Money on Chain RRC20 Audit

October 1st, 2019

By CoinFabrik

Introduction

CoinFabrik was asked to audit the contracts for the Money on Chain RRC20 project. Firstly, we will provide a summary of our discoveries and secondly, we will show the details of our findings.

Summary

The contracts audited are from the Money on Chain RRC20 repository. The audit is based on the commit 6289770453a95cb1f63999915a6ebccb11335ac8, and updated to reflect changes at 6289770453a95cb1f63999915a6ebccb11335ac8.

We have properly checked the differences with respect to the previously audited contracts. With this information we ensured that the changes introduced to allow a token be the reserve of the project were made correctly. For this reason we will only include the contracts that were subject to changes in the following list, but all contracts were taken into consideration when it comes to possible problems or vulnerabilities. Please see the appendix to get a better view of what changed between the different audits.

The audited contracts are:

- auxiliar/CommissionSplitter.sol: Split balance based on proportion.
- base/MoCReserve.sol: Safe wrapper for RRC20 token calls
- MoCBucketContainer.sol: Storage of RiskProx balances.
- MoC.sol: Project main contract entry point.
- MocEMACalculator.sol: Calculation of exponential moving average.
- MoCRiskProxManager.sol: Manipulation of RiskProx balances.
- MoCBurnout.sol: Burnout queue in case of liquidation.
- MoCInrate.sol: Calculates interest rates.
- MoCConverter.sol: Exchange conversion between used units.
- MoCExchange.sol: Core exchange functionality (Minting/Redeeming)
- MoCHelperLib.sol: Helper functions used by other contracts.
- MoCSettlement.sol: Manages the settlement process.
- MoCState.sol: The core state of the project.

The following analyses were performed:

- Misuse of the different call methods: call.value(), send() and transfer().
- Integer rounding errors, overflow, underflow and related usage of SafeMath functions.
- Old compiler version pragmas.
- Race conditions such as reentrancy attacks or front running.
- Misuse of block timestamps, assuming anything other than them being strictly increasing.
- Contract softlocking attacks (DoS).
- Potential gas cost of functions being over the gas limit.
- Missing function qualifiers and their misuse.
- Fallback functions with a higher gas cost than the one that a transfer or send call allows.
- Fraudulent or erroneous code.
- Code and contract interaction complexity.
- Wrong or missing error handling.
- Overuse of transfers in a single transaction instead of using withdrawal patterns.
- Insufficient analysis of the function input requirements.

Detailed findings

Critical severity

None found

Medium severity

None found

Minor severity

Non-initialized return value

The function `dailyInratePayment` declares a return value but doesn't initialize it. As a result, the default value of 0 will be returned.

```
  function dailyInratePayment() public
  onlyWhitelisted(msg.sender) onlyOnceADay() returns(uint256) {
    uint256 toPay = dailyInrate();
    lastDailyPayBlock = block.number;

    if (toPay != 0) {
      riskProxManager.deliverInrate(BUCKET_C0, toPay);
    }

    emit InrateDailyPay(toPay, mocState.daysToSettlement(),
mocState.getBucketNReserve(BUCKET_C0));
  }
```

If you don't need the return value of the function we recommend removing it. Otherwise it is better to be explicit about the return value of a function, even if it is guaranteed to be 0 by the language semantics.

Observations

Token Trust

Unlike the original MoC project, this one is able to use an arbitrary token instead of RBTC. This implies the project relies on the trust and security of the token itself. Unless properly checked, the token has the power to:

- Deny certain transactions or addresses.
- Substract or change balance arbitrarily.
- Be paused or locked from making transactions.
- Call the MoC contracts back, opening reentrancy attacks.
- Any other action allowed by its own storage and the interfaces it holds

It is also subject to vulnerabilities in itself so it should be audited to ensure maximum confidence.

Conclusion

We found the contracts to be simple and have an adequate amount of documentation. The changes from the original project are straightforward, and just allow the project to be used with an arbitrary token.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the Money on Chain RRC20 project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.

Appendix

We compared current contracts with the ones we previously audited and found the following differences:

New contracts:

- auxiliar/CommissionSplitter.sol
- base/MoCReserve.sol

New functions:

MoC.sol:

- function safeWithdrawFromReserve(address, uint256) internal
- function safeDepositInReserve(address, uint256) private

MoCInrate.sol:

- function calcFinalRedeemInterestValue(bytes32, uint256) public view returns(uint256)
- function calcProportionalInterestValue(bytes32, uint256) internal view returns(uint256)
- function calcFullRedeemInterestValue(bytes32) internal view returns(uint256)

Variable name changes only:

- MoCBucketContainer.sol
- MocEMACalculator.sol
- MoCRiskProxManager.sol
- MoCBurnout.sol
- MoCConverter.sol
- MoCExchange.sol
- MoCHelperLib.sol
- MoCSettlement.sol
- MoCState.sol