# PowerPool Security Analysis

by Pessimistic

This report is public

March 9, 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 smart contracts. We performed our audit according to the procedure described below.

The initial audit showed one issue of medium severity: Insufficient slippage protection.

Also, multiple low-severity issues were found.

After the initial audit, the codebase was updated. Most of the issues were fixed.

## General recommendations

We recommend adding NatSpec comments. We also recommend refactoring the codebase to simplify code structure and improve readability.

# Project overview

## Project description

For the audit, we were provided with PowerPool project on a public GitHub repository, commit d5a7c76322afb158d81e665444e5ba02774a4ea7.

The scope of the audit includes the following files:

- **Erc20VaultPoolSwap.sol**
- **Erc20PiptSwap.sol**
- **EthPiptSwap.sol**
- **IndicesSupplyRedeemZap.sol**

There is no detailed documentation available. The codebase contains no NatSpec comments.

The overall 461 of 462 tests pass on the project. On the scope: 10 of 10 tests pass.

The total LOC of audited sources is 1574.

## Codebase update

After the initial audit, the codebase was updated. For the recheck, we were provided with commit 4e7665b27904eabac2570007caac3b4c85a769c5

In this update most of the issues were fixed. No new functions were added, no new issues were found.

## 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 codebase with automated tools: Slither.
    - We manually verify (reject or confirm) all the issues found by tools.
- Manual audit
    - We manually analyze codebase 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 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.

### Insufficient slippage protection (fixed)

Consider adding an argument for minimum expected return amount to enable slippage protection in the `swapErc20ToVaultPool` and `swapVaultPoolToErc20` functions of the **Erc20VaultPoolSwap** contract and in the `swapPiptToEth` function of the **EthPiptSwap** contract.

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

## 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.

### Incorrect total claimed amount (fixed)

The `_claimPoke` function of the **IndicesSupplyRedeemZap** contract includes an extra (`.add(10)`) at line 528. It does not endanger the project security; however, it should be removed.

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

### Inaccurate accounting of the contract balance (fixed)

**EthPiptSwap** contract applies wrapper fees within `_joinPool` function. The contract unwraps WETH if needed. However, withdrawal amount equals fee size regardless of the current contract balance.

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

### Undocumented rounding constants (fixed)

The **EthPiptSwap** contract performs rounding at lines 272 and 414 in - all tokens are requested with the small reserve: `.add(100)`; the remainder is returned to the user in ethers. Similar to this, **Erc20VaultPoolSwap** contract utilizes `.sub(100)` for rounding.

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

### Code quality (fixed)

- Unnecessary arithmetic. The value of the variable `piptTotalSupply` is not precisely required to calculate the pool ratio. Consider omitting it at line 214 of the **EthPiptSwap** contract.

- Redundant input parameter. Consider removing the `piptTotalSupply` parameter from the function signature, as it is the value retrieved from the pool address (first function argument). Calculating the total supply amount inside the function will eliminate the risk of data inconsistency in the `getVaultCalcsForSupply` function of the **Erc20VaultPoolSwap** contract.

- Inaccurate variable naming. Since this method operates with three kinds of fee values (the pool joining fee, the wrapper fee, and the swap fee), avoid naming variables with unspecific names like `feeAmount` to reduce the probability of errors at line 381 in `_swapWethToPiptByPoolOut` function of the **EthPiptSwap** contract.

- Unstructured arguments format. Consider utilizing an array of `VaultConfig` structures as an input for the `setVaultConfigs` function to improve code readability in the **Erc20VaultPoolSwap** contract.

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

### Lack of user abstraction

Avoid utilizing `msg.sender` within internal functions. The best practice here is to pass the user address to internal functions as an argument.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Vladimir Tarasov, Security Engineer
Nikita Kirillov, Junior Security Engineer
Irina Vikhareva, Project Manager

March 9, 2022