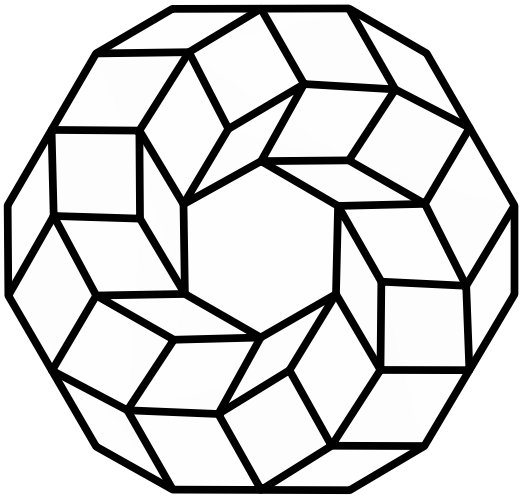

PowerPool Security Analysis

by Pessimistic

This report is public.

Published: June 28, 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 initial audit showed a few issues of medium severity, including Overpowered owner, and ERC20 standard violation. Also, several code quality issues of low severity were found.

After the initial audit, the project team discovered an issue in the logic of the project, so the contracts were updated. The recheck of the fixes did not reveal any security issues.

After the recheck #1, update #2 was performed to improve integration with Curve protocol. In this update, the documentation was added to the project. However, it does not describe all the technical details of the overly complicated implementation.

After the recheck #2, the contracts were run on a testnet, which revealed new logic issues. These issues were fixed with update #3.

General recommendations

We recommend fixing the mentioned issues and adding NatSpecs to the project and improving the documentation. We also recommend simplifying the logic of the code and making it more modular.

Project overview

Project description

For the audit, we were provided with PowerPool project on GitHub repository, commit 55fe1316991c8f643e710473a49dfdc5cd2664eb.

The scope of the audit includes only the following contracts:

- **weight-strategies/YearnVaultInstantRebindStrategy.sol**
- **weight-strategies/blocks/YearnFeeRefund.sol**
- **weight-strategies/blocks/SinglePoolManagement.sol**
- **weight-strategies/WeightValueAbstract.sol**

All the tests pass successfully.

The total LOC of audited sources is 574.

Update #1

After the initial audit, the code base was updated. For the recheck, we were provided with pull request #139, commit d1016c50f70c6aa06662b57a5036aa80a22fbfb2.

The recheck #1 was performed within the scope mentioned in the project description.

Update #2

After the recheck #1, another update of the code was performed. For the recheck #2, we were provided with commit e03ad2d2994a390c464593465f54e7852639e35e.

The recheck #2 was performed within the scope mentioned in the project description.

The tests coverage was improved, and new tests reveal a bug in rate logic. The bug was fixed in this version of the code.

Also, the documentation was added to the project in powerpool-docs repository, commit 15ed27d7e06eb798142be7edea2cd5abc8d40f0c.

Update #3

After the recheck #2, one more update was made. For the recheck #3, we were provided with commit 5a56d3a36f996b99cb31321133ad4dd754a00f7f.

This update includes the fixes of logic issues found during inspections on a testnet.

Several code quality issues found during this recheck were reflected in a review on GitHub.

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 project's code base with automated tool SmartCheck. Slither failed to run on this project.
  - We manually verify (reject or confirm) all the issues found by tools.
- 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.

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.

ERC20 standard violation (fixed)

EIP-20 states:

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

However, the returned value of `approve()` function calls is not checked in the code.

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

No documentation (fixed)

The project has no documentation. As a result, it is sometimes unclear for the auditor what is the intention of the code, is its behavior correct, and whether the architecture of the project is appropriate.

Considering the complexity of the project, the documentation is critically important not only for the audit but also for development process. It should explicitly explain the purpose and behavior of the contracts, their interactions, and main design choices.

The documentation was added to the project.

Overpowered owner

The owner of **YearnVaultInstantRebindStrategy** contract can use `seizeERC20()` function to seize any tokens from the contract.

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: The contract was designed in the way when it should not keep any significant token amounts. A poke operation atomically receives vaults from a pool, unwraps them into USDC, wraps again and sends all the wrapped vaults to the pool. There could be USDC leftovers (~up to 20 weis) after the poke only but they will be re-used in the next poke operation.*

Code complexity

The logic of the contracts is overly complex. We highly recommend increasing code modularity, adding documentation and NatSpec comments, and testing the code thoroughly.

Low severity issues

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

Code quality

- `refundFees()` function of **YearnFeeRefund** contract fails at line 75 if a caller provides `crvAmount` that exceeds `pendingCrvAmount`. Instead, the function could proceed using only `pendingCrvAmount` of tokens.

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

- In `computeWeightsChange()` function of **WeightValueAbstract** contract, `_maxWPS` argument is not used. Consider implementing the proper check or removing this argument.

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

- Using `immutable` variables in upgradable contracts is confusing, and their behavior must be checked thoroughly. The use of `immutable` variable at line 9 of **SinglePoolManagement** contract leads to necessity of use both `constructor` and `initialize()` function in **YearnVaultInstantRebindStrategy** contract.

  We recommend avoiding the usage of `immutable` variables in upgradable contracts.

  > *Comment from developers: It could be confusing since we use both constructor for immutables and initializer for state variables. But it save some gas, so we'd like to continue using immutable variables.*

- Consider using `type(uint256).max` instead of `uint256(-1)` to obtain the maximum value of the type in **YearnVaultInstantRebindStrategy** contract at lines 169–171.

  > *Comment from developers: We would like to keep the current codestyle in this repo in order to not mix both options in the existing codebase.*

- In **YearnVaultInstantRebindStrategy** contract, consider moving time checks from `_poke` function to `pokeFromReporter()` and `pokeFromSlasher()` functions to simplify the logic.

  > *Comment from developers: We would like to keep the current code in order to avoid code duplications.*

- In **YearnVaultInstantRebindStrategy** contract, the implementation of `onlyEOA()` function is considered a bad practice. Consider designing the contracts the way that the check for EOA is not needed. Note, that EIP-3074 that might be implemented in the future versions of the compiler allows to bypass this check.

  > *Comment from developers: Checking for EOA helps us in preventing malicious Poker behavior. If EIP-3074 will be accepted in the future versions of EVM we will redesign contracts correspondingly.*

- The project has integration with Curve protocol. However, the developers of Curve protocol do not follow best practices. We recommend creating a wrapper for the protocol that will provide convenient interface despite the implementation details.

> *Comment from developers: There is **Erc20VaultPoolSwap** contract that have wrapper logic for Curve protocol. We will use that contract for **YearnVaultInstantRebindStrategy** in the future.*

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Vladimir Tarasov, Security Engineer
Daria Korepanova, Security Engineer
Boris Nikashin, Analyst
Alexander Seleznev, Founder

June 28, 2021