SWT Token
Security Analysis

by Pessimistic

4 October, 2021

Abstract

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

The audit showed no critical issues. However, two violations of BEP20 standard and number of medium and low severity issues were found. They do not endanger project security. Nevertheless, we highly recommend addressing them.

General recommendations

We recommend fixing the mentioned issues and covering the code with tests and NatSpec comments.

Project overview

Project description

For the audit, we were provided with SWT Token project on a private GitHub repository, commit 60d16ab33f6de8db49671739932d630f86c1c3d3.

The repo only includes a single smart contract with its dependencies. The project doesn't have public documentation, README, or detailed comments in NatSpec format. There are no tests or project files.

The total LOC of audited sources is 145.

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 often lead to the loss of funds or other catastrophic failures. The contracts should not be deployed before these issues are fixed. We highly recommend fixing them.

The audit showed no critical issues

Medium severity issues

Medium issues can influence project operation in 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.

BEP20 standard violation

- The **BEP20Interface** interface doesn't follow ERC-20 standard : `transfer`, `transferFrom`, and `approve` functions at lines 29-31 must return bool value.
- ERC-20 standard states :

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

  However, the value returned by `transfer` call is not checked inside `transferAnyBEP20Token` function of **SWT** contract.

No Documentation

The project doesn't have public documentation. The code base of the project includes only minimal comments, contracts and functions don't have a description. We recommend covering the code with `NatSpec` comments, as it helps to avoid errors and accelerates development process.

No Tests

The provided code does not contain tests. Testing is crucial for code security and audit does not replace tests in any way. We highly recommend both covering the code with tests and making sure that the test coverage is sufficient.

Low severity issues

Low severity issues don't directly affect project's operations. However, they might lead to various problems in the future versions of the code. We recommend taking them into account.

SafeMath and Solidity 0.8

Math operations will revert on over- and underflows by default starting with Solidity version `0.8.0`. It is recommended to remove SafeMath from the project to improve code readability and decrease gas consumption.

Code quality

- Storage variables are considered `internal` by default. Declare visibility of `_totalSupply`, `balances`, and `allowed` variables explicitly to improve readability of **SWT** contract.
- Mark functions as `external` whenever possible. It improves readability and saves a bit of gas.

Notes

Overpowered role

The owner of the contract can pause transfers and mint new tokens. Therefore, there are scenarios that may lead to undesirable consequences for the project and its users, e.g., if admin's private keys become compromised. We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

Design drawback

We don't recommend implementing `approveAndCall` functionality, as it can lead to a reentrancy or other vulnerabilities for the projects that interact with **SWT** token.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Vladimir Tarasov, Security Engineer
Alexander Seleznev, Founder

4 October, 2021