PowerPool TimedVesting Security Analysis

by Pessimistic

This report is public.

Published: May 28, 2021.

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. 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 PPTimedVesting smart contract of PowerPool project. We performed our audit according to the procedure described below.

The initial audit showed several issues of medium severity, including DoS, Vested rights transfer, Double Voting, and ERC20 standard violation. Also, two Code quality issues of low severity were found.

The project has specification, however we consider this specification incomplete.

After the initial audit, the code base was updated to the latest version. All medium severity issues except DoS were fixed. Also, developers fixed one code quality issue of low severity. For the rest of the issues, developers provided comments. Besides, a few clarifications of the documentation were made.

General recommendations

We do not have any further recommendations.

Project overview

Project description

For the audit, we were provided with PowerPool TimedVesting project on GitHub repository, commit b4043b0ac3de6bc74843048fde2b68f802d39798. The scope of the audit included only **PPTimedVesting.sol** file.

The project has a specification on powerpool-docs GitHub repository, commit 933b4d41deb2cb5f9f4ba738e5d0f6e2b7d461ad. However, it does not cover some cases of the behavior of the contract.

The code compiles without any issues, all tests pass. The overall coverage is 92.31%.

The total LOC of audited sources is 482.

Latest version of the code

After the initial audit, the code base was updated. For the recheck, we were provided with commit ccddcf4fee182e88809664178be5bd1451a09981.

Also, the documentation for the project was clarified with pull requests #8.

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 found issues.
- Manual audit
  - We manually analyze code base for security vulnerabilities.
  - We assess overall project structure and quality.
  - We check smart contracts logic and compare it with the one described in the documentation
- 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 smart contracts operation in current implementation. We highly recommend addressing them.

DoS

In **PPTimedVesting** contract, when the owner increases the duration of the vesting using `increaseDurationT()` or `increasePersonalDurationsT()` function, `getAvailable()` function might revert due to underflow check at line 444. As a result, `claimTokens()` calls will fail without proper reason string for some users, and view functions `getAvailableTokens()` and `getAvailableTokensForMember()` will not work.

Comment from developers: these `increaseDurationT()` and `increasePersonalDurationsT()` introduce a gap periods when an affected user could have more claimed tokens than he should according to the new duration. The `claimTokens()` method will fail only during this period and will continue work correctly after there will be some tokens to claim according to the `getAvailableTokens()` method. We will show an explanation for such cases in UI.

Vested rights transfer (fixed)

When vested rights are transferred to an address via `PPTimedVesting.transfer()`, this address might be a delegatee and have some checkpoints. This scenario is not considered in the current implementation and might break voting power history available via `getPriorVotes()`.

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

Double voting (fixed)

In **PPTimedVesting** contract, users can vote twice with the same tokens using personal vesting duration feature.

To perform a double voting, user A can delegate tokens to user B, who has increased vesting duration. After the vesting ends for user A, he/she claims all his/her tokens and vote with them. Meanwhile, user B participates in the same voting as a delegate of user A.

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

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, `IERC20.transfer()` function does not return any value in **PPTimedVesting.sol** at line 12. As a result, there are no checks for returned value at lines 527 and 655.

Comment from developers: although this vesting contract is intended to be used with CVP token which reverts transaction instead of returning bools, we decided to implement this check with this update.

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

Discrepancy with documentation (fixed)

According to the documentation, `A member can delegate votes to a non-member address;`

However, a non-member delegatee cannot vote due to membership check in `getPriorVotes()` function.

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

Low severity issues

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

Code quality

- `getPriorVotes()` and `getMemberEndT()` functions of **PPTimedVesting** should be declared as `external`.

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

- In **PPTimedVesting** contract, `getAvailableVotes()` function is called with named arguments list (`foo(arg1:val1, arg2:val2)`) for no reason. Consider using standard syntax at lines 384, 553, 645, and 735 to improve code readability.

  Comment from developers: we'd like to keep using named arguments so it helps understand how does each argument is used.

This analysis was performed by Pessimistic.

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

May 28, 2021