KUDELSKI SECURITY

MOC V2 Secure Code Review

Findings and Recommendations Report Presented to:

Rif On Chain community with the support of RootstockLabs and mimLABS

May 13, 2024
Version: 1.4

Presented by:

NAGRAVISION SÀRL
Route de Genève 22-24
1033 Cheseaux-sur-Lausanne Switzerland

FOR PUBLIC RELEASE

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Overview

Rif On Chain community with the support of RootstockLabs and mimLABS engaged Kudelski Security to perform a MOC V2 Secure Code Review.

The assessment was conducted remotely by the Kudelski Security App & Blockchain Team. Auditing took place on January 16th, 2024 - February 16th, 2024, and May 10th, 2024, focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the result of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security App & Blockchain Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following are the major themes and issues identified during the testing period. These, along with other items, within the findings section, should be prioritized for remediation to reduce to the risk they pose.

- Missing checks in unpause function
- Missing input validation

During the audit, the following positive observations were noted regarding the scope of the engagement:

- The code is well structured, in a maintainable state, and of production quality.
- Quick and open communication via Slack
- Convenient build and test environment

Scope and Rules Of Engagement

Kudelski Security performed a MOC V2 Secure Code Review for Rif On Chain community with the support of RootstockLabs and mimLABS. This report describes the results of the security audit of the following source code with the commit hash:

Repository and Commit hash:

- stable-protocol-core-v2: https://github.com/money-on-chain/stable-protocol-core-v2/releases/tag/v1.0.2-rc (commit 974a50348b8896e56cba71c0fa551717ada784e2)
- RDOC-Contract: https://github.com/money-on-chain/RDOC-Contract (commit e863445e2df531689fd751914d7ccbaf4f4f995a)

Further changes:

- fix: check protThrld on mintTCandTP (commit 1efd604627dddd12882283f09aeadf218e167fb8)

The goal of the evaluation was to perform a security audit on the source code.

- No additional systems or resources were in scope for this assessment.

In-Scope folders	

├── stable-protocol-core-v2
  ├── contracts
  ├── collateral
  ├── core
  ├── governance
  ├── interfaces
  ├── queue
  ├── tokens
  ├── utils
  ├── vendors
  ├── deploy
  ├── scripts
  ├── hardhat.base.config.ts
	├── RDOC-Contract
	  ├── contracts
	  ├── V2_migration
	  ├── Deprecated.sol
	  ├── MoC_Migrator.sol
	  ├── changers
	  ├── V2MigrationChanger.sol
	  ├── scripts
	  ├── deploy
	  ├── upgrade_v0.2.0
	  ├── 1_deploy_MoC.js
	  ├── 2_deploy_MoCExchange.js
	  ├── 3_deploy_Deprecated.js
	  ├── 4_deploy_Changer.js
	  ├── 5_verification_Changer.js

Table 1: In Scope Folders

While our comprehensive source code review has provided valuable insights into security posture of smart contracts, it is important to point out that this assessment does not guarantee the identification of all potential vulnerabilities, as the constantly evolving nature of the cybersecurity landscape requires ongoing vigilance and adaptation.

Follow-up:

After the initial report (V1.0) was delivered, Rif On Chain community with the support of RootstockLabs and mimLABS addressed all vulnerabilities and weaknesses in the following codebase revision:

- Release v1.0.3 (commit 2bf5a95ca7923b27dc2fe552c4bef56be0e2024e)
- No changes on the RDOC-Contract code base

Further Follow-up:

Rif On Chain community with the support of RootstockLabs and mimLABS released a further update in the following codebase revision:

- stable-protocol-core-v2: Release Candidate v1.0 (commit 333d2b98941adbc045cb085d73b497848ed76061)
- RDOC-Contract: https://github.com/money-on-chain/RDOC-Contract (commit c0edaeba8247a52d566bd87d31329aadbbc01943).

Kudelski Security evaluated these released codebase and no vulnerabilities were identified.

TECHNICAL ANALYSIS & FINDINGS

During the MOC V2 Secure Code Review, we discovered 9 Low severity findings.

