QuillAudits

Audit Report
September, 2023

For

IOVlabs

Table of Content

Table of Content

Executive Summary

Project Name	RDOC-Contract-Internal
Overview	A fork of the Money-on-Chain protocol.
Timeline	25th May, 2023 - 8th July, 2023
Method	Manual Review, Functional Testing, etc.
Audit Scope	The scope of this audit was to analyze RDOC-Contract-Internal's codebase for quality, security, and correctness. *https://github.com/money-on-chain/RDOC-Contract-Internal*
Branch Name	v0.1.16-1.0
Commit Hash	*5aee9e9055A.23dcde01297117e441544f64ac8*
Fixed In	*https://github.com/money-on-chain/RDOC-Contract-Internal/pull/32/commits*
Commit Hash	*bcb8c77*
Please Note	The links mentioned under "Audit Scope" and "Fixed In" are in a Private GitHub repository. However, the IOV Labs team has relocated the Audited code to the Public Repo listed below:- https://github.com/money-on-chain/RDOC-Contract/tree/feature/audited-stable-token-migration/contracts It is important to note that in public repository, minor changes has been done to MoC.sol to account for gas prices as described in the proposal here: https://github.com/money-on-chain/main-RBTC-contract/pull/114

Contracts in Audit Scope

- MoC.sol
- MoCBucketContainer.sol
- MoCEMACalculator.sol
- MoCExchange.sol
- MoCHelperLib.sol
- MoCInrate.sol
- MoCLibConnection.sol
- MoCRiskProxManager.sol
- MoCSettlement.sol
- MoCState.sol
- MoCVendors.sol
- PartialExecution.sol
- MoCToken.sol
- OwnerBurnableToken.sol
- RiskProToken.sol
- StableToken.sol
- MoCConnector_v0116.sol
- MoCExchange_v0116.sol
- MoCSettlement_v0116.sol
- MoCState_v0116.sol
- MoC_v0116.sol
- CommissionSplitter.sol
- MoCBase.sol
- MoCConnector.sol
- MoCConstants.sol
- MoCReserve.sol
- MoCWhitelist.sol
- StableTokenMigrationChanger.sol
- MocRC20.sol
- StableTokenV2.sol
- TokenMigrator.sol
- Governod.sol

Number of Security Issues per Severity

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	2	4
Partially Resolved Issues	0	1	0	0
Resolved Issues	2	2	1	1

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Hardhat, Solhint.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

High Severity Issues

A.1: Incorrect Calculations | MoCVendors.sol

Description

The `totalMocAmount` variable in updatePaidMarkup is assigned the user-defined value `mocAmount` but is overriden on **L200**. The initial value stored (mocAmount) gets changed to the result of the calculation performed. If this is the desired functionality then the input parameter can be discarded as well as **L193**.

Remediation

Update **L200** to allow the value stored in totalMocAmount be incremented with the calculated value instead of being overridden.

```
totalMoCAmount += reserveTokenPrice.mul(resTokenAmount).div(mocPrice);
```

Status

Resolved

IOV Team's Comment

mocAmount is only overridden in the case "(resTokenAmount > 0)" which is correct given that you can either pay with mocTokens or reserve, not both. From the coverage tool, the "if" path is taken ~300 times while the other ~500, so yes, both scenarios are tested. I can paste an image with that info.

Auditor's Remark

Closed as a false positive.

A.2: Contract Initialization

Description

An attacker can take over the implementation contract if not properly initialized. Almost all of the contracts in the codebase are missing the check that restricts the implementation contract initialization. Some of the contracts listed below

- MoC.sol
- MoCExchange.sol
- MoCRiskProxManager.sol
- MoCSettlement.sol

A.2: Contract Initialization

If the logic contract is not initialized then an attacker can takeover the implementation contract by initializing the implementation contract. More details of the attack can be read from **here**.

Remediation

_disableInitializers() should be used in the constructor to avoid initialization of the implementation contract.

Status

Resolved

IOV Team's Comment

The deployment process will be automatic using scripts to mitigate frontrunning risk.

Medium Severity Issues

A.3: Whitelist Addresses | MoCConnector_v0116.sol

Description

The `migrateStableToken(..)` function has the previous stableToken address removed (L23) from the whitelist but it does not add the new address to the whitelist.

Remediation

Add the new address to the whitelist to have the same privileges as the old one.

Status

Resolved

IOV Team's Comment

It is not necessary to add the stable token to the whitelist because it doesn’t call any contract of the solution with the necessity of special permissions. We don't see this as an issue.

A.4: Outdated Libraries

Description

The codebase is replete with old packages from libraries that have gotten updates and newer releases. If the older versions are still in use and they contain bugs, then the entire codebase would be opened up to a larger attack surface than necessary.

```
...
import "openzeppelin-solidity/contracts/math/SafeMath.sol";
import "moc-governance/contracts/Governance/Governed.sol";
...
```

Remediation

Use tested newer libraries and packages when available. Solidity 0.8.0 also has internal checks for overflows and underflows, thereby reducing the need for external libraries which in turn would cause lesser contract sizes, deployment costs, and lower fees for users per transaction.

