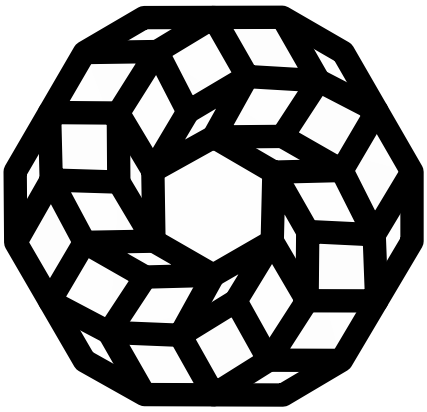

PowerPool PowerIndex Security Analysis

by Pessimistic

This report is public.

Published: December 5, 2020

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 initial audit showed a critical bug, three issues of medium severity, including an overpowered owner issue, a discrepancy with the documentation, and unrestricted access. Also, several issues of low severity were found.

After the audit, the developers updated the code to the latest version with updates and fixes. They also provided comments for the rest of issues. The bug was fixed, also developers specified that Owner accesses the contracts via multisig. Thus, we moved this issue to Notes section.

The project does not have general documentation.

The logic of the project is overcomplicated and hard to analyze without documentation. However, the code quality of the implementation is good.

We interacted with developers during the analysis to clarify certain solutions and implementation decisions. Developers are aware of several logical issues and plan to address them.

General recommendations

We recommend fixing all the issues and adding documentation to the project.

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 tools: MythX and 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.

Project overview

Project description

In our analysis we consider PowerIndex smart contracts of PowerPool project on GitHub repository, commit 0c8b3f509621535e0c4eb1f1b5fb77367be430d5.

The scope of the audit at the first stage included the analysis of following contracts:

- **lib/DelegatableCheckpoints.sol**
- **powerindex-mining/DelegatableVotes.sol**
- **powerindex-mining/VestedLPMining.sol**
- **PowerIndexPoolController.sol**
- **powerindex-router/PowerIndexWrappedController.sol**
- **PowerIndexAbstractController.sol**

The project has no documentation. We were only provided with description for vesting algorithms: vesting_math.pdf, sha1sum a75f9fa2ae993d5f007eacd376ebd0d0fcaef4d2.

The total LOC of audited sources is 848.

Latest version of the code

After the initial audit, the developers made several pull requests. We reviewed the changes in these requests. After this, the requests were merged to `master` branch, commit 2d9048a97b42c030a4d71144cdcc44cf1bf938ed.

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.

Bug (fixed)

There is a bug in **powerindex-mining/VestedLPMining** contract at line 316:

```
314 if (user.pendedCvp > 0) {
315   // TODO: Make user.pendedCvp be updated as of the pool'
    lastUpdateBlock
316   if (user.pendedCvp > cvpVestingPool) {
317     cvpVestingPool = cvpVestingPool.sub(user.pendedCvp);
318   } else {
319     cvpVestingPool = 0;
320   }
321 }
```

If `user.pendedCvp > cvpVestingPool`, the transaction will revert. This can result in a situation when a user is unable to withdraw funds with `emergencyWithdraw()` function.

> *Comment from developers: There is indeed a logic error in the `if` statement: instead of `user.pendedCvp < cvpVestingPool`, the code contains `user.pendedCvp > cvpVestingPool`. However, this expression can never become `true`. On the other hand, resetting `cvpVestingPool` does not revert other functions calls. So, no matter if the code includes this expression or not, it never reverts the function call. Therefore, we believe this error shall not be assigned a "critical" severity. Please note, the PR https://github.com/powerpool-finance/powerindex/pull/70 fixes this error.*

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

Medium severity issues

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

Discrepancy with the documentation

The provided document vesting_math.pdf describes a specific schedule for rewards vesting. However, **powerindex-mining/VestedLPMining** contract implements a simplified version of this schedule. This might result in slight deviations of claimable reward amounts.

We recommend updating this part of the documentation so that it describes real behavior of the code clearly.