The following chart displays the findings by severity.

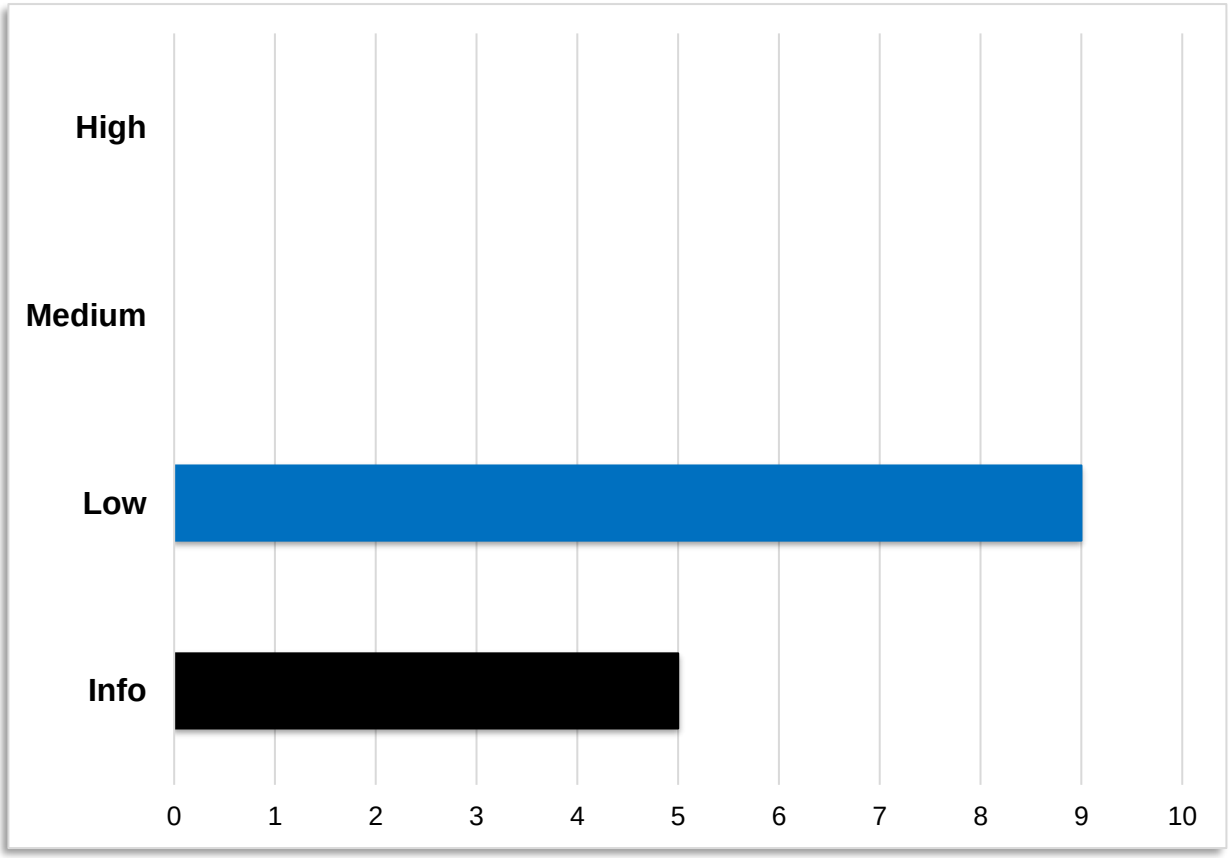

Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

ID	Severity	Description	Status
KS-RSKL-02	**Low**	Missing Check in the unpause Function	Resolved
KS-RSKL-03	**Low**	Mismatch Between Comments and the Implementation	Resolved
KS-RSKL-04	**Low**	Missing Input Validation	Acknowledged
KS-RSKL-05	**Low**	Outdated Dependencies	Resolved
KS-RSKL-06	**Low**	Improper Order of Role Uniqueness Validation and State Modification	Resolved
KS-RSKL-07	**Low**	Unchecked Condition on ctargemaCA and ctargemaTP	Acknowledged
KS-RSKL-08	**Low**	Unchecked Boundary Condition	Resolved
KS-RSKL-09	**Low**	Inconsistent Input Validation Before Division	Acknowledged
KS-RSKL-10	**Low**	Incomplete Check for Redeem Liquidation	Acknowledged
KS-RSKL-11	**Informational**	Validation of Value After Usage	Resolved
KS-RSKL-12	**Informational**	Confusing Variable Naming	Acknowledged
KS-RSKL-13	**Informational**	Unmatched Comment with Calculation	Resolved
KS-RSKL-14	**Informational**	Outdated Solidity Pragma Version	Acknowledged
KS-RSKL-15	**Informational**	Unresolved TOD in Code	Resolved

Table 2: Findings Overview

BUILD AND TEST

Compilation

The stable-protocol-core-v2 binary files was built by the following steps:

1. Create `.env` file
   ```
   $ cp .env.example .env
   ```
2. Install the dependencies
   ```
   $ npm install
   ```
3. Build
   ```
   $ npm run compile
   ```

The compilation passed successfully with 4 warnings, which are not relevant to the security audit.

Tests

A total of 1614 tests are written. The unit test was performed by running the following command:

```
$ npm run test
```

Alternatively, the unit test using Waffle was performed by the following steps:

1. Install Mocha and Chai
   ```
   $ npm install --global mocha
   $ npm install chai
   ```
2. Run the following command
   ```
   $ npx hardhat test
   ```

Due to the slow test platform, the timeout limit value was changed from 100000 to 400000 in hardhat.base.config.ts.

In result, 1614 tests passed successfully out of 1614 tests in total.

Deployment

The solution allows external repositories to define custom network configurations and execute deploys using them. Shell scripts and instructions were provided to deploy the contract on the test network.

```
