OpenZeppelin

1inch Liquidity Protocol Audit

OPENZEPPELIN SECURITY | JANUARY 29, 2021

Security Audits

Mooniswap is a constant-product AMM created by 1Inch that uses virtual balances to reduce liquidity providers' losses to arbitrageurs.

The 1inch team asked us to review the code for the 1inch Liquidity Protocol which is the upgraded and rebranded version of Mooniswap. We reviewed the code and here we publish our findings.

Summary

Overall, we are happy with the structure and implementation of the code and found it to be very efficient. Our main recommendation is to introduce thorough documentation and code comments to better communicate the expected behavior.

Scope

We audited commit `b9c06335fb1d68b19054442547fc677f42795b44` of the `1inch-exchange/1inch-liquidity-protocol` repo.

The following files were in scope:

- Mooniswap.sol
- MooniswapDeployer.sol
- MooniswapFactory.sol
- governance/ExchangeGovernance.sol

- inch/GovernanceMothership.sol
- interfaces/IMooniswapFactoryGovernance.sol
- interfaces/IExchangeGovernance.sol
- interfaces/IFeeCollector.sol
- interfaces/IMooniswapDeployer.sol
- interfaces/IMooniswapFactory.sol
- libraries/ExchangeConstants.sol
- libraries/ExplicitLiquidVoting.sol
- libraries/LiquidVoting.sol
- libraries/SafeCast.sol
- libraries/Sqrt.sol
- libraries/UniERC20.sol
- libraries/VirtualBalance.sol
- libraries/VirtualVote.sol
- libraries/Voting.sol
- utils/BalanceAccounting.sol

The following files were out of scope:

- ReferralFeeReceiver.sol
- interfaces/IReferralFeeReceiver.sol
- governance/GovernanceFeeReceiver.sol
- inch/farming/FarmingRewards.sol
- governance/GovernanceRewards.sol
- utils/BaseRewards.sol
- utils/Converter.sol
- everything in the mocks directory

System Overview

The system behaves similarly to other popular AMMs, but with the exception that the prices used when swapping do not change instantaneously after a swap. Instead, after a swap, the prices used for swaps change gradually over a period of time (called the `decayPeriod`). The idea is that, after a large swap, this gradual price adjustment should result in arbitrage happening at a price

The `decayPeriod`, along with various fees, can be set by a governance module (within certain bounds). This governance module is controlled by 1Inch token holders. Voting happens in a continuous fashion, and the values adjust gradually towards the new values in much the same way as the AMM swap prices do after a swap.

Privileged Roles

In the constructor function of the `MooniswapFactory` contract, a `poolOwner` is set (this is `immutable`).
When a new `Mooniswap` pair contract is deployed via the `MooniswapFactory.deploy` function, the `MooniswapDeployer` contract sets the `poolOwner` as the `owner` of the new `Mooniswap` pair contract.

This `owner` of the `Mooniswap` pair contract has the ability to:

1. Remove excess tokens from the `Mooniswap` pair contract via the `rescueFunds` function.
2. Set the `mooniswapFactoryGovernance` value for the pair via the `MooniswapGovernance.setMooniswapFactoryGovernance` function — which in turn allows the `owner` to:
3. Set various default fees — including the max fees.
4. Set the decay period, share parameters, and `feeCollector` addresses.
5. Shutdown the `Mooniswap` contract via the `shutdown` function, which disallows swaps.

So it is important that the `poolOwner` value that is set in the `constructor` function of the `MooniswapFactory` contract be fully trusted.

Here we present our findings.

Critical

None. 🙂

High

Medium

Mooniswap pairs cannot be unpaused

The `MooniswapFactoryGovernance` contract has a `shutdown` function that can be used to pause the contract and prevent any future swaps. However there is no function to unpause the contract. There is also no way for the factory contract to redeploy a `Mooniswap` instance for a given pair of tokens. Therefore, if a `Mooniswap` contract is ever shutdown/paused, it will not be possible for that pair of tokens to ever be traded on the Mooniswap platform again, unless a new factory contract is deployed.

Consider providing a way for `Mooniswap` contracts to be unpaused.

Low

Fee Collectors can not be removed

The `MooniswapFactoryGovernance` contract has an `isFeeCollector` mapping. Fee collectors can be added by the `owner`, but there is no way for the `owner` to remove fee collectors.

Consider providing a way for te `owner` to remove fee collectors.

Notes

Use of named return variables and implicit returns

Some functions (for example, the `Mooniswap.depositFor` function) use named return values and implicitly return those values. Consider removing all named return variables and explicitly returning all return values using a `return` statement. This should improve both explicitness and readability of the code, and will make the code less susceptible to regressions during future updates.

Mooniswap constructor does not enforce token order

However, the `Mooniswap` pair contract itself does not enforce the correct ordering in its constructor function. If the order is not correct, then the `Mooniswap._getReturn` function will not behave correctly.

Since the correctness of the ordering is crucial for the `Mooniswap` contract to operate correctly, consider having the `Mooniswap` contract's constructor function enforce the correct token ordering. That way the correctness of the `Mooniswap` contract will be independent of the factory contract that deploys it.

Inconsistent token identifiers

The `MooniswapFactory` contract identifies the `Mooniswap` pair's tokens as `token1` and `token2`, whereas the `Mooniswap` contract itself identifies its tokens as `token0` and `token1`. Consider making these token identifiers consistent.

Missing docstrings

The contracts and functions in the reviewed code base lack documentation. This hinders reviewers' understanding of the code's intention, which is fundamental to correctly assess not only security, but also correctness. Additionally, docstrings improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned and the events emitted.

Consider thoroughly documenting all functions (and their parameters) that are part of the contracts' public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Conclusions

No critical or high severity issues were found. Some changes were proposed to follow best practices, and reduce the potential attack surface.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits

OpenZeppelin

Defender Platform

- Secure Code & Audit
- Secure Deploy
- Threat Monitoring
- Incident Response
- Operation and Automation

Services

- Smart Contract Security Audit
- Incident Response
- Zero Knowledge Proof Practice

Learn

- Docs
- Ethernaut CTF
- Blog

Company

Contracts Library

Docs

OpenZeppelin