PowerPool Vesting Security Analysis

by Pessimistic

Published: September 23, 2020

Abstract 2
Disclaimer 2
Summary 2
General recommendations 2
Procedure 3
Project overview 4
  Project description 4
  Latest version of the code 4
Manual analysis 5
  Critical issues 5
  Medium severity issues 5
  Low severity issues 5
    Gas optimization 5
    Tests 6
    Empty contract 6
    Documentation 6
    Code style 7
    Design 7

Abstract

In this report, we consider the security of CVP token vesting contract 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 CVP token vesting contract of PowerPool project. We performed our audit according to the procedure described below.

The initial audit showed no vulnerabilities. We provided some recommendations on improving code style and optimizing gas consumption. Though, these were minor issues that did not endanger project security.

After initial audit, the code was updated. Developers fixed some issues and provided comments for the rest.

General recommendations

We do not have any further recommendations.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether code logic corresponds to the specification.
2. Whether the code is secure.
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Manual audit
    - We inspect the specification and check whether the logic of smart contracts is consistent with it.
    - 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.

Project overview

Project description

In our analysis we consider CVP vesting contract of PowerPool project on GitHub repository, commit c0e7e0d60bb8db0b86561d9b8b272002641328a9, and the specification, commit d142179ba15916b80aaf8c92a23be36466300dbe.

The total LOC of audited sources is 282.

Latest version of the code

After initial audit, some fixes were applied. The latest version of the code, commit e5f478f97a6ba58c851e56d5addeedc2629ed84b. The documentation was also updated, commit cef6cea1ca19cdd15b42fd9de9a48fc531a8645e.

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.

The audit showed no medium severity issues.

Low severity issues

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

Gas optimization

- According to documentation, `there will be multiple instances of this contract with different initialization parameters`.
  Consider using a factory or proxies to decrease gas consumption when deploying contracts.

  > *Comment from developers: There will be just 2 or 3 instances of this contract. Our team will deploy all these instances. So, the factory will be redundant.*

- According to documentation, `compilation configuration: Solidity version: v0.6.12, optimizer: enabled, runs: 20000`.
  Consider decreasing `runs` value to appropriate level.

  The documentation was updated. The issue is not present in the latest version.

- There are `uint32` and `uint96` used in **PPVesting.sol** to reduce storage footprint. However, in memory, it is cheaper to work with `uint256`. Shorter `uint` types are useful when declaring structs or assigning to structs fields.

  > *Comment from developers: We want to reuse the caching pattern from CVP token and `_writeCheckpoint` method, so we accept a small gas usage increase due to these `uint96`/`uint32` integer operations.*

Tests

Tests pass without any issues. However, two tests fail when running coverage:

```
Error: ✕ 2 test(s) failed under coverage.
    1) should deny initialization with non-erc20 contract address
    2) should deny initialization with non-erc20 address
```

> *Comment from developers: These failing methods rely on JS-level error messages, not EVM ones. Perhaps, the solidity-coverage environment cuts them out due to some unknown reason. We can fix this by asserting that the method calls revert without an additional message check. But we prefer to keep these assertions against the particular error messages and allow coverage tests to fail.*

Empty contract

The repository contains **Imports.sol** contract that has empty body.

> *Comment from developers: Using an **Imports.sol** file with an empty body is a typical pattern to force importing the contracts not imported by any other contracts. This particular case includes a contract used only by test scripts, but we are ok keeping this file in the **Contracts** folder.*

Documentation

- The documentation has a Methods section. However, the list of methods here is incomplete.

  The documentation was updated. The issue is not present in the latest version.

- According to documentation, `a member has an option to transfer his or her vested tokens to another address, for example in a case when a private key is compromised`. However, `transfer` function requires recipient address to be new.

  The documentation was updated. The issue is not present in the latest version.

Code style

There are the following style issues in **PPVesting.sol** contract:

- Lines 171–200: consider removing commented code.

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

- Lines 316, 342: there is no use in storing `msg.sender` in local variable.

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

- mappings `checkpoints` and `numCheckpoints` can be effectively combined into a single `mapping(address => Checkpoint[]).`

  > *Comment from developers: We prefer to keep this logic resembling CVP token checkpoint behavior.*

- It is unclear whether `availableToWithdraw` function should be `pure`, as it results in code duplication in `availableToWithdrawForMemberInTheNextBlock` and `availableToWithdrawFor` functions.

  > *Comment from developers: We keep this function pure to make it explicit how the vesting formula works. Also, making this function pure makes unit testing more convenient.*

Design

- It is possible to preallocate tokens to the address where the contract will be deployed. This allows perform a check inside the `constructor`, whether the contract has proper balance.

  > *Comment from developers: It is not necessary to pre-allocate funds before the contract deployment.*

- Sometimes block times behave unreliably. Thus, vesting should probably be time-dependent, not block-dependent.

  > *Comment from developers: Such deviations are not crucial for the task of token vesting.*

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer

Boris Nikashin, Analyst

Alexander Seleznev, Founder

September 23, 2020