Security Audit Report

AshSwap pool-v2
MultiversX smart contract

by ARDA

on December 15, 2023

Table of Contents

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

Code: The code with which users interact.

Inherent risk: A risk for users that comes from a behavior inherent to the code's design.

Inherent risks only represent the risks inherent to the code's 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 code's 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 upgradability of the code might 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.

Objective

Our objective is to share everything we have found that would help assessing and improving the safety of the code:

1. The **inherent risks** of the code, labelled R1, R2, etc.
2. The **issues** in the **code**, labelled C1, C2, etc.
3. The **issues** in the **testing** of the code, labelled T1, T2, etc.
4. The **issues** in the **other** parts related to the code, labelled O1, O2, etc.
5. The **recommendations** to address each issue.

Audit Summary

Initial scope

- **Repository:** https://github.com/ashswap/ash-exchange-sc
- **Commit:** `afe0c57d0662974de954909cab791aca14b950db`
- **MultiversX smart contract path:** ./contracts-0.39.8/dex/pool_v2/

Final scope

- **Repository:** https://github.com/ashswap/ash-exchange-sc
- **Commit:** `e3ff87288b2fd26dea8edd7c7b02a07cd81d19e6`
- **MultiversX smart contract path:** ./contracts-0.39.8/dex/pool_v2/

2 inherent risks in the final scope

0 issue in the final scope

16 issues reported in the initial scope and 0 remaining in the final scope:

Severity	Reported			Remaining		
	Code	Test	Other	Code	Test	Other
Critical	1	0	0	0	0	0
Major	1	0	0	0	0	0
Medium	5	1	0	0	0	0
Minor	8	0	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.

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 in the pool, in return he now owns a share of the pool. In case of a *balanced deposit*, i.e. the tokens are provided according to the ratio of the two token’s reserves of the pool, then the liquidity provider’s share equals the amount of any deposited token relative to the reserve of the pool in that token. Otherwise, in case of an *imbalanced deposit*, the liquidity provider’s share is calculated differently.
- 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. However, even if the ratio of the pool’s reserves equals the initial ratio, if the user had made an imbalanced deposit, then he might receive tokens in different amounts than he initially deposited, because of the different calculation of his share of the pool.

Code Issues & Recommendations

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

Test Issues & Recommendations

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