Security Assessment Report

RDOC Flux Capacitor - August 23'

Project dashboard

Application Summary:

Name	RDOC Contracts
Version	**Branch:** upgrade-flux-capacitor **Commit:** 8e94548
Language	Solidity

Engagement Summary:

Dates	17 August - 25 August
Reviewers	Ulas Anil Acikel
Level of effort	1 week (working days)

Vulnerability Summary:

Total Critical-Severity Issues	0	
Total High-Severity Issues	0	
Total Medium-Severity Issues	1	▌
Total Low-Severity Issues	0	
Total Informational-Severity Issues	0	
Total	1	▌

Code Maturity Evaluation:

Category Name	Description
Access Control	**Strong.** The functions correctly check which roles can access which functions. Contract management functions can only be called by the governance protocol or the Stopper account which is managed by MoC.
Arithmetic	**Moderate.** Throughout the contract SafeMath library is used. This prevents overflow/underflow attacks. However, the recent changes in the flux capacitor don't use that library. Although highly unlikely to happen, we found some edge cases where integer overflow was possible.
Key Management	Not reviewed.
Specification	**Strong.** Flux capacitor changes and use cases were documented in detail and were provided to us before the review started.
Testing	Not reviewed

Goals

The main goal of this audit is to validate the security of the RDOC minting/redeeming protocol after the changes in the upgrade-flux-capacitor applied.

Coverage

Mainly the code changes in the upgrade-flux-capacitor branch were reviewed. These changes could affect the RDOC minting/redeeming protocol. This was taken into account during the audit. As such, additional code that might be affected by these changes were also reviewed. However, this is not a complete review of the MoC protocol. Test cases and deployment processes weren't reviewed.

Recommendations summary

Short term

- Fix FLUX-01

Long term

- Pseudo-Linear Decay Factor model could be vulnerable to sophisticated attacks such as congesting the protocol. Although this is not an immediate risk, additional reviews could be done to verify solidness of the protocol. Simulating **unusual** transaction traffic could help with testing the model's edge cases.

Findings summary

Following is a list of the findings, with their severity and current status:

ID	Severity		Status
FLUX-01		Medium	Fixed

Findings

FLUX-01 Absolute Accumulator can be inflated and can overflow during minting

ID	Severity		Status
FLUX-01		Medium	Fixed

Affected Assets

contracts/MoCExchange.sol

Fix

The finding is fixed in the following commit:
https://github.com/money-on-chain/RDOC-Contract/commit/8e94548fa92eb191300513f1752a2b60f1ebad08#diff-346093d1d51d830248ff180a181f0985e52f2f7b96c6c434a8e5c51bdb355cc0R442

Description

_updateAccumulatorsOnMint is called inside the mintStableToken during the minting operation. _updateAccumulatorsOnMint is called with user supplied resTokensToMint value which might not be the actual reserve token amount.

```javascript
function mintStableToken(address account, uint256 resTokensToMint,
address vendorAccount)
  public
  onlyWhitelisted(msg.sender)
  returns (uint256, uint256, uint256, uint256, uint256)
 {
  // reverts if not allowed by accumulators
  _updateAccumulatorsOnMint(resTokensToMint);
```

```
StableTokenMintStruct memory details;

// StableTokens to issue with tx value amount
if (resTokensToMint > 0) {
 uint256 resTokenPrice = mocState.getReserveTokenPrice();
                                          details.stableTokens      =
mocLibConfig.maxStableTokensWithResTokens(resTokensToMint,
resTokenPrice); //reserve token to stable token
       details.stableTokenAmount = Math.min(details.stableTokens,
mocState.absoluteMaxStableToken());
              details.totalCost  =  details.stableTokenAmount  ==
details.stableTokens
    ? resTokensToMint
                                                                  :
mocLibConfig.stableTokensResTokensValue(details.stableTokenAmount
, mocState.peg(), resTokenPrice);
... redacted for brevity ...
```

Here the actual reserve token value used depends on the maximum available stable token (mocState.absoluteMaxStableToken()). Even if the user supplies really large resTokensToMint value, the actual reserve token used will be re-calculated according to the available stable tokens.

Since _updateAccumulatorsOnMint uses the user supplied value, a user can overflow the AA calculation if they can mint the max stable tokens. Currently, this value is around $1MM. So the cost of the attack is $1MM but this value can go down depending on the demand on minting stable tokens.

Proof of Concept

N/A. A PoC code can be provided upon request.

Remediation

_updateAccumulatorsOnMint should be called after the final reserve token amount is calculated. In this case the final reserve token value is details.totalCost.

Appendix

Vulnerability Classifications

Severity Categories

Severity	Description
Critical	Vulnerabilities where the exploitation is likely result in a root-level compromise of any of the project components. Exploitation is straightforward in the sense that the attacker does not need any special authentication credentials or additional knowledge or persuade a target user into performing any special functions.
High	The issue affects numerous users and has serious reputational, legal or financial implications.
Medium	Individual users' information is at risk; exploitation could pose reputational, legal or moderate financial risk.
Low	The risk is relatively small or not a risk considered important.
Informational	The issue does not pose an immediate risk but is relevant to security best practices.

Code Maturity Classifications

Code Maturity Classes

Category Name	Description
Access Controls	Related to the authentication and authorization components.
Arithmetic	Related to the proper use of mathematical operations and semantics.
Centralization	Related to the existence of a single point of failure.
Upgradeability	Related to contract upgradeability
Function Composition	Related to separation of the logic into functions with clear purposes.
Key Management	Related to the existence of proper procedures for key generation, distribution, and access.
Monitoring	Related to the use of events and monitoring procedures.
Specification	Related to the expected codebase documentation.
Testing	Related to the use of testing techniques and code coverage.