Super Protocol

Super Protocol Token and Vesting Security Analysis

by Pessimistic

This report is public

May 19, 2022

Abstract

In this report, we consider the security of smart contracts of Super Protocol 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 Super Protocol smart contracts. We performed our audit according to the procedure described below.

The initial audit showed one issue of medium severity: Initialization prevention. Also, three low-severity issues were found.

Two concerns regarding the smart contracts functionality and project management were put into the Notes section.

After the initial audit, the codebase was updated. In this update, the developers fixed some of the previously discovered issues. During the recheck process, we discovered ERC20 standard violation issue of medium severity. The developers fixed this issue on the latest commit. However, some of the previously discovered issues remain relevant.

General recommendations

We recommend fixing the mentioned issues and improving NatSpec coverage. We also recommend analyzing code with linters and security tools.

Project overview

Project description

For the audit, we were provided with Super Protocol Token and Super Protocol Vesting projects on private GitHub repositories.

Commits:

- Token: db67de0c9bd58e84d413104d62343b0d47b1e7d4.
- Vesting: 2a4655174a2e8f948c5c45a02d64d9876ca05d2a.

Scope:

- Token: Everything.
- Vesting: `InsidersVesting.sol`.

The project has detailed documentation.

All tests pass. The code coverage is 100%.

The total LOC of audited sources is 207.

Codebase update

After the initial audit, the developers performed a few updates. The latest commits for the recheck were:

- Token: 98e8a236d99ece0d4d1639a831750de789b3e2da.
- Vesting: d9faf2d4cb145074fa30eecd37ebb5f182d4e3bb.

The audit scope did not change.

During the recheck process, we found an issue of medium severity. The developers fixed it at the latest commit.

Overall, in this update, the developers fixed two issues of medium severity and two low-severity issues. They also provided comments for some of the issues that were not fixed.

Token details

Name:	Super Protocol
Symbol:	TEE
Decimals:	18
Total Supply:	1 000 000 000

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 the project's codebase with the automated tool: Slither
  - We manually verify (reject or confirm) all the issues found by the tool.
- Manual audit
  - We manually analyze the codebase for security vulnerabilities.
  - We assess the 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.

M01. Initialization prevention

The `initialize` function in **InsidersVesting** may fail due to `require` at line 95, which checks that all tokens are distributed to beneficiaries. If a malicious user sends tokens before the `initialize` function execution, the token balance check will never pass.

> *Comment from the developers: The process of Vesting initialization is done manually by trusted authorities right after TGE. TGE, in its turn, will airdrop all of the minted tokens to a strict list of receivers (smart contracts and multisigs) according to their shares. That makes it very unlikely that malicious actors can get access to tokens. Checking that all tokens are distributed to beneficiaries helps us ensure that we did all the calculations correctly.*

M02. ERC20 standard violation (fixed)

ERC-20 standard states:

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

However, the returned value from `transfer` call at line 113 is not checked.

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.

L01. Code quality (fixed)

Consider declaring `airdrop` function in **SuperProtocol** as `external` rather than `public` to improve code readability and optimize gas consumption.

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

L02. Code quality

Consider moving `airdrop` function from **SuperProtocol** to a separate smart contract as it is not related to the token contract. This will improve the code structure and the reduce attack surface of the token contract.

L03. Project management (fixed)

Both **InsidersVesting** and **SuperProtocol** contracts include copied standard code like `OpenZeppelin`. Consider managing external libraries as dependencies.

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

Notes

N01. Gas consumption

`_calculateClaimAndStage`, `claim`, and `_transfer` functions update the same storage slots multiple times. Repetitive writing to storage increases gas consumption. Consider updating a struct as a whole when possible.

> *Comment from the developers: In those functions, changes to storage variables are consecutive, which aims to minimize complexity, increase readability and coherence of the contract. First, we freeze the vesting state, then we make operations on balances.*

N02. Project management (fixed)

Consider storing contracts that are used for testing in a separate folder to better indicate their intended usage.

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

This analysis was performed by Pessimistic:

Pavel Kondratenkov, Security Engineer
Nikita Kirillov, Junior Security Engineer
Irina Vikhareva, Project Manager
Alexander Seleznev, Founder

May 19, 2022