> *Comment from developers: The dynamic development of the project imposes certain restrictions on the quality and relevance of documentation. We will definitely fix this.*

Unrestricted access

`unbindNotActualToken()` function of **PowerIndexPoolController** contract should have `onlyOwner` modifier.

> *Comment from developers: We designed the `unbindNotActualToken()` method to remove a token from the pool when its weight and balance are almost `0` without any permissions. Since the Owner of the **PowerIndexPoolController** contract is **PPGovernorL1** and it starts the process of changing the token weights, there is no point in additional voting on the formal removal of the token, the weight and balance of which are close to zero.*
> *The rest of the balance is transferred to the community treasury called **PermanentVotingPower**. Thus, the community has an economic incentive to carry out this operation.*

Low severity issues

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

Missing input validation

`_poolType` argument is passed to `set()` and `add()` functions of **powerindex-mining/VestedLPMining** contract. This argument can take only two values. However, it is not checked to have a valid value.

We recommend using `emum` type for this argument or adding a proper check in these functions.

> *Comment from developers: We intentionally did that so that the Owner (**PPGovernanceL1** contract) can add new pool types, and the contract code will handle them accordingly. Using an `enum` type does not allow this. In any case, only the Owner calls these methods.*

Code quality

- In `_doCheckpointVotes()` function of **powerindex-mining/VestedLPMining** contract at lines 370–372 and 382–383, `lpCvp` and `userLpCvp` values calculation is overcomplicated:

  ```
  d' = a * scale / c
  d = d' * b / scale
  ```

  However, it can be simplified to `d = a * b / c`. This will improve code readability, decrease gas consumption, and increase calculation precision.

  > *Comment from developers: In our opinion, we cannot do this without loss of accuracy.*

- Function `callPool()` of **PowerIndexAbstractController** contract can be used to send ether to the pool. However, pool controller should not have any ether, also pools do not have `payable` functions.
  We recommend removing `{ value: value }` part from the code at line 35.

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

- In **DelegatableVotes** contract, there should be `vote` instead of `voice` in comments at lines 72, 94, and 120.

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

- There is `TODO` in **powerindex-mining/VestedLPMining** contract at line 315.

  > *Comment from developers: This `TODO` refers to the fact that the sum of `user.pendedCvp` for all users and `cvpVestingPool` should be equal, but in reality, not equal. This error occurs because `user.pendedCvp` is calculated at the time of user action, and `cvpVestingPool` when the pool is updated.*

> *In the current implementation, this error does not affect anything. But perhaps in the next version, we will need this, which is why TODO is present.*

Notes

Overpowered owner

The owner of **powerindex-mining/VestedLPMining** contract can manipulate rewards calculation and even nullify them. Also, token reservoir can stop releasing reward tokens at any moment.

The system depends heavily on the owner. Therefore, there are scenarios that may lead to undesirable consequences, e.g. if the owner's private keys become compromised.

We recommend designing contracts in a trustless manner or implementing a proper key management system, e.g. multisig.

> *Comment from developers: We designed the **VestedLPMining** contract to manage the size of rewards flexibly according to CVP holders' community decisions based on CVP market price, TVL, and other parameters. The CVP community can change `cvpPerBlock`, `cvpVestingPeriodInBlocks` and other parameters and even upgrade contract.*
>
> *We designed The **Reservoir** contract to store CVP tokens and distribute them through different contracts, like **VestedLPMining** or **PowerOracle**. These contracts are subject to change or deactivation due to CVP holders' community decisions. The community can decide to deprecate one **VestedLPMining** and set its allowance to `0` and set allowance to new **VestedLPMining**.*
>
> *The **PPGovernorL1** contract will perform all these operations as an Owner of **VestedLPMining** and **Reservoir**.*
>
> *While active development is taking place, the Owner of the contracts is Team's Gnosis Safe `0xB258302C3f209491d6041655490796807085 81Cc`.*

This analysis was performed by Pessimistic:

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

December 5, 2020