Resolv Security Analysis

by Pessimistic

This report is public

June 5, 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 smart contracts. We described the audit process in the section below.

The initial audit showed several issues of medium severity:
Impossible to call distribute with current protocol fee,
Missing initializations for base contracts, Project Roles. Also, several low severity issues were found.

After the audit, the developers provided us with an updated version of the code. In that update, they fixed most of the issues.

Following the codebase update, the developers introduced several changes. As a result, all issues from the report were either fixed or commented on.

The overall quality of the project is good.

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 74d6560ccfdf6257449312232b32378b0f35072e.

The scope of the audit included:

- **contracts/SimpleToken.sol**;
- **contracts/StUSR.sol**;
- **contracts/AddressesWhitelist.sol**;
- **contracts/beta/ExternalRequestsManagerBetaV1.sol**;
- **contracts/RewardDistributor.sol**.

The documentation for the project included a private notion document.

All 362 tests pass successfully. The code coverage is 98%.

The total LOC of audited sources is 518.

Codebase update #1

After the audit, the developers provided a new version of the code: commit 9ff45f6c1cf7387a40eb577339c6fe20653a5580. In that update, they fixed most of the issues and refactored **stUSR** contract. We added a new L13 issue related to the refactored code.

As a result of the refactoring, the current files were added to the scope:

- **contracts/ERC20RebasingPermitUpgradeable.sol**;
- **contracts/ERC20RebasingUpgradeable.sol**;
- **contracts/interfaces/IERC20Rebasing.sol**.

All 365 tests pass successfully. The code coverage is 84.76%.

Codebase update #2

After the first codebase update, the developers provided us with a commit a1575cdc00cf04cc1f4344f5db268670c093dc2b. In that update, they fixed L13, N03 and N04 issues.

We reviewed the changes. After the review, the code was uploaded to a public GitHub repository, commit 294168806fe2d24add6b6dd4f0c431e00009c65c.

We checked that the contracts, deployed to the addresses from the **README.md** file, match their audited versions. For the proxy contracts, we checked their implementation at the time of writing the report.

Audit process

We started the audit on April 19, 2024 and finished on April 26, 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 specified those parts of the code and logic that require additional attention during an audit:

- Whether unprivileged roles cannot mint tokens directly;
- Whether rewards distribution works with the current restrictions of the project;
- Whether idempotency keys protect from double mints or double burns;
- Whether all the functions use the right modifiers.

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

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 emitted events are accurate;
- Whether tokens correctly implement ERC20 interface;
- Whether functions in **StUSR** contract are overridden correctly;
- Whether there is enough validation for different corner cases.

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 also sent the results to the developers in the text file.

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 May 1, 2024 the developers provided us with an updated version of the code. In this update, they fixed most of the issues from our report, refactored **stUSR** contract and tests accordingly.

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

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

As a result of the manual review, we added a new L13 issue.

After that, we updated the report. Comments from the developers can be found in the respective issue sections.

After the first codebase update, we were provided with a new version of the code. That version contained fixes for L13, N03 and N04 issues. In addition to this, the developers asked us to point out that the deployed contracts match the audited ones, as well as the contracts provided in the new public repository. More information can be found in the Codebase update #2 section.

Manual analysis

The contracts were completely manually analyzed, and 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. Impossible to call distribute with current protocol fee (fixed)

According to the documentation, the current protocol fee is set to zero. This setting implies that `_feeReward` will be zero during the call of `distribute` function of **RewardDistributor** contract. However, the distribution will revert with such input because of the zero check at line 48 in `distribute`.

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

M02. Missing initializations for base contracts (fixed)

Several contracts in the scope do not initialize their base contracts:

- **SimpleToken** does not call base `__AccessControlDefaultAdminRules_init` method, causing `_currentDelay` to be zero;
- **StUST** and **SimpleToken** do not call `__ERC20Permit_init`, leaving default `name` and `version` values.

Consider initializing base contracts with proper values.

The issues have been fixed and are not present in the latest version of the code.

M03. **Project Roles** (addressed)

Several privileged roles are crucial for the correct operation of the protocol:

- Everyone with `SimpleToken.SERVICE_ROLE` can `mint`/`burn` any amount of tokens;
- `DEFAULT_ADMIN_ROLE` in **SimpleToken**, **ExecutorRequestmanagerBetaV1** and **RewardDistributor** is not revoked, allowing to assign other roles;
- Everyone with `SERVICE_ROLE` in **RewardDistributor** contract can call `distribute`, minting any amount of new tokens;
- Owner of `DEFAULT_ADMIN_ROLE` can call `emergencyWithdraw` function of **ExecutorRequestmanagerBetaV1** contract to withdraw any ERC20 token from the contract;
- Owner of **AddressesWhitelist** can add and remove addresses from the whitelist.

In the current implementation, the system depends heavily on these roles. Thus, some scenarios can lead to undesirable consequences for the project and its users, e.g., if private keys of privileged accounts become compromised.

The developers are planning to give `DEFAULT_ADMIN_ROLE` and `Owner` roles 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. Calls without effects (fixed)

The value of `StUSR.totalShares` is not changed after the mint of **SimpleToken**, and the value of `totalSupply` is increased exactly on `_stakingReward` when `distribute` function of **RewardDistributor** contract is called. Consider removing external calls at lines 57–58 of `distribute` function and perform calculations in-place to reduce gas consumption.