$ npm run export
```

The command was performed successfully without a warning or an error. However, a fully functional deployment on testnet is out of scope of this audit.

STATIC ANALYSIS

NPM Audit

NPM audit (v9.5.1) identified 5 findings, which are relevant to the outdated dependencies.

Slither

Slither (v0.10.0) analyzed 102 contracts with 91 detectors on stable-protocol-core-v2 by the command below:

```
npm run security-default
```

In result, Slither identified 164 findings. Among those, only meaningful findings are reported here.

Mythril

Mythril (v0.24.1) analyzed 102 contracts with 91 detectors on stable-protocol-core-v2 by the command below:

```
npm run mythril:MocCACoinbase
```

```
npm run mythril:MocCARC20
```

```
npm run mythril:MocCoreExpansion
```

```
npm run mythril:MocVendors
```

```
npm run mythril:MocTC
```

```
npm run mythril:MocQueue
```

In result, Mythril did not identify any finding.

Semgrep

Semgrep (v1.20.0) was performed on Solidity code by the following command:

```
semgrep --config "p/smart-contracts" ./contracts/
```

In result, Semgrep ran 49 rules and identified 29 findings on smart contracts under stable-protocol-core-v2, and 2 findings on smart contracts under RDOC-Contract.

MANUAL CODE REVIEW

Note that the code review for ROC V2 has been done, together with MOC V2 code base. Since the findings are overlapped with those in the MOC V2 code review, the IDs of findings are not produced separately, rather, taken from the audit report of MOC V2. For details of the findings, please refer to “ROC V2 Secure Code Review, v1.3, April 8, 2024”.

KS-RSKL-02 – Missing Check in the unpause Function

Severity	**Low**
Status	Resolved

Impact	Likelihood	Difficulty
High	Low	High

Description

The contract `Stoppable.sol` has emergency `pause` and `unpause` functions. While, the `pause` function can only be called if `stoppable` is set to true, there is no such check in the `unpause` function.

KS-RSKL-03 – Mismatch Between Comments and the Implementation

Severity	**Low**
Status	Resolved

Impact	Likelihood	Difficulty
Low	Low	High

Description

`makeUnstoppable` and `makeStoppable` functions in the Stoppable contract suggest that these functions also affect the `_paused` state of the contract. However, the implementation of these functions only changes the stoppable state.

