KickToken Security Analysis

by Pessimistic

This report is public.

Published: June 28, 2021

Abstract

In this report, we consider the security of token contract of KickToken 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 KickToken smart contract. We performed our audit according to the procedure described below.

The initial audit showed critical Access control issue, three issues of medium severity, including ERC20 incompatibility, Overpowered owner, and DeFi integration. Also, several issues of low severity were found.

After the initial audit, the code base was updated to the latest version. In this update, most of the issues were fixed, including Access control and Overpowered owner.

General recommendations

We recommend adding NatSpecs to the code.

Project overview

Project description

For the audit, we were provided with KickToken project on a private GitHub repository, commit 61d6822b8d433b6dfcf51684254cec95fe857236.

The project has **README.md** file. However, the code has no NatSpecs.

All tests pass without issues, the coverage is 99.15%/

The total LOC of audited sources is 240.

Latest version of the code

After the initial audit, the code base was updated. For the recheck, we were provided with commit ac39c1d94e9ef68c16cad4110b3e12d85a2c749f.

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 smart contracts security. We highly recommend fixing them.

Access control (fixed)

Function `_distribute()` at line 270 is declared as `public`. Since this function does not have any internal checks, it allows anyone to distribute tokens from any address.

The function should be declared as `internal`.

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

Medium severity issues

Medium issues can influence project operation in current implementation. We highly recommend addressing them.

ERC20 standard incompatibility

According to ERC20 standard, `Transfer` event `MUST trigger when tokens are transferred, including zero value transfers`. Therefore, `_distribute()` function MUST emit `Transfer(account, address(0), tAmount)` event and then `Transfer(address(0), user, tAmount)` events for other users which is barely possible. Thus, the token cannot be ERC20 compatible.

> *Comment from developers: it is impossible to emit fully ERC20 compatible events here due to the token design (distribution mechanism). Fixed via adding a custom event* `Distribution(address, amount)`.

Overpowered owner (fixed)

The owner of the token can burn or distribute tokens from any address.

In the current implementation, the system depends heavily on the owner of the contract. Thus, there are scenarios that may lead to undesirable consequences for investors, e.g. if the owner's private keys become compromised.

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

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

DeFi integration

The token contract has transfer fee and rebase features, that severely complicate DeFi integration. Similar properties can be achieved without compromising composability.

We recommend redesigning the system.

> *Comment from developers: we are aware that the token design (distribution mechanism) leads to some DeFi integration problems. For correct DeFi smart contracts execution, there is NoIncomeFee role: no burn happens and no distribution fee is taken when tokens are transferred to these addresses*.

Low severity issues

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

Code quality

- For the maximum value of type `uint256` consider using `type(uint256).max` instead of `~uint256(0)` at line 14.

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

- `_transferStandard()` function does not emit all required events. Consider adding them or using `_burn()` function from within.

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

- When performing arithmetical operations in Solidity, multiplication should precede division to minimize rounding error. Consider applying this rule to lines 180 and 187.

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

- Consider declaring functions as `external` instead of `public` where possible.

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

- Since version 0.8.0 Solidity performs overflow checks automatically. Therefore, `SafeMath` library is redundant.

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

- Typos in the code:
  - At line 41 should be `ticker` instead of `tiker`.
  - At line 165 should be `distribute` instead of `destribute`.

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

- In `totalSupply()` function, `_tTotal` value at line 81 may be not equal to the sum of `balanceOf()` of all users due to rounding issue.

  > *Comment from developers: the token design (distribution mechanism) has this side effect. However, it does not lead to token execution problems or any integration issues. Rounding of those micro amounts is of negligible importance and can be neglected*.

- The code has no NatSpecs.

Gas consumption (fixed)

Smaller `uint` types increase gas consumption when used for parameters only. Consider using `uint256` type in `constructor` at lines 44–45.

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

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Igor Sobolev, Security Engineer
Vladimir Tarasov, Security Engineer
Boris Nikashin, Analyst
Alexander Seleznev, Founder

June 28, 2021