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

L02. Code style (fixed)

There are several places within the codebase where some changes can be made to improve the quality of the code in the project:

- In the `removeAllowedToken` function, the address is converted to the same type during the event emission at line 107 of **ExternalRequestsManagerBetaV1** contract;
- The `amount` and `token` arguments have different order in `MintRequestCreated` and `BurnRequestCreated` events;
- Return valued of the `_assertNonZero` function is never used in **RewardDistributor** contract. Consider removing the return statement at line 90 in `_assertNonZero` function.

The issues have been fixed and are not present in the latest version of the code.

L03. Gas optimization (fixed)

The `exists` field can be replaced with `request.provider != address(0)` check to reduce the usage of storage space in **ExternalRequestsManagerBetaV1** contract.

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

L04. Gas optimization (commented)

There is no need to use the `idempotencyKey` for `completeMint` function as the change of the `request.state` to `State.COMPLETED` provides this idempotency. The issue can be found on line 174 in **ExternalRequestsManagerBetaV1** contract.

The developers decided to keep the idempotency logic unified across the project for easier debugging in the future.

L05. Gas optimization (fixed)

In `transferShares` and `transferSharesFrom` functions of **StUSR** contract, the `usrAmount` value is calculated before the actual transfer. Consider calculating it after the successful transfer, as this value will not be needed on fails.

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

L06. Immutable variable (fixed)

`usr` storage variable can be marked as `immutable` and be assigned in the constructor to reduce storage usage at line 29 in **StUSR** contract.

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

L07. Ineffective comparison (fixed)

There are several places where `<=` check is used to compare `uint256` with zero. Consider using `==` for such checks as `uint256` can not be negative. Issues can be found in **RewardDistributor** and **ExternalRequestsManagerBetaV1** contracts.

The issues have been fixed and are not present in the latest version of the code.

L08. Initializers are not disabled (fixed)

Initializers of **StUSR** and **SimpleToken** contracts are not disabled in the constructors. Consider adding calls to `_disableInitializers` function in constructors.

The issues have been fixed and are not present in the latest version of the code.

L09. Possible mint of zero shares (fixed)

The call to `previewDeposit` function may return zero if `usrAmount * (totalShare + 1) < usrBalance + 1`. Consider adding a check that the amount of minted shares is not zero. Otherwise, the user will transfer the deposit but receive nothing at lines 103-104 in `deposit` function of **StUSR** contract.

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

L10. Redundant code (fixed)

There are multiple calls of `safeIncreaseAllowance` function in the code but they have no effect. There is no need to make a self approve to make a direct transfer after. Consider removing the usages of `safeIncreaseAllowance` in **ExternalRequestsManagerBetaV1** contract.

The issues have been fixed and are not present in the latest version of the code.

L11. The actual transfer amount differs from the requested (fixed)

The events at lines 57 and 81 in **StUSR** contract are emitted with the requested amount, which could be larger than the actual transferred amount because of the floor rounding during the convertation to shares.

<u>The issues have been fixed and are not present in the latest version of the code.</u>

L12. Unreachable branch (fixed)

`super.transfer` и `super.transferFrom` calls to **OpenZeppelin** contracts always return `true` on success and revert on failure. Thus, the `false` at lines 56, 66, 80, and 94 in **StUSTR** contract is never returned on transfers as they will revert internally.

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

L13. The actual transfer amount differs from allowance spent (fixed)

In the **ERC20RebasingUpgradeable** contract, in the `transferFrom` function, we recommend decreasing the allowance for the same number of underlying tokens that are emitted in the `Transfer` event.

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

Notes

N01. Disallowed users can not cancel requests (fixed)

It is possible to lose allowance to use the protocol after the creation of the request. Such change would not allow users to cancel their requests and return the deposited funds as `cancelMint` and `cancelBurn` functions have `onlyAllowedProviders` modifier in **ExternalRequestsManagerBetaV1** contract.

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

N02. Potential complexities in management (commented)

Several contracts from the scope will be deployed in two versions: one for `USR` token and one for `RLP` token. Management of two separate versions can produce some additional risks and require more attention to details. Currently, similar parameters, like list of allowed tokens, are managed fully separately for both tokens in the project.

Comment from the developers: Noted, it will be left as is for now.

N03. Minimum receive check (fixed)

It would be a good enhancement for users to provide a value of minimum receiving amount for mints and burns after the request completion. If this value can not be satisfied, user can cancel the request and create a new one with another minimum amount in **ExternalRequestsManagerBetaV1** contract.

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

N04. Possible rounding errors (fixed)

When `StUSR.tranferFrom` function is called the double conversion is made: the first one at line 79 converts `usrAmount` to `shares`, and the second one is made at line 50 during the allowance spending to convert shares to `usr` amount. During both convertations the result is rounded down. The conversion in `_spendAllowance` may result in the requirement of less allowance for some amount of tokens. As a result, more tokens will be transferred for less allowance. Consider rounding up the value in `_spendAllowance` to eliminate such case at line 50 in `_spendAllowance` function of **StUSR** contract.

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

This analysis was performed by Pessimistic:

Pavel Kondratenkov, Senior Security Engineer
Oleg Bobrov, Security Engineer
Konstantin Zherebtsov, Business Development Lead
Irina Vikhareva, Project Manager

June 5, 2024