source, only the remaining issues are published.

Test Issues & Recommendations

Since the smart contract code is not open-source, only the remaining issues are published.

T1: Missing tests for swaps across several pairs

Severity: Medium **Status:** Won't fix

Description

There is no tests for the endpoints `swap_multi_tokens_fixed_input` and `swap_multi_tokens_fixed_output` when multiple consecutive swaps occur, i.e. the path of tokens `path_args` has length 3 or greater.

Recommendation

We suggest adding tests for these endpoints with `path_args` of length 3.