PowerPool PVP Security Analysis

by Pessimistic

26 July, 2021

Abstract 2
Disclaimer 2
Summary 2
General recommendations 2
Project overview 3
  Project description 3
  Code base update 3
Procedure 4
Manual analysis 5
  Critical issues 5
  Medium severity issues 5
    Missing check (fixed) 5
    Erc20 standard violation (fixed) 5
  Low severity issues 6
    Code quality (fixed) 6
    Gas consumption (fixed) 6

Abstract

In this report, we consider the security of smarts contracts of PowerPool PVP project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

Summary

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

The audit showed two issues of medium severity including Missing check and ERC20 standard violation. Also, several low severity issues were found. They do not endanger project security. Nevertheless, we highly recommend addressing them.

After the initial auidt, the code base was updated. All the issues were fixed.

General recommendations

We recommend addressing compilation warnings, adding public documentation and improving NatSpec coverage.

Project overview

Project description

For the audit, we were provided with PowerPool project on a public GitHub repository, commit 91fec61830389685aead66b517ba1534787af8e5.

The scope of the audit included only **/contracts/pvp** folder.

The project has no public documentation, and the source code lacks NatSpecs. README.md file has a brief description of the structure of the project.

The project compiles with warnings. All tests pass, the code coverage is 89%.

The total LOC of audited sources is 1092.

Code base update

After the initial audit, the code base was updated. For the recheck, we were provided with commit 4bf7261cee5ea35dd3df1525c140977ab59f8ab3.

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 tools: Slither and SmartCheck.
  - 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.

Missing check (fixed)

Executing custom strategies prevents the code from performing CVP amount check in `_swap()` function of **CVPMaker** contract. Also, `Swap` events are not emitted for custom and external strategies.

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

Erc20 standard violation (fixed)

EIP-20 standard states:

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

However, returned values from `transfer()` and `transferFrom()` calls are not checked at lines 26 and 33 of **xCVP** contract and at lines 137, 315, and 398 of **CVPMaker** contract.

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

Low severity issues

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

Code quality (fixed)

- In `_swap()` function of **CVPMaker** contract, consider using `enum` type instead of `uint256` for `swapType`.
- In `setExternalStrategy()` function of **CVPMaker** contract, consider using `type(uint256).max()` instead of `uint256(-1)` at line 470.
- Adding constants to `block.timestamp` is redundant for on-chain Uniswap calls in **CVPMaker** contract at lines 195, 213, and 231.
- Consider assigning return values explicitly for `executeUniLikeFrom` address in **CVPMakerZapStrategy** contract at lines 44 and 72.
- Consider declaring functions `enter()` and `leave()` of **xCVP** contract as `external` instead of `public`.
- In **xCVP** contract, the name of the token is set to `Permanent Voting Power Token` value in the contructor at line 9. However, `name()` function returns `Permanent Voting Power` value at line 37.

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

Gas consumption (fixed)

- Since `zap` strategy is implemented as `external`, `strategy4Config` struct at line 62 of **CVPMakerStorage** contract is redundant.
- Using `memory` type for `config` variable can reduce gas conumption in **CVPMaker** contract at line 243. Also, consider using this variable when possible in `_executeExternalStrategy()` function.

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

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Vladimir Tarasov, Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

26 July, 2021