AEGIS Token Security Analysis

by Pessimistic

This report is public

June 28, 2022

Abstract

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

The initial audit showed issues of low and note severities.

During recheck the code base was updated. In this update, all issues were fixed.

General recommendations

We do not have any further recommendations regarding the code base. Consider setting up monitoring to deployed contracts.

Project overview

Project description

For the audit, we were provided with AEGIS Token project on a private GitHub repository, commit 167821b0513f3fb5e0f4ad8444dbed02d93c57b3.

The project compiles successfully, has no tests, no dependency management.

The total LOC of audited sources is 10 lines.

Token details

Name:	AEGIS
Symbol:	AEGIS
Decimals:	18
Total Supply:	1100000000

Code base update

For the recheck, we were provided with AEGIS Token project on a private GitHub repository, commit 364db60df7e2a885b9803f1763455a5d7f5f1f06.

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 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.

The audit showed no medium issues.

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.

Code quality (fixed)

- According to openzeppelin documentation `_beforeTokenTransfer` function should be declared as `virtual`.
- Consider declaring `pause` and `unpause` functions as `external` instead of `public` to improve code readability and optimize gas consumption.

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

Gas consumption (fixed)

`dec` variable can be declared as `local` since it is used in `constructor` at lines 13-14 only for quick calculation.

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

Note

Overpowered owner (fixed)

The owner of **AEGIS** can `pause` and `unpause` contract.

In the current implementation, the system depends heavily on owner of this contract. Thus, there are scenarios that can lead to undesirable consequences for the project and its users, e.g., if private key of the owner becomes compromised.

We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

> *Comment from the developers: This contract has been deployed at `0x15Dd37a83564559927bBDcDbfcF85F60A2C6a419` address. The owner of the contract is the multisig wallet at `0xE9f439bCC2FdbEd88f2d2b699550C1562F6F92b9` address.*

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Daria Korepanova, Security Engineer
Irina Vikhareva, Project Manager

June 28, 2022