KS-RSKL-04 – Missing Input Validation

Severity	**Low**
Status	Acknowledged

Impact	Likelihood	Difficulty
High	Low	High

Description

In Solidity, checking for `address(0)` is essential primarily to prevent the accidental loss of assets, as it represents the default value for uninitialized addresses and transactions to it are irrecoverable. It also ensures that addresses are intentionally set, guarding against errors or uninitialized values in smart contracts. It is observed that in contracts `MocUpgradable.sol, Stoppable.sol and Governed.sol` there are missing validity checks for `address(0)` in the `MocUpgradable_init(), setPauser() and changeGovernor()` functions respectively.

KS-RSKL-05 – Outdated Dependencies

Severity	**Low**
Status	Resolved

Impact	Likelihood	Difficulty
High	Low	High

Description

Some of the dev dependencies are outdated as they are reported to be vulnerable: CVE-2023-40014, CVE-2023-45857, and CVE-2023-26159.

KS-RSKL-06 – Improper Order of Role Uniqueness Validation and State Modification

Severity	**Low**
Status	Resolved

Impact	Likelihood	Difficulty
Low	Low	High

Description

The `transferAllRoles` function in the smart contract `MocRC20.sol` function transfers the `PAUSER_ROLE` from the sender to a new account but checks for role uniqueness after state changes, leading to potential unnecessary gas costs. If the role isn't unique, the transaction reverts, incurring costs despite reversal. This design may cause inefficient gas usage and unintended effects from premature state modifications.

KS-RSKL-07 – Unchecked Condition on ctargemaCA and ctargemaTP

Severity	**Low**
Status	Acknowledged

Impact	Likelihood	Difficulty
Medium	Low	High

Description

In the function `redeemTCandTPto`, `ctargemaCA` and `ctargemaTP` needs to be bigger than ONE. However, such condition is not explicitly checked before they are used. Furthermore, the function `_calcCtargemaCA` may return `protThrld*2` if `den == 0`, which leads a condition that `protThrld > 0.5`. Although this condition is obvious, it needs to be explicitly checked to avoid setting a wrong value by mistake.

KS-RSKL-08 – Unchecked Boundary Condition

Severity	**Low**
Status	Resolved

Impact	Likelihood	Difficulty
Low	Low	High

Description

The function `_checkLessThanOne` checks `value_` is less than `PRECISION ONE`, but does not revert when `value_` is equal to `PRECISION ONE`.
Also, the function `_MocBaseBucket_init_unchained` checks `protThrld` is less than `ONE`, but does not revert when `protThrld` is equal to `ONE`.

KS-RSKL-09 – Inconsistent Input Validation Before Division

Severity	**Low**
Status	Acknowledged

Impact	Likelihood	Difficulty
High	Low	High

Description

The function `swapTCforTPto` does not check whether `nTCcb` is equal to zero before the division is executed, whereas the function `_getPTCac` checks the denominator and return ONE if the zero division is expected.

KS-RSKL-10 – Incomplete Check for Redeem Liquidation

Severity	**Low**
Status	Acknowledged

Impact	Likelihood	Difficulty
Low	Low	High

Description

The function `redeemTCandTPto` does not check `qTC / qTP > prop` as in the pseudo code on the white paper (page 23).

KS-RSKL-11 – Validation of Value After Usage

Severity	Informational
Status	Resolved

Description

If `qACtotalToRedeem` is equal to zero, it should be reverted before usage.

KS-RSKL-12 – Confusing Variable Naming

Severity	**Informational**
Status	Acknowledged

Description

Variable naming is an important aspect in making your code readable. However, the naming convention used in the entire codebase does not provide the clear readability nor intuition on the variables and functions.

Reference

- Variable Naming Conventions: https://curc.readthedocs.io/en/latest/programming/coding-best-practices.html#variable-naming-conventions

KS-RSKL-13 – Unmatched Comment with Calculation

Severity	Informational
Status	Resolved

Description

In the function `_calcFees`, the comment is not matched with the code implementation. the total token is adjusted by multiplying the precision factor but it is not implemented in the code.
In the function `_calcCtargemaCA`, the constant `PRECISION` is multiplied twice since `_divPrec` itself also executes the multiplication of `PRECISION`.

