KyberPool Security Recheck

This report is public.

Published: August 11, 2020

Abstract

In this report, we consider the security of the KyberPool project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, security audit is not an investment advice.

Summary

In this report, we considered the security of KyberPool smart contracts. We performed our audit according to the procedure described below.

The initial audit showed several issues of low severity. In the latest version of the code, most of these issues were fixed. None of the issues found during the audit endanger security of the project.

General recommendations

All the issues found during the initial audit were either fixed in the latest version of the code or commented by the developers. Overall project quality is high. Thus, we do not have any further recommendations.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- manual audit
  - we manually analyze code base for security vulnerabilities
  - we assess overall project structure and quality
- report
  - we reflect all the gathered information in the report

Project overview

Project description

In our analysis we consider smart contracts of KyberPool project on Git repository, commit 4f15de9147521a23c92ae43ec2b000cc37a45973.

Latest version of the code

After the initial audit, developers updated the code to the latest version (Git repository, commit 5668b9af3407d573687d7d8b232473b838179b22).

Project architecture

For the audit, we were provided with a git repository. The project has tests and documentation.

The total LOC of audited sources is 586.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger smart contracts security. We highly recommend fixing them.

The audit showed no critical issues.

Medium severity issues

Medium issues can influence project operation in current implementation. We highly recommend addressing them.

The audit showed no issues of medium severity.

Low severity issues

Low severity issues can influence project operation in future versions of code. We recommend taking them into account.

Gas consumption (fixed)

KyberPoolMaster.sol:

- Lines 38-46: it is possible to combine two mappings into one `epoch -> feeHandler -> Struct`, where `Struct` contains `bool, uint, uint`. Moreover, one can use `uint248` to pack `bool + uint + uint` and thus, reduce the number of used storage slots.
- Line 333: `delegationFees` array can grow significantly if the PoolMaster changes fee frequently. Consider implementing binary search in `getEpochDFeeDataId()`.
- Line 466: `_epoch` to `delegationFees` computational complexity is currently $O(n^2)$. We recommend either
    - perform this calculation off-chain and only check the correctness on-chain or
    - require `_epochGroup` to be sorted, so that you can perform binary search to find the first occurrence of `DFeeData` and then move through the rest, in this case the complexity will be $O(n)$.

The issues have been fixed and are not present in the latest version of the code.

Code style (fixed)

KyberPoolMaster.sol:

- Lines 354, 372: consider returning `DFeeData` struct instead (requires `ABIEncoderV2`).
- Line 393: `getUnclaimedRewards` should be view.
- Line 539: `applyFee(epochDFee);` should be called right after `epochDFee` declaration.
- Lines 521-536 and 765-775: `tokensWithRewards`/`findIndex` algorithm is suboptimal. Consider filling `accruedByToken` array for all `feeHandler`s instead and skip those with zero rewards when calling `sendTokens`.

The issues have been fixed and are not present in the latest version of the code.

Bad design (partially fixed)

- The contract's most costly operations are `getAllEpochWithUnclaimedRewards() -> claimRewardsMaster()` and `_getAllEpochWithUnclaimedRewardsMember() -> _claimRewardsMember()` pairs of calls. These actions are difficult to design, which resulted in poor code quality.
  If the system does not have too many `feeHolder`s, redesigning these operations as `feeHolder`-dependant functions might improve the performance.

  The issue has been fixed and is not present in the latest version of the code.

- Line 407: `rewardsPerEpoch` function is not present in `IKyberFeeHandler` interface, but only in the `KyberFeeHandler` contract, so you rely on implementation rather than abstraction.

  > *Comment from developers:* *`RewardsPerEpoch` is a must to design the pool master code. Thus, we will have to keep supporting it. Please do use it as part of Kyber interface.*

- Line 606: there is an inconvenience in `getUnclaimedRewardsMember` logic. If `claimRewardsMaster` function has not been called for some epoch yet, the `getUnclaimedRewardsMember` function will ignore it. If a user calls the second variant of `getAllEpochWithUnclaimedRewardsMember` function (the one with `_fromEpoch`, `_toEpoch` arguments), rewards for some epochs might be missed.

  > *Comment from developers: this is expected behavior. A member can only claim its rewards for an epoch which the pool already claimed successfully (at least for 1 fee Handler).*

This analysis was performed by:

Evgeny Marchenko
Boris Nikashin
Alexander Seleznev

August 11, 2020