Pulselnu Audit

November 2022

By CoinFabrik

v202209

Executive Summary

CoinFabrik was asked to audit the contracts for the PulseINU project. The audited files are from the git repository located at https://github.com/ricknightcrypto/pulseinu_V2. The audit is based on the commit `ebd2172b07c104235a8f23fda73fdbe989d53f71`.

Fixes were checked on commit `487b56f08f19b23a684107b2461b8cd893b5f708`.

During this audit we found one critical issue, one minor issue and no medium issues. Also, an enhancement was proposed. All issues were fixed.

Scope

The scope for this audit includes and is limited to the following files:

- `contracts/PulseInu.sol`: Main token contract.
- `contracts/interfaces/IPulseInu.sol`: Token interface.
- scripts/deploy.js: Deployment script.

No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

Methodology

CoinFabrik was provided with the source code. Our auditors spent one week auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function, understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. Without being limited to them, the audit process included the following analyses:

- Arithmetic errors
- Outdated version of Solidity compiler
- Race conditions
- Reentrancy attacks
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters

- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

After delivering a report with our findings, the development team had the opportunity to comment on every finding and fix the issues they considered convenient. Once fixed and/or commented, our team ran a second review process to verify that the changes to the code effectively solve the issues found and do not unintentionally add new ones. This report includes the final status after the second review.

Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

ID	Title	Severity	Status
CR-01	Incorrect Calculation Of INITIAL_SUPPLY	Critical	Resolved
MI-01	Loss Of Precision Due To Multiplication After Division	Minor	Resolved

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. They must be fixed **immediately**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of, but might be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit has one of the following statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision.
- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.

Critical Severity Issues

CR-01 Incorrect Calculation Of INITIAL_SUPPLY

Location:

- `contracts/PulseInu.sol:86`

The `INITIAL_SUPPLY` variable stores `PINU` tokens. In the function `claimFreeClaimer()`, the variable `airdropAmount` is added to it. This variable `airdropAmount` is then multiplied by the decimals, so the correct value can be minted.

But in the contract math operations, it's expected that `INITIAL_SUPPLY` must be the full token value, including decimals, so the function `claimFreeCliamer()` is calculating `INITIAL_SUPPLY` incorrectly.

Recommendation

Multiply the `airdropAmount` value by the decimals before adding it to `INITIAL_SUPPLY`.

Status

Resolved. The `INITIAL_SUPPLY` variable is now correctly calculated.

Medium Severity Issues

No issues found.

Minor Severity Issues

MI-01 Loss Of Precision Due To Multiplication After Division

Location:

- `contracts/PulseInu.sol:113`

In the `ClaimReferrer()` function, the variable `_referrerAmount` contains the result of several divisions and multiplications. Due to the nature of integer arithmetic, a division will produce a loss of precision in the final result, that is then amplified by the subsequent multiplication.

Recommendation
Refactor the calculation so all multiplications are done first, and then all divisions.

Status
Resolved. The calculation is now done in the correct order.

Enhancements

These items do not represent a security risk. They are best practices that we suggest implementing.

ID	Title	Status
EN-01	Reentrancy Guards Not Needed	Not implemented

EN-01 Reentrancy Guards Not Needed

Location:
- `contracts/PulseInu.sol:154,171`

Recommendation
As the contract does not call any external functions, nor has any callback functionality, reentrancy guards are not needed.

Status
Not implemented.

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest to other stakeholders of the project, including users of the audited contracts, token holders or project investors.

Centralization

After the minting period is over, the address at `OWNER_ADDRESS` can always withdraw all funds from the contract using the function `withdraw()`.

Upgrades

No mechanism for upgrade is provided by the audited contract.

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

PulseInu.sol

OWNER_ADDRESS

This address can withdraw all funds from the contract, after the minting period is over. This address is hardcoded and cannot be changed.

Changelog

- 2022-11-11 – Initial report based on commit `ebd2172b07c104235a8f23fda73fdbe989d53f71`.
- 2022-11-15 – Checked fixes based on commit `487b56f08f19b23a684107b2461b8cd893b5f708`.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the PulseInu project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.