KS-RSKL-14 – Outdated Solidity Pragma Version

Severity	Informational
Status	Acknowledged

Description

Multiple contracts in RDOC-Contract repository use an outdated solidity pragma version 0.5.8 which has some known bugs. Also, ABI encoder and Yul compiler have been upgraded significantly in the version 0.6.0 upward.

KS-RSKL-15 – Unresolved TODO in Code

Severity	Informational
Status	Resolved

Description

There is a TODO comment in the function _calcQACforRedeemTCandTP, which should be resolved before the code release. In particular, there is no clear explanation how the code in the TODO comment could replace the current implementation and improve the function against the rounding error.

CONCLUSION

During the MOC V2 Secure Code Review, multiple vulnerabilities and weaknesses were identified in the codebase, and those vulnerabilities and weaknesses were all addressed or acknowledged by the Rif On Chain community with the support of RootstockLabs and mimLABS in the follow-up revision of the codebase.

METHODOLOGY

During this source code review, the Kudelski Security Services team reviewed code within the project within an appropriate IDE. During every review, the team spends considerable time working with the client to determine correct and expected functionality, business logic, and content to ensure that findings incorporate this business logic into each description and impact. Following this discovery phase, the team works through the following categories:

- Key / Secrets handling
- Error handling and logging
- Handling of exception / boundary condition
- Nonce and randomness
- Countermeasures against known vulnerabilities
- Input validation
- Logical flaws
- Authentication
- Code practice

Tools

The following tools were used during this portion of the test. A link for more information about the tool is provided as well.

- Visual Studio Code
- Slither
- Mythril
- Echidna
- NPM audit

Vulnerability Scoring Systems

Kudelski Security utilizes a vulnerability scoring system based on impact of the vulnerability, likelihood of an attack against the vulnerability, and the difficulty of executing an attack against the vulnerability based on a high, medium, and low rating system.

Impact
The overall effect of the vulnerability against the system or organization based on the areas of concern or affected components discussed with the client during the scoping of the engagement.

> **High:**
> The vulnerability has a severe effect on the company and systems or has an effect within one of the primary areas of concern noted by the client
>
> **Medium:**
> It is reasonable to assume that the vulnerability would have a measurable effect on the company and systems that may cause minor financial or reputational damage.
>
> **Low:**
> There is little to no effect from the vulnerability being compromised. These vulnerabilities could lead to complex attacks or create footholds used in more severe attacks.

Likelihood
The likelihood of an attacker discovering a vulnerability, exploiting it, and obtaining a foothold varies based on a variety of factors including compensating controls, location of the application, availability of commonly used exploits, and institutional knowledge

> **High:**
> It is extremely likely that this vulnerability will be discovered and abused
>
> **Medium:**
> It is likely that this vulnerability will be discovered and abused by a skilled attacker
>
> **Low:**
> It is unlikely that this vulnerability will be discovered or abused when discovered.

Difficulty
Difficulty is measured according to the ease of exploit by an attacker based on availability of readily available exploits, knowledge of the system, and complexity of attack. It should be noted that a LOW difficulty results in a HIGHER severity.

> **Low:**
> The vulnerability is easy to exploit or has readily available techniques for exploit
>
> **Medium:**
> The vulnerability is partially defended against, difficult to exploit, or requires a skilled attacker to exploit.
>
> **Low:**
> The vulnerability is difficult to exploit and requires advanced knowledge from a skilled attacker to write an exploit

Severity
Severity is the overall score of the weakness or vulnerability as it is measured from Impact, Likelihood, and Difficulty

REFERENCES

- Rif On Chain Stablecoin Protocol Collateralized with RIF, Technical whitepaper release 2, revision 1, December 2023
- MOC main protocol
  - https://github.com/money-on-chain/stable-protocol-core-v2/tree/master/docs
- ROC V1 to V2 Migration plan overview
- RoC Stable Platform Wiki
  - https://docs.moneyonchain.com/rdoc-contract/