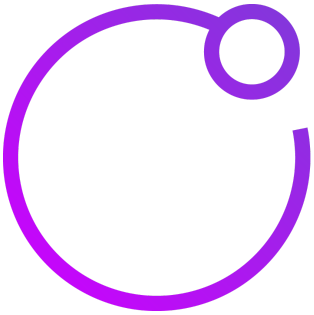

DeNet Security Analysis

by Pessimistic

This report is public

February 7, 2023

Abstract

In this report, we consider the security of smart contracts of DeNet 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 DeNet smart contracts. We described the audit process in the section below.

The initial audit showed several issues of medium severity: Problem with share deletion, Tests, Treasury requirement for the vesting process, Unchecked transfer result. Also, several low-severity issues were found.

The initial project was of mediocre code quality. It had detailed NatSpec comments, however it contained several issues and poorly handled edge cases. Moreover, the project contained few tests.

After the audit, we were provided with the updated version of the code. In that update, the developers fixed all of the found issues. However, they implemented only one new test, and we do not consider coverage as a sufficient one.

General recommendations

We recommend implementing more tests.

Project overview

Project description

For the audit, we were provided with DeNet project on a private GitHub repository, commit cdba018cf19b008d800ed45c39a32a7577e88763.

The scope of the audit includes **Constant.sol**, **DeNetFileToken.sol**, **ERC20Vesting.sol** files and their dependencies.

The documentation for the project includes README.md file and comments in the code.

All 3 tests pass successfully. The code coverage is 34.29%.

The total LOC of audited sources is 191.

Codebase update

After the initial audit, the codebase was updated. For the recheck, we were provided with commit c46f96df4283eeafc6b425837180f74abdf8b411.

This update fixes all found issues except Tests one. The developers implemented new test, however we still recommend implementing more.

All 4 tests pass successfully. The code coverage is 47.74%.

Audit process

We started the audit on January 23, 2023 and finished on January 24, 2023.

We inspected the materials provided for the audit. Then, we contacted the developers for an introduction to the project.

During the work, we stayed in touch with the developers and discussed confusing parts of the code.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the contracts:

- The correctness of the vesting process, i.e. users receive their tokens in time;
- Compliance of the code with the documentation.

We scanned the project with the static analyzer Slither and our private plugin with extended set of rules and then manually verified their output.

We ran tests and calculated the code coverage.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

After the initial audit, we discussed the results with the developers. On February 6 the developers provided us with an updated version of the code. In this update, they fixed all of the issues from our report and added one test. We reviewed the updated codebase and did not find any new issues.

Finally, we updated 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. Problem with share deletion (fixed)

In **DeNetFileToken.sol** the `_removeShares` function allows to remove user shares. However, it does not remove user from `sharesVector` mapping in case when resulting user shares is zero. This behavior allows to save gas, however it has the following consequences:

- `smartMint` function will not work properly if some user has zero shares. Line 180 of **DeNetFileToken.sol** will revert transaction due to `require` check at line 47 of **ERC20Vesting.sol**;
- If the owner deletes user shares and then adds them again, the user address will be stored twice in `sharesVector` mapping due to check at line 81. As a result, the `smartMint` function will not work properly since the `createVesting` function does not allow to vest tokens to the same user twice;
- Since `sharesCount` variable value is not decreased, it is possible to bypass check at line 162.

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

M02. Tests

The project contains only 3 tests and a code coverage is 34.29%. Consider increasing the code coverage in order to mitigate possible future issues.

In the new version of the code, the developers added one test. Despite the fact that the least tested code is out of the audit scope, we still recommend covering the whole project with tests.

M03. Treasury requirement for the vesting process (fixed)

In the current implementation, the owner is able to distribute less than 100% of the tokens. In that case, undistributed tokens will be vested to the `treasury` address. According to the `_addShares` function, distribution of 100% of the tokens is a valid scenario (line 78). However, if all of the tokens are distributed, the `smartMint` function will not work due to lines 184-185 since **ERC20Vesting** contract does not allow to create vesting with zero amount.

In addition to this, the `smartMint` function checks that the `treasury` address is not zero at line 164 even if `_treasuryAmount` is zero.

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

M04. Unchecked transfer result (fixed)

According to ERC20 standard:

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

Thus, we recommend adding `require` check at line 83 in **ERC20Vesting.sol** file, despite the fact that vested token is a known one.

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)

In **DeNetFileToken.sol**, lines 79 and 98 contain the following revert message: `Shares: Size < 0`. However, `_size` variable cannot be less than zero since it has `uint` type. We recommend changing revert message for the appropriate one.

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

L02. Code quality (fixed)

In **DeNetFileToken.sol** file, `require` check at line 97 is redundant. If `shares[_receiver]` is less than `_size`, then subtraction operation will revert the transaction. We recommend changing `require` statement to the following: `shares[_reciever] >= _size`.

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

L03. External vs public (fixed)

Consider declaring functions as `external` instead of `public` where possible to improve code readability and optimize gas consumption in the project.

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

L04. Mark variable as immutable (fixed)

In **ERC20Vesting.sol**, consider marking `_vestingToken` variable as immutable in order to reduce gas costs at deployment.

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

L05. Missing check (fixed)

In **ERC20Vesting.sol** file, `getAmountToWithdraw` function can be called for the user without vesting information. In that case, this function will revert transaction due to zero division error at line 66. We recommend handling this case.

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

L06. Possible storage layout optimization (fixed)

Developers use various integer types. Thus, we recommend considering optimizing storage layout by using integer types that require less bytes. For example, in **DeNetFileToken.sol** `timeNextYear`, `currentYear` and `sharesAvailable` variables can be packed with others.

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

Notes

N01. Multiplication after division

In **ERC20Vesting.sol** file, `getAmountToWithdraw` function contains multiplication after division. Since Solidity language operates with integers, such order of operation reduces the calculation accuracy. Note that in that case it is not a vulnerability since user receives all of the tokens after vesting ending. Still, we recommend paying attention to operations order.

N02. Owner role

The owner role is crucial for the project. It is responsible for the following:

- Distributing tokens by calling `addShares` and `removeShares` functions;
- Setting treasury address;
- Creating vesting contracts by calling `smartVesting` function.

In the current implementation, the system depends heavily on the owner. Thus, there are scenarios that can lead to undesirable consequences for the project and its users, e.g., if admin's private keys become compromised. Note, that the developers are planning to use multisig wallet for the owner role address.

This analysis was performed by Pessimistic:

Pavel Kondratenkov, Security Engineer
Yhtyyar Sahatov, Junior Security Engineer
Irina Vikhareva, Project Manager
Alexander Seleznev, Founder

February 7, 2023