CRYPTO RECOVERY

Recovery Crypto Security Analysis

by Pessimistic

This report is public

January 18, 2023

Abstract 2
Disclaimer 2
Summary 2
General recommendations 2
Project overview 3
  Project description 3
  Codebase update #1 3
Audit process 4
Manual analysis 5
  Critical issues 5
  Medium severity issues 6
    M01. Reentrancy (fixed) 6
    M02. Overpowered role (commented) 6
    M03. Possible price manipulation attacks (commented) 7
    M04. Lack of documentation (fixed) 7
    M05. Code logic (commented) 7
  Low severity issues 8
    L01. Missing slippage check 8
    L02. Violation of the CEI pattern 8
    L03. Code logic (fixed) 8
    L04. Gas consumption 8
    L05. Indexed arguments (commented) 9
    L06. Missing check (fixed) 9
    L07. Missing check (fixed) 9
    L08. Not updated proposal (fixed) 9
    L09. Bug in vote weight calculation (fixed) 10
  Notes 10
    N01. Inconsistency in the insurance parameter changes (fixed) 10
    N02. Edge case in price calculation formula 10

Abstract

In this report, we consider the security of smart contracts of Recovery Crypto 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 Recovery Crypto smart contracts. We described the audit process in the section below.

The initial audit showed several issues of medium severity: Reentrancy, Overpowered role, Possible price manipulation attacks, Lack of documentation, and Code logic. Also, several low severity issues were found.

The initial implementation of the logic in the **RC** contract was complicated. However, the developers improved the code structure.

After the audit, the developers provided a new version of the code. Most of the issues were not addressed in that update and one new issue of low severity was found.

General recommendations

We recommend fixing the mentioned issues and improving NatSpec coverage. We also recommend implementing CI to run tests, calculate code coverage, and analyze code with linters and security tools.

Project overview

Project description

For the audit, we were provided with Recovery Crypto project on a private GitHub repository, commit 3d1e2012215e17c719a2a61ad784a036d699fa13.

The scope of the audit includes **RC.sol**.

The documentation for the project includes the following link.

All 8 tests pass successfully after our minor fixes. The code coverage is 82.05%.

The total LOC of audited sources is 491.

Codebase update #1

After the initial audit, the codebase was updated. For the recheck, we were provided with commit e3dc9ed5d0750c1495068de2339d8855b78411f2.

This update included fix for the Reentrancy issue, as well as improvements for the code structure. However, most of the issues were not fixed.

All 8 tests pass successfully. The code coverage is 79.7%.

Audit process

The audit began on December 21, 2022, and finished on December 27, 2022.

We inspected the materials provided for the audit. Then, we contacted the developers for an introduction to the project.

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

- Whether the calculation of payment price is correct and safe;
- Whether it is not possible to withdraw someone else's tokens;
- Whether the voting flow depending on the proposal status works properly;
- If it is possible to cheat the protocol and not pay for the subscription;
- Whether it is possible to bypass the limit of the number of withdrawn tokens;
- How a user can exit the project and cancel the subscription.

We ran tests and calculated the code coverage.

Also, we analyzed the project with Slither and manually verified all found occurrences.

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

After the initial audit, we discussed the results with the developers. The developers provided us with an updated version of the code. In this update, they fixed some of the issues from our report, slightly changed validators functionality and improved the code structure. In the update, we found one new issue of low severity.

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 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. **Reentrancy** (fixed)

The CEI (checks-effects-interactions) pattern is violated in `insuranceEvent` function. The storage is updated at line 473 after the external call `safeBatchTransferFrom`. This function from the OpenZeppelin library contains an external call to `msg.sender`. Therefore, a malicious user can migrate more tokens than `recoveryTokenLimit` of the plan since the `rcCount` field of the `Insurance` structure is updated at the end of the function.

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

M02. **Overpowered role** (commented)

In the current implementation, `owner` role is crucial for the project as it is able to do the following:

- Change the penalties that user pays when they use the project without paying in time. Since `paymentPenalty.feeAmountBP` is not limited, `owner` is able to charge an arbitrary fee;

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

- Add and remove subscriptions;
- Change payment admin and fee addresses;

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

- Charge money for the subscription from the users. Since `userInsurance.autopayment` flag value is not checked in `autoPayment` function, the user is not able to cancel their subscription.

  In the latest version of the code, `userInsurance.autopayment` flag prevent `paymentAdmin` from charging money.

In the current implementation, the system depends heavily on the owner. Thus, there are scenarios that can lead to undesirable consequences for the project and its users, e.g., if admin'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 increased power of the owner role is a part of the business logic.*

M03. **Possible price manipulation attacks** (commented)

Consider not relying on pair reserves to avoid incorrect price calculation. Pair reserves can be manipulated by usual swap. The best practice is to use TWAP (Time-Weighted Average Price).

The developers do acknowledge the existence of price manipulation attacks. However, they do not see that as an issue for their project.

