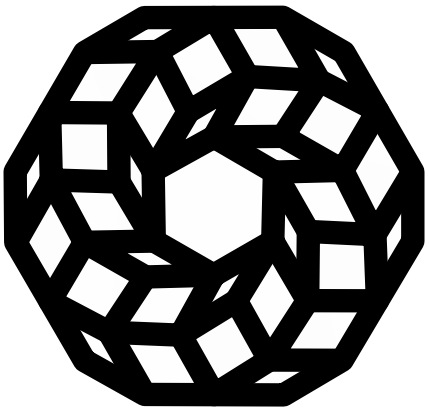

PowerPool PowerIndex Routers Security Analysis

by Pessimistic

This report is public.

Published: April 30, 2021

Abstract

In this report, we consider the security of smart contracts of the PowerPool project. Our task is to find and describe security issues in 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 smart contracts of PowerPool project. We performed our audit according to the procedure described below.

The analysis showed two issues of medium severity (code logic issue and possible misuse of `setUsdcYfiSwapPath()` function) and several low severity issues.

After the initial audit, the code base was updated. All the mentioned issues were either fixed or commented. Also, new functionality was added to the scope of the audit. However, new issues were found, including No slippage protection and several code quality issues.

After the recheck, another update was made. This update includes new functionality. The audit of the updated code did not reveal any issues.

General recommendations

We recommend using automated code analyzers as a part of development process and paying extra attention to implementation of third-party protocols when integrating routers with them.

Project overview

Project description

In our analysis we consider PowerIndex smart contracts of PowerPool project on GitHub repository, commit 758f6bf698e6a2bc5ba1602907ab29b7bb10aba8.

The scope of the audit included the analysis of contracts in **contracts/powerindex-router/**:

- **WrappedPiErc20.sol**
- **PowerIndexNaiveRouter.sol**
- **PowerIndexBasicRouter.sol**
- **implementations/AavePowerIndexRouter.sol**
- **implementations/YearnPowerIndexRouter.sol**

For the audit, we were provided with the specification on GitHub repository, commit 0cdc51937f8da3750ba3d586a8cb18ac01ad9c2d.

The total LOC of audited sources is 685.

Code base update #1

After the initial audit, the code base was updated. For the recheck, we were provided with commit 709cf2940ef54c1d7e987dcc31db39d51f166776.

Code base update #2

After the recheck, new functionality was added to the project. For the next recheck, we were provided with commit 5f4b6f30010fecc2c82d23a41f6c5827cec5e88d.

Procedure

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

1. Whether code logic corresponds to the specification.
2. Whether the code is secure.
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
  - We scan project's code base with automated tool SmartCheck.
  - We manually verify (reject or confirm) all the issues found by tools.
- Manual audit
  - We inspect the specification and check whether the logic of smart contracts is consistent with it.
  - 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.

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.

Code logic (fixed)

In **AavePowerIndexRouter** contract, `unstakeWindow` and `rebalancingInterval` variables are independent. Thus, it is possible that `unstakeWindow` interval is shorter than `rebalancingInterval`. In this case, there can be a situation when `unstakeWindow` has expired and the contract is unable to redeem tokens.

We recommend adding the check that `rebalancingInterval` value is shorter than `unstakeWindow`.

> *Comment from developers: we'd like to do not check `rebalancingInterval` values during the staking address assignment. In this case, there is still a chance when `UNSTAKE_WINDOW` will be lower than the currently assigned `rebalancingInterval`. We need an option to disable staking anytime by assigning its address to one of the non-contract addresses.*

In this case, we recommend modifying **AavePowerIndexRouter** contract:

- Adding `staking == address(0)` check to `require()` statement in `setReserveConfig()` function.
- Adding `_basicConfig.staking == address(0)` check to `constructor()`.

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

setUsdcYfiSwapPath misuse possibility

In **YearnPowerInderRouter** contract, `setUsdcYfiSwapPath()` function at line 170 can be used improperly. Passing an unsuitable path to the function can result in overpriced token swaps and significant losses due to slippage.

We recommend checking the path thoroughly and implementing a proper key management system to minimize the chance of malicious use of this function.

