Rarible

Rarible Migration Security Analysis

by Pessimistic

This report is public

January 10, 2024

Abstract

In this report, we consider the security of the migration process of Rarible project. Our task is to find and describe security issues and concerns in the process.

Disclaimer

The consultation does not give any warranties on the security of the code. It cannot replace an audit of the whole system. A single consultation cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the system. Besides, a security consultation is not investment advice.

Summary

In this report, we considered the security of Rarible migration process. We described our review process in the section below.

The overview showed one issue of medium severity: Code logic. Also, several low-severity issues were found.

During the initial review, we identified one medium-severity and several low-severity issues. We recommended reviewing these issues prior to deployment, although they did not prevent the code from being used in the production environment in that implementation.

After the review, we received a new version of the code. In this version, the developers resolved all but one low-severity issue. Additionally, we identified two new issues (L04 and L05).

Project overview

Project description

For the consultation, we were provided with Rarible project on a public GitHub repository.

The scope of the consultation included pull request #263. At the moment of the consultation, the pull request contained changes between commits 500f748264ea74a0ba38fab99907c56358d27134 and dd65a17de6a0934b9ded776bc3bf73f5f73e6cbf.

The documentation for the project included google docs link.

After the fixes, all 11 tests pass successfully (see L01). The code coverage is 100%.

Codebase update

After the initial code review, the developers provided us with a new commit: caf3bd36c8baa232c93090392e86d48cfe5533db. This commit was added to the aforementioned pull request. We reviewed the changes to the code and ran tests.

All 15 tests passed successfully. The code coverage is 100%.

Consultation process

We started the consultation on December 11, 2023, and finished it on December 15, 2023.

We inspected the materials provided for the review. Then, we contacted the developers for an introduction to the project. After a discussion, we performed preliminary research and specified those parts of the code and logic that require additional attention during a review:

- The overall correctness of the scripts;
- What is tested in the PR and the presence of testing for the possibility of migration rollback;
- The correctness of the changes to **UpgradeExecutor.sol**;
- The behavior of the system during the process of migration.

During the work, we stayed in touch with the developers and discussed confusing or suspicious parts of the code.

As a result, we verified the following properties of the contracts:

- The correctness of upgradeable and non-upgradeable patterns in **UpgradeExecutor.sol**;
- The safety of the actions;
- The convenience of building batches of transactions in case of emergencies and the possibility of mitigating MEV attacks;
- Whether all contracts with roles are present in the PR;
- The correctness of used role interfaces in the migration scripts;
- Whether any discrepancies with the documentation are present;
- Implemented checks of correctness after the migration is complete;
- What will happen if one of the migration transactions is stuck?

We ran tests and calculated the code coverage.

We combined all the verified issues we found during the manual review in a report.

After the initial consultation, we discussed the results with the developers. On December 19, 2023, the developers provided us with an updated version of the code. In this update, they created atomic scripts for every contract migration and implemented additional tests. Additionally, they resolved all the previously identified issues issues except L03. The code coverage is 100%.

We updated the report, marking new and fixed issues with "new" and "fixed" tags, respectively.

Manual analysis

The tests, migration script and settings files were completely manually analyzed, their logic was checked. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

The consultation showed no critical issues.

Medium severity issues

Medium severity issues can influence project operation in the 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.

M01. Code logic (fixed)

In the current implementation, the **UpgradeExecutor** contract combines two patterns. On one hand, it inherits contracts from the OpenZeppelin **contracts-upgradeable** repository. On the other hand, the developers do not intend to use a proxy with it. We recommend either removing upgradeable contracts from the dependencies or adding an `_disableInitializers()` call to the constructor.

Moreover, in the current implementation, there is a possibility of a frontrun attack. Since the **UpgradeExecutor** contract contains a `delegatecall`, a malicious actor is able to frontrun the initialization transaction, grant themselves `ADMIN_ROLE` and `EXECUTOR_ROLE` roles, and change the storage so that the contract can be initialized one more time.

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

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Complex project setup (fixed)

In the current implementation, the tests are not working properly. TypeScript types are required for project compilation. However, to generate these types, the project needs to be compiled first.

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

L02. Deprecated function (fixed)

The **UpgradeExecutor** contract contains calls to the `_setupRole` function, which is deprecated in favor of the `_grantRole` function, according to OpenZeppelin's documentation.

Furthermore, please note that the `_setupRole` function has been removed since OpenZeppelin version 0.5.0. This could potentially introduce issues with the codebase update.

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

L03. Typo

In the **transfer-ownership.test.ts** file, there is a typo in the function name `chekOwner`. It should likely be corrected to `checkOwner` in the project.

L04. Typo (new)

In the reviewed code, there are several typos:

- The **trasnfer-ownership.test** file has an incorrect name;
- In multiple files, the word `Transfering` is used. The correct spelling is `Transferring`.

L05. Unused imports (new)

The scripts in the **deploy** folder contain unused imports. For example, scripts that transfer ownership of the **Ownable** contracts import functions such as `transferTimelockAdminRole` and `renounceTimelockAdminRole`, which are not used. Similarly, timelock contracts import the `transferSingleContractOwnership` function unnecessarily.

Notes

N01. Checking the migration results

We recommend adding additional security checks for the results of the migration script execution. Such checks might include (but are not limited to) the following ones:

- Call `view` functions in order to check that the roles were migrated successfully;
- Additional checks before the migration on the correctness of the new role addresses.

N02. Rollback script

We recommend implementing a rollback script before migration in case of emergency situations in the project.

N03. Similar action names

In the current implementation, all action functions are named `perform`. This is not an issue; however, since there is no inheritance, we recommend using more explicit names to avoid the potential issue of accidentally calling the wrong action with the same signature.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Pavel Kondratenkov, Senior Security Engineer
Yhtyyar Sahatov, Security Engineer
Konstantin Zherebtsov, Business Development Lead
Irina Vikhareva, Project Manager
Alexander Seleznev, Founder

January 10, 2024