Resolv × PESSIMISTIC

WSTUSR SECURITY ANALYSIS

by Pessimistic

This report is public

September 2, 2024

ABSTRACT

In this report, we consider the security of smart contracts of Resolv 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. A single 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, a security audit is not investment advice.

SUMMARY

In this report, we considered the security of Resolv WstUSR smart contract. We described the audit process in the section below.

The audit showed two issues of medium severity: Discrepancy with ERC4626 and WstUSR can be upgraded. Also, two low-severity issues were found.

After the initial audit, the developers provided a new version of the code. As a result, all issues from the report were either fixed or commented on.

GENERAL RECOMMENDATIONS

We do not have any additional recommendations.

PROJECT OVERVIEW

Project description

For the audit, we were provided with Resolv project on a private GitHub repository, commit 0f6c295e71ca2c95c5584186112a8563d497f0da.

The scope of the audit included **WstUSR.sol**.

The documentation for the project included private notion document and public litepaper.

All 477 tests pass successfully. The code coverage is 97.76%.

The total LOC of audited sources is 210.

Codebase update #1

After the initial audit, the developers provided an updated version of the code: commit 29d56b272398419046229992b3b5594611f723a4. In that update, Discrepancy with ERC4626 issue was fixed. No new issues were found.

All 478 tests passed. The code coverage is 97.76%.

After the review, the code was updated with an insignificant change that brings `wstUSR` and `stUSR` to the same order of magnitude.

Finally, the code was uploaded to a public GitHub repository, commit c131133ae1413bcafeb217d6eac46ac0b94de53f.

AUDIT PROCESS

We started the audit on July 24, 2024 and finished on July 25, 2024.

We inspected the materials provided for the audit. Then, we contacted the developers for an introduction to the project. After a discussion, we performed preliminary research and started the review.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the contracts:

- Whether the ERC4626 standard is implemented correctly;
- Whether shares in the vault are managed correctly;
- Whether there is no possibility of draining the vault.

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- Semgrep rules for smart contracts.

We ran tests and calculated the code coverage.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

After the initial audit, we discussed the results with the developers. On July 31, 2024 the developers provided us with an updated version of the code. In this update, they fixed one issue from our report.

Additionally, the code was audited by another security team in parallel. Two new issues were found. We reviewed the issues and suggested a fix for one of them. As a result, both were fixed in the final commit.

We reviewed the updated codebase and scanned it with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules.

We did not find any issues in the updated version of the code.

We also checked the validity of the code provided in the public repository. More information can be found in the Codebase update #1 section.

Finally, we updated 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 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 audit 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. Discrepancy with ERC4626 (fixed)

The contract incorrectly implements the `totalAssets` function, returning the total amount of `USR` stored for the `StUSR` token. By the ERC4626 standard, the contract must return the amount of underlying asset held by the vault. The current implementation assumes that all `StUSR` are converted to `WstUSR`, which is not valid. The issue can be found in line 47 of the **WstUSR** contract.

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

M02. WstUSR can be upgraded (commented)

The design of the **WstUSR** contract supposes its upgradeability. Thus, some scenarios can lead to undesirable consequences for the project and its users, e.g., if the proxy owner's private keys become compromised. We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

> `Comment from the developers:`
> The developers are planning to transfer an owner role to a multisignature wallet.

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. Gas optimization (commented)

Consider marking `constructor` with the `payable` keyword to reduce gas consumption during the deployment of the **WstUSR** contract.

> `Comment from the developers:`
> Acknowledged.

L02. Immutable variables (commented)

`stUSRAddress` and `usrAddress`, defined at lines 19 and 29 of the **WstUSR** contract, are set only during initialization. Consider making variables as `immutable` and initializing them in the `constructor`.

> `Comment from the developers:`
> Acknowledged.

This analysis was performed by Pessimistic:

Oleg Bobrov, Security Engineer
Evgeny Bokarev, Junior Security Engineer
Konstantin Zherebtsov, Business Development Lead
Irina Vikhareva, Project Manager
Alexander Seleznev, CEO

September 2, 2024