> *Comment from developers: only the Owner of the contract can call this method. Currently, we assigned the Owner to the team's multisig. We will transfer ownership to the protocol governance contract in the future.*

No slippage protection

In **YearnPowerInderRouter** contract, `distributeRewards()` function has no slippage protection and thus is susceptible to sandwich attacks at lines 115 and 122. This can result in significant losses: the Uniswap's rate can be manipulated so that **YearnPowerInderRouter** contract will receive much less YFI tokens than expected.

We recommend adding slippage protection by allowing a user to explicitly specify the minimal amount of tokens that he/she expects to receive.

> *Comment from developers: At the moment, we estimate possible losses from such attacks as minimal, taking into account the volume of commissions and liquidity. If the losses exceed the allowable ones, we will add the necessary checks and deploy new version.*

Low severity issues

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

Code quality

- In **PowerIndexBasicRouter** contract, `rewardPools` should not be declared as `public` as it has its own getter.

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

- In **AavePowerIndexRouter** contract, there is a typo at line 58: should be `NOTHING` instead of `NOTING`.

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

- In **WrappedPiErc20** contract, the default case in assembly block at lines 155–162 can be replaced with `assembly { revert(add(data, 32), size) }`.

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

- In **PowerIndexBasicRouter** contract, consider adding checks to the constructor:
  - `_basicConfig.reserveRatio` must be less than `HUNDRED_PCT`
  - `_basicConfig.pvp` and `_basicConfig.poolRestrictions` must have non-zero values.

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

- In `_redeem()` function of **AavePowerIndexRouter** contract, `approve()` call is redundant.

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

- In **YearnPowerIndexRouter** contract, `setUsdcYfiSwapPath()` function should have the following checks to decrease the chances of an incorrect call and make sure that USDC will be converted to YFI:
  - `_usdcYfiSwapPath[0] == USDC`
  - `_usdcYfiSwapPath[usdcYfiSwapPath.length - 1] == YFI`

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

- In **PowerIndexWrappedController** contract, there is a typo in the natspec for `createPiToken()` and `replacePoolTokenWithNewPiToken()` functions: there should be `_symbol` instead of `_name` at lines 80 and 98 in the code after update #1.

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

- In `_distributePiRemainderToPools()` function of **PowerIndexBasicRouter** contract, the calculation of `poolReward` is inaccurate due to rounding. As a result, the sum of `poolReward[i]` values may differ from `piBalanceToDistribute` value.

  We recommend calculating the last `poolReward[i]` value as a diff between `piBalanceToDistribute` and the sum of all previous `poolReward[i]` values.

> *Comment from developers: the rounding leftovers are negligible and we are ok to keep them for the future distribution round.*

Code logic

In **YearnPowerIndexRouter** contract, `remove_liquidity_one_coin()` function is called with a constant as a second parameter. However, there are cases when another value is preferrable.

We recommend passing a variable, that can be changed according to the situation, as a second parameter.

> *Comment from developers: the idea is to unwrap `yCrv` into `USDC`, which resides in the slot #1 of `YDeposit(0xbbc81d23ea2c3ec7e56d39296f0cbb648873a5d3).coins()`. There is no intention to swap it into something other than `USDC`.*

Using tokens other than `USDC` when calling `remove_liquidity_one_coin()` function may be more profitable.

> *Comment from developers: we will implement this feature in future versions.*

Locked Ether

PowerIndexBasicRouter contract can receive ether. However, there are no methods to withdraw ether from the contract. Thus, all the ether sent to this contract will be stuck.

We recommend adding withdraw functionality.

> *Comment from developers: we will upgrade the **PowerIndexBasicRouter** to provide ETH refunds, so no withdrawal methods are required.*

Notes

Costly loop

In **WrappedPiErc20** contract, the loop at lines 125–127 can run out of gas.

In such a case, we recommend splitting the call into several transactions.

> *Comment from developers: this method is useful when the caller needs to execute several actions atomically within a single transaction. It is up to the caller to ensure that it will not run out of gas.*

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Igor Sobolev, Security Engineer
Boris Nikashin, Analyst
Alexander Seleznev, Founder

April 30, 2021