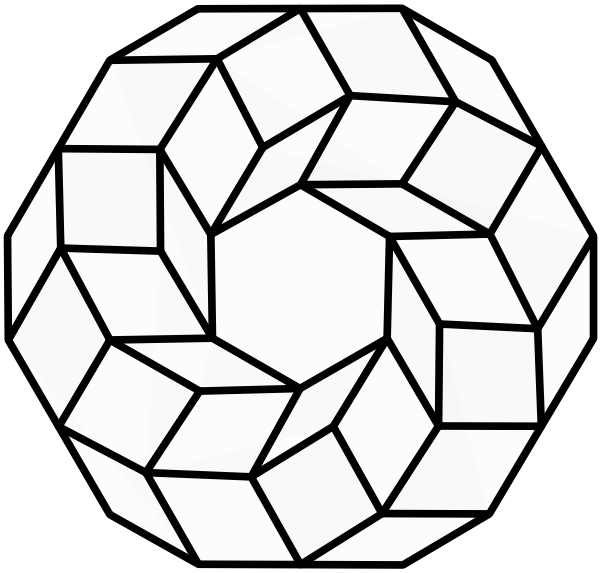

PowerPool Balancer Connectors Security Analysis

by Pessimistic

This report is public

May 29, 2022

Abstract

In this report, we consider the security of smart contracts of PowerPool 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. A single 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, a security audit is not investment advice.

Summary

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

The initial audit showed two issues of medium severity: ERC20 standard violation and Lack of slippage protection. Also, several low-severity issues were found.

The codebase is of average quality and mainly suffers from suboptimal design and overcomplicated code structure. Besides, the project lacks documentation.

The developers updated the code, which fixed ERC20 standard violation and most of low severity issues. No new issues were discovered during the recheck.

General recommendations

We recommend fixing the mentioned issues and improving the project documentation. We also recommend redesigning connectors and simplifying the overall code structure.

Project overview

Project description

For the audit, we were provided with PowerPool Vaults project on a public GitHub repository, commit b5a83fe14889a64c7a75925e5b5d02313580436d. The scope of the audit included the following files and their dependencies:

- `AssetManager.sol`
- `connectors/BProtocolPowerIndexConnector.sol`
- `connectors/BalPowerIndexConnector.sol`

The total LOC of audited sources is 1221.

The project has README.md file with a short description of the project. The code has some NatSpec comments. However, there is no detailed documentation available.

117 out of 117 tests pass, the code coverage is 76.37%.

Codebase update

After the initial audit, the codebase was updated. For the recheck, we were provided with commit 01f14269b93d851113d6dba695764f602f5eb39c.

The developers fixed or commented most of known issues. They also introduced emergency withdraw feature. The audit did not discover any issues within new functionality.

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:

- Automated analysis
  - We scan the project's codebase with the automated tools: Slither and SmartCheck.
  - We manually verify (reject or confirm) all the issues found by the tools.
- Manual audit
  - We manually analyze the codebase for security vulnerabilities.
  - We assess the 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 project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

The audit showed no critical issues.

Medium severity issues

Medium issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. ERC20 standard violation (fixed)

ERC-20 standard states :

```
Callers MUST handle false from returns (bool success). Callers MUST
NOT assume that false is never returned!
```

However, returned values of `transfer` calls are not checked in the following cases:

- In `_transferFeeToReceiver` function of **BProtocolPowerIndexConnector** and **BalPowerIndexConnector** contracts at lines 86 and 88 respectively.
- In `redeem` of **BProtocolPowerIndexConnector** contract at line 166.

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

M02. Lack of slippage protection

Both connectors collect rewards and change them to underlying tokens. Neither of them has any slippage protection, and claim rewards operation may yield less profit than expected.

Besides, the swap logic of **BProtocolPowerIndexConnector** is implemented within `swapByMiddleWeth` function of **UniswapV3OracleHelper** library and might be later reused for other scenarios. We recommend introducing slippage protection to this library, e.g., minimum expected return parameter.

> *Comment from the developers: It's impossible to check due to automatic operations, doing by poke operators, but amounts to swap controls by router owner, so owner can use small amounts to reinvest to avoid sandwich-attack risks.*

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Code quality (fixed)

Asset Manager calls `claimRewards` function of **BProtocolPowerIndexConnector** contract via delegatecall. Therefore, `address(this)` points to the Asset Manager contract. Consider using it instead of `ASSET_MANAGER` constant since it is marginally more in line with Solidity best practices.

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

L02. Unused variable (fixed)

`RATIO_CONSTANT` constant is from line 19 of **BalPowerIndexConnector** contract is never used within the smart contract system.

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

L03. Missing check

`isReadyToClaim` function of **BalPowerIndexConnector** contract doesn't check if claim params have been decoded successfully. As a result, this function approves the claim for any incorrect claim params value.

> *Comment from the developers: Parameters sets by admins, so they must check inputs carefully. It can't be done in smart contract due to values variativity.*

L04. Poor error handling

`unpackStakeData`, `unpackClaimParams`, and `unpackStakeParams` functions of **BalPowerIndexConnector** contract have limited ability to report an error. We recommend at least clearly describing this behavior with NatSpec comments.

L05. Function visibility (fixed)

Consider declaring functions as `external` instead of `public` when possible to improve code readability and optimize gas consumption. Multiple functions within the scope of the audit could be improved this way.

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

L06. Checks-Effects-Interactions pattern violation

`_pokeFrom` function of **AbstractPowerIndexRouter** contract violates CEI pattern. Consider rearranging the logic of the function and always following CEI pattern.

> *Comment from the developers: The sequence of lines of code here plays an important role, since the state is first changed through the _rebalance method, and then a check is performed using the state variable.*

L07. Implicit return values (fixed)

`stake` and `redeem` functions of **BalPowerIndexConnector** contract return implicit return values. We recommend always assigning return values explicitly.

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

Notes

N01. Controversial connector design

AssetManager contract utilizes connectors to implement protocol-specific functionality. Besides, this design introduces upgradability to the system. Both goals are achieved naturally via the `delegatecall` feature of the EVM, and the AssetManager-Connector pair works similarly to the Proxy-Logic pattern. However, BProtocol and Balancer connectors also behave as independent entities: they include view functions that facilitate interaction with corresponding protocols. Besides, `BalPowerIndexConnector` directly interacts with Balancer on its own (see `initRouter` function). We find this design of the connectors confusing. Normally, Solidity contracts are utilized either directly or via delegatecall, but not both. We recommend extracting view functionality into a separate contract that is called via a regular `call`.

> *Comment from the developers: BalPowerIndexConnector does not directly interacts with any contracts in initRouter function. Connector provides initialization logic for router, and it can by executed by initRouterByConnector router method. Connectors does not have any permissions except REWARDS_MINTER approval. We did it on purpose, due to architectural limitation in curve staking contract. It's do not allow to get the pending rewards amount on-chain without doing the claim transaction. But we have to provide getPendingRewards method to PowerPool clients, so we were forced to add setMinterApproval execution to initRouter method to make getPendingRewards method work. This function should be manually changed to "view" in the clients ABI, but it can be executed by transaction, so claimed amount will sent to AssetManager balance.*

N02. Complicated code structure

The project includes connectors to multiple protocols deployed across different chains. Some of them have very little in common, and their implementations work differently. However, they share pieces of the code via inheritance, which adds unnecessary coupling between otherwise independent modules. Besides, inheritance complicates contract structure and negatively impacts readability. Consider refactoring connectors into independent implementations that all share the same interface.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Ivan Gladkikh, Junior Security Engineer
Irina Vikhareva, Project Manager

May 29, 2022