M04. **Lack of documentation** (fixed)

The `getPriceInPaymentToken` function converts subscription price from BUSD token to a payment token. In order to calculate the new price, it uses reserves from the pair contract. However, in case when `reserveBase` is zero, the function returns `1`. Since this value does not reflect the real price, we recommend clarifying in the documentation that case.

Moreover, we recommend covering the code with NatSpec comments, as it helps to avoid errors and accelerates the development process.

The developers changed returned value of `1` to a revert statement.

M05. **Code logic** (commented)

There is a difference between the classical subscription systems and the audited one. In the current implementation, there is no possibility to stop subscription and then restart it since the user has to pay for all the time they have not used it. However, classic subscription systems allow suspending them for a time.

The developers commented that the absence of suspension of the subscription is a part of the business logic. The implementation of such logic would prevent user from restoring the wallet in case when subscription is suspended, which is a more serious problem.

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. Missing slippage check

We recommend adding parameter for the maximum amount of payment tokens that user spends since it helps to mitigate price slippage.

L02. Violation of the CEI pattern

The CEI (checks-effects-interactions) pattern is violated in `billPayment`, `autopayment`, `createInsurance`, `upgradeInsurancePlan` functions. Note that mentioned functions are interacting with predefined ERC20 tokens, added by the owner of the contract, hence these interactions are safe. Nevertheless, we still recommend following best practices in order to improve the quality of the code.

L03. Code logic (fixed)

`backupWalletOwners` mapping allows getting a list of backup wallet users by its address. However, this mapping is not updated in `executeProposal` function despite the backup wallet address change.

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

L04. Gas consumption

The gas costs can be optimized in some places of the code:

- `validators[msg.sender]` variable can be read before the loop to reduce gas consumption at line 245 in the `createInsurance` function;

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

- In case of removing the last element of `subscriptions` in the `removeSubscription` function it is possible to reduce gas consumption. Consider checking whether the deleted element is the last one in order to avoid overwriting the same storage slot;

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

- The `bool` and `address` fields inside the `Insurance` structure can be rearranged so that they are in the same storage slot. This allows to reduce gas consumption at lines 248-255 by setting structure via `Insurance({...})`.

L05. **Indexed arguments** (commented)

Marking event arguments as `indexed` is useful when they are considered for further search. The following events have no indexed arguments:

- CreateInsurance;
- UpgradeInsurancePlan;
- BillPayment;
- ProposalCreated;
- Vote;
- ProposalConfirmed;
- ProposalExecuted.

The developers commented that the arguments are not indexed in the events intentionally to save gas.

L06. **Missing check** (fixed)

In `addSubscription` function there is no check that `paymentToken` is not equal to `BUSD`. This may lead to an incorrect price calculation.

The issue has been fixed in commit f94f9bc87206eb4758cb035d3df8ca5d352258e0.

L07. **Missing check** (fixed)

In `createInsurance` function if the user sets `voteQuorum` to 0, `getProposalState` function will not work correctly. Because of this, the wallet owner will not be able to create new proposals for voting.

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

L08. **Not updated proposal** (fixed)

If the insurance owner changes the backup wallet when they change free subscription to unfree, the previous proposal values are still in the storage. That means this proposal can be executed after the mentioned backup wallet change, which may result in an unexpected value.

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

L09. Bug in vote weight calculation (fixed)

According to the comment at line 242, the code supports different voting weight for validators. This weight is accounted correctly in `confirmProposal` function: the more times validator address is stored in `userInsurance.validators` array, the more votes they have. However, this voting weight is not calculated properly in `proposeChangeBackupWallet` function since line 615 clears bit mask every time validator address is met.

Despite the fact that this bug consequences could be mitigated by calling `confirmProposal` function after proposal creation, we recommend fixing it in order to reduce gas costs and improve user experience.

The issue has been fixed in commit afedc8e0f4ca1571aa5e9f5f1b26969be7dd0f0b.

Notes

N01. Inconsistency in the insurance parameter changes (fixed)

In the current version of the code, there is an inconsistency in the insurance parameters that the user can change. On the first hand, the insurance owner fully controls the voting process since they are able to add and remove voters at any time. That means they are able to change backup wallet at any time. On the other hand, the insurance owner cannot change `voteQuorum` when they are using paid plan. That means in order to fully control the voting process, they need to create multiple accounts. We recommend either limiting the ability to change voter list in order to remove power from the insurance owner or allowing them to change vote quorum and backup wallet at any time in order to increase the user experience.

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

N02. Edge case in price calculation formula

If `reserveQuote` is less than `reserveBase`, there might be issues with the formula at line 196. If the ratio is very small, then the price will equal zero and adding one will give an inaccuracy of calculation. We recommend carefully reviewing pairs that will be used in the project.

This analysis was performed by Pessimistic:

Daria Korepanova, Security Engineer
Pavel Kondratenkov, Security Engineer
Yhtyyar Sahatov, Junior Security Engineer
Irina Vikhareva, Project Manager

January 18, 2023