Parsiq Security Analysis

by Pessimistic

This report is public.

Published: March 29, 2021

Abstract

In this report, we consider the security of token smart contracts of Parsiq project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

Summary

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

The audit showed no critical issues. However, several issues of medium severity were found including Missing modifier and Overpowered owner. Also, the audit revealed several low severity issues of Code quality and Dependency management types.

The project has no documentation. Tests coverage is low, some functionality is not covered with tests at all.

The token contract is deployed to mainnet.

After the audit, developers provided comments for issues of medium severity. We added these comments to the report.

General recommendations

We recommend adding documentation to the project. In case if developers decide to redeploy the contract, we highly recommend fixing the mentioned issues, and monitoring and improving tests coverage.

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.

Project overview

Project description

For the audit, we were provided with a project on a GitLab repository, commit a66d37fe57b40231f1a63f718a00fded51a3472a.

The project has no documentation.

The token contract has been deployed to mainnet already at address 0x362bc847A3a9637d3af6624EeC853618a43ed7D2.

The total LOC of audited sources is 267.

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.

The audit showed no critical issues.

Medium severity issues

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

Overpowered owner

The owner of the contract can pause the contract and grab tokens from any account.

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.

> *Comment from developers: We are aware of this moment and we are implementing a proper multisig in the future. The owner will be owerpowered in some sense, anyway, but multisig will add an extra layer of security for sure.*

Missing modifier

`permit()` function does not have `whenNotPaused` modifier. Therefore, it can be used when the contract is paused. All other methods that perform `_approve()` call do have `whenNotPaused` modifier.

> *Comment from developers: Agree, but it does not influence critical token functionality. The idea behind "Pausable" is to stop all token transfers when some critical bug is found or on force majeure situation. Current missing modifier allows only to "approve" tokens, but not to transfer them when the token is paused, which fully corresponds to "Pausable" idea.*

No documentation

The project has no documentation. The documentation helps auditors to verify the correctness of the code. It also helps other developers with token integration.

We strongly recommend adding public documentation to the project.

> *Comment from developers: Will be provided at some point of time in the future.*

Low tests coverage

The provided code has tests. However, `permit` functionality is not covered with tests. Also, the coverage is not measured.

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.

> *Comment from developers*: `permit()` *functionality was copied from Uniswap ERC20 source code, which is tested thoroughly. But yes, we agree more tests should be added to the project.*

Low severity issues

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

Code quality

- Consider using `type(uint256).max` syntax to obtain maximum value of `uint256` type at line 193. This will also make `MAX_UINT256` constant redundant.
- Consider declaring functions as `external` instead of `public` where possible.
- Consider using `1 days` instead of `86400` at lines 28–29 for better readability.
- We recommend assigning `keccak256("Permit(address ...")` value to `PERMIT_TYPEHASH` constant instead of hexadecimal string. It is computed at compile time and significantly improves code readability.

Dependency management

- The project does not work if `truffle` package is not installed globally. Consider running `truffle` commands using `npx`, e.g. `npx truffle compile`.
- Packages `ganache-cli` and `npm-run-all` should be moved to `devDependencies`.

This analysis was performed by Pessimistic:

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

March 29, 2021