ROC V2 Secure Code Review

Findings and Recommendations Report Presented to:

Rif On Chain community with the support of RootstockLabs

May 13, 2024
Version: 1.4

Presented by:

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

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Overview

Rif On Chain community with the support of RootstockLabs engaged Kudelski Security to perform a ROC 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 Countermeasure Against Oracle Manipulation
- Outdated Dependencies

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 ROC V2 Secure Code Review for Rif On Chain community with the support of RootstockLabs. This report describes the results of the security audit of the following source code with the commit hash:

Repository and Commit hash:

- stable-protocol-roc-v2: https://github.com/money-on-chain/stable-protocol-roc-v2/releases/tag/v1.0.2-rc (commit e4e3159aa61069fc8f1d44c8095294d3464cf2a5)

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-roc-v2`	
`    ├── contracts`	
`        ├── MocRif.sol`	
`        ├── interfaces`	
`            ├── IDataProvider.sol`	
`        ├── providers`	
`            ├── CMaxAbsoluteOpProvider.sol`	
`            ├── FCMaxOpDifferenceProvider.sol`	
`    ├── deploy`	
`            ├── deploy_MocRif.ts`	
`            ├── deploy_MocRifExpansion.ts`	
`            ├── migrate_MocRif.ts`	
`    ├── scripts`	
`            ├── utils.ts`	
`    ├── hardhat.base.config.ts`	

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 addressed all vulnerabilities and weaknesses in the following codebase revision:

- v1.0.3-rc (commit 0d5f33032f33e6f56eef4be62d6cba10df63944d)

Further Follow-up:

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

- stable-protocol-roc-v2: Release Candidate v1.0.6 (commit 454f108b7643e9ea5872055a19a413bd19d735af)

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

TECHNICAL ANALYSIS & FINDINGS

During the ROC V2 Secure Code Review, we discovered 1 finding each with medium and low severity.

The following chart displays the findings by severity.

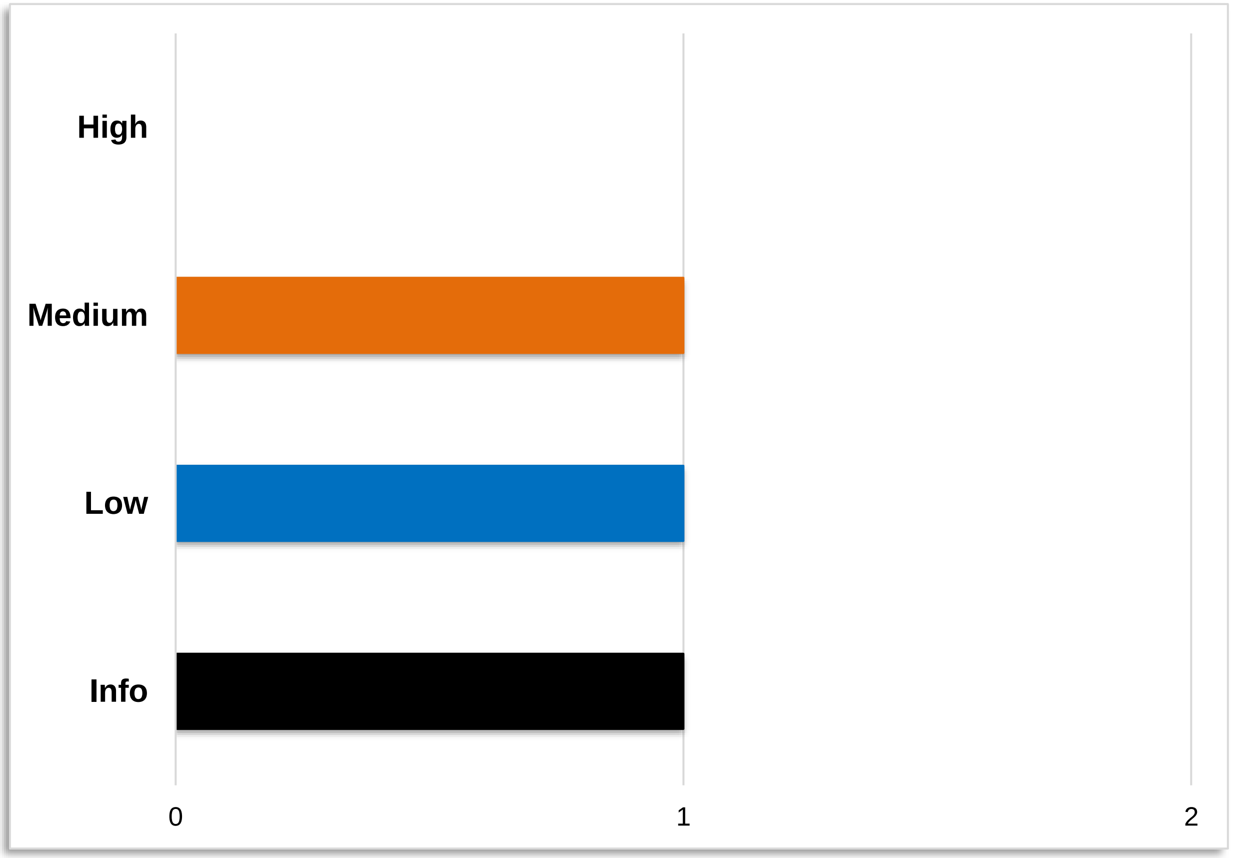

Figure 1: Findings by Severity

Findings

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

ID	Severity	Description	Status
KS-RSKL-01	Medium	Missing Countermeasure Against Oracle Manipulation	Acknowledged
KS-RSKL-05	Low	Outdated Dependencies	Resolved
KS-RSKL-12	**Informational**	Confusing Variable Naming	Acknowledged

Table 2: Findings Overview

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 “MOC V2 Secure Code Review, v1.2, Feb. 2024”.

BUILD AND TEST

Compilation

The stable-protocol-roc-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.

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 deploy-rskTestnet
```

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.

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 4 findings on smart contracts under stable-protocol-roc-v2.

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 "MOC V2 Secure Code Review, v1.3, April 8, 2024".

KS-RSKL-01 – Missing Countermeasure Against Oracle Manipulation

Severity	Medium
Status	Acknowledged

Impact	Likelihood	Difficulty
High	Low	High

Description

The price of asset is determined by the price/data from a single oracle through the `IPriceProvider` and `IDataProvider` interfaces. Also, there is no lower/upper limit of price, compared with the previous price. This is risky due to the possibility of oracle price manipulation by attackers. It is also to note that Amphiraos-Oracle is an internal oracle developed by RSKL labs based on MakerDao Medianizer and PriceFeed solution.

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-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

CONCLUSION

During the ROC V2 Secure Code Review, a vulnerability was identified in the codebase, and this vulnerability was addressed by the Rif On Chain community with the support of RootstockLabs 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

- MOC V2 Secure Code Review, v1.2, Feb. 2024
- 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/
- Rif On Chain Stablecoin Protocol Collateralized with RIF, Technical whitepaper release 2, revision 1, December 2023