DeFiPie Security Analysis

by Pessimistic

This report is public.

Published: January 26, 2021

Abstract

In this report, we consider the security of smart contracts of DeFiPie project. Our task is to find and describe security issues in 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 smart contracts of DeFiPie project. We performed our audit according to the procedure described below.

The initial analysis only showed an Overpowered owner issue (medium severity) and several issues of low severity, mostly referred to as code quality issues.

The project has documentation and tests. Overall code quality is good.

After the audit, the developers provided comments to most of the mentioned issues. These comments give comprehensive explanations why exact decisions were made. According to the comments, Overpowered owner issue will be addressed soon when the governance module testing is complete.

General recommendations

We recommend using multisig for the owner of the contracts until governance module is deployed on the mainnet.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether code logic corresponds to the specification.
2. Whether the code is secure.
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 compare the project's code to Compound's codebase and check whether the changes are consistent with documentation.
    - We manually analyze new code for security vulnerabilities.
    - We assess overall project structure and quality.
- Report
    - We reflect all the gathered information in the report.

Project overview

Project description

In our analysis we consider smart contracts of DeFiPie project on GitHub repository, commit 81ab5212ce13b5a3625123fe80980f13520e7677.

The documentation for the project was provided as a Google Docs file. The document saved as a plain text (.txt) file has sha1sum d4cbb27c6e5c321ec66b1404528b7a170c10b665.

The project is based on Compound codebase, which was updated to Solidity 0.7.

The total LOC of audited sources is 6853.

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 can update contracts and manipulate critical settings of the project. In Compound, these changes can only be made by Governance system.

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 the developers: at the moment, the governance module is being tested in the rinkeby test network, with the subsequent transfer of control to the community.*

Low severity issues

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

Code quality

- In **RegistryProxy**, **UniswapPriceOracleProxy**, **PErc20Delegator**, and **PPIEDelegator** contracts, `delegateAndReturn()` function is redundant as `delegateTo(_pTokenImplementation(), msg.data)` can be used instead. We recommend removing `delegateAndReturn()` function.

  > *<u>Comment from the developers</u>: Because this redundant code is in proxy contracts and does not endanger the system, we decided to postpone these changes until the new version.*

- In **Unitroller**, **UniswapPriceOracleProxy**, **PPIEDelegator**, **RegistryProxy**, and **PErc20Delegator** contracts, the checks inside `receive()` and `fallback()` functions are redundant. To prohibit ether transfers, declaring `fallback()` function as non-payable and removing `receive()` function is enough.

- In **ErrorReporter.sol** file, `enum`s `Error` and `FailureInfo` were updated and new values were inserted at lines 19 and 41 (`PRICE_UPDATE_ERROR` and `SET_PAUSE_GUARDIAN_OWNER_CHECK`, respectively). As a result, `uint` values of all subsequent `enum` elements changed. This might affect off-chain code.

  We recommend appending new elements to the end of `enum`s or checking thoroughly that all the values are processed correctly by off-chain code.

  > *<u>Comment from the developers</u>: These updates were made before the contracts had been deployed, so they could not affect the off-chain code. In the future, we will add new elements only at the end of the enums.*

- In `_acceptAdmin()` of **RegistryProxy** contract, `msg.sender == address(0)` check is redundant: `address(0)` cannot be a `msg.sender`.

- In **Registry** contract, the returned value of the call at line 86 in `addPToken()` function is not checked. We recommend adding `require` check for better readability.

  > *<u>Comment from the developers</u>: In this case, the result of the return of the `isPToken` function is not important to us, it is important that this attribute is present in the contract to make sure that the `pToken` address was specified correctly. After all, if this attribute is not present, the transaction will fail.*

- In **ErrorReporter.sol** file, there are TODO comments at lines 251 and 292:

  ```
  //TODO: Add more cases
  ```

  We recommend completing the code or removing/updating these comments.

- When declaring functions, consider using `external` instead of `public` where possible.

Code logic

In `delegateBySig()` function of **PPIE** contract, if a meta-transaction is not executed in time (i.e., before `expiry`), the following meta-transactions cannot be executed as well due to incorrect `nonce` value.

We recommend implementing the functionality so that system recognizes such situations and resets `nonce` value to the correct one.

> *Comment from the developers: if the user makes 3 transactions logically dependent on each other (within the framework of the contract state), then with independent processing, the logical sequence may be violated.*
>
> *In turn, if the transaction is timed out, the user can re-sign the transaction again.*

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer

Boris Nikashin, Analyst

Alexander Seleznev, Founder

January 26, 2021