IOV Team's Remarks

We agree that it's a good practice to work with the latest tools and libraries, or at least follow the fixes of your selected mayor's versions for each one. But we believe that deployed smart contracts are a special case; as the risk of a change (unless you fix a known vulnerability) outweighs the possible benefit of an update.
The changes introduced by USDRIF upgrade where indeed made on a separate folder that uses a new toolset with the latest stable version available. Applying this backward to all existing code simply doesn't make sense.

Status

Partially Resolved

A.5: Two-step Ownable

Description

OwnerBurnableToken inherits the Ownable library which has been improved via two-step ownable to provide a more seamless method of ownership transfer by allowing the new user to accept or reject the new role.

Remediation

Implement two-step ownable instead of single-step ownership transfer.

A.5: Two-step Ownable

References

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol and
https://github.com/razzorsec/RazzorSec-Contracts/blob/main/AccessControl/SafeOwn.sol

Status

Resolved

IOV Team's Comment

This is not a security issue: Any change to the protocol, including governor changes, needs to go through the proposal, audit and voting process. The proposal itself must include the new governor address and everyone can verify it, it should be a requirement for the proposal to be verifiable actually, and provide proof that the new Governor actually has capabilities. So we don't think making this part of the protocol is necessary, as the same "goal" of Ownable2StepUpgradeable can be achieved by doing that verification step on the changer, and not in the protocol itself, following the principle of externalizing not core functionalities from it.

Even if it is necessary to prevent an error in the address of the future governor, instead of adding a claim function, in the changer we could verify it by calling some external function to check that it exists in the Governor contract, for example, isAuthorizedChanger() should return false and not revert.

Low Severity Issues

A.6: Unit Testing

Description

It is highly recommended to have above 95% of code functionality tested before going into production so as to catch bugs that could be introduced from user input as well as return values from function calls. Some tests have not been implemented, if issues persist consider using more recent frameworks like Hardhat or Foundry.

Remediation

Increase the coverage of unit tests in the codebase to adequately test for all branches in code.

Status

Acknowledged

IOV Team's Comment

The current test coverage is 82%.

A.7: Unfixed TODO in Code | MoCBucketContainer.sol

Description

The codebase contains a TODO which has not been implemented in setBucketCobj() on **L85**.

Remediation

It is recommended that mature codebases do not have TODOs in their code. Implement the logic described in the TODO.

Fixed In

https://github.com/money-on-chain/RDOC-Contract-Internal/pull/33 [pending]

Status

Resolved

A.8: Older Solidity Version Used

Description

Multiple solidity pragma versions are used in this codebase with majority of the contracts on version 0.5.8 which has some known bugs fixed in newer versions from **0.6.0** upward. The SMTChecker, Yul compiler and ABI encoder have received significant upgrades from that time till now.

Remediation

It is advised to use more recent versions when deploying to production to reduce the surface of attacks and future-proof the codebase as much as possible.

Status

Acknowledged

Informational Issues

A.9: Convert Modifier to Internal Function | MoCBucketContainer.sol

Description

The `bucketStateUpdate()` **modifier** can be implemented as an internal function instead of a modifier. It should only emit the latest state after the update.

Remediation

Convert the modifier to an internal function and adjust its occurrences in the codebase.

Status

Acknowledged

A.10: Untracked State Changes | MoCBucketContainer.sol

Description

The `payInrate()` function on **L190** does not emit an event after updating the bucket state (using the bucketStateUpdate modifier).

Remediation

For consistency and ease of tracking with monitoring software, ensure that changes to state are marked with event emissions.

A.10: Untracked State Changes | MoCBucketContainer.sol

Status

Acknowledged

IOV Team's Comment

We don't see this as a security issue. This should be informational.

Auditor’s Remark

If it would not affect any calculable/reported values, it can be downgraded.

A.11: Unused Variables

Description

Some variables in the codebase are not used and can be discarded. The bool `initialized` in **MoCBase** and **MoCConnector** are unused.

Remediation

Remove unused variables.

Status

Acknowledged

IOV Team's Comment

Removing a variable, even if unused, can generate a hole in the storage that needs to be compensated with the gap, or garbage slot filling. The risk of doing it massively outweighs the benefits of removing a harmless variable.

A.12: Wrong Comments

Description

On **L114** in MoCBucketContainer, the comment for the function getActiveAddressesCount() should be ‘returns the number of addresses...’.

On **L45** in MoCEMACalculator, the comment for the function setEmaCalculationBlockSpan() should use ‘EMA’ instead of ‘BMA’.

On **L131**, the return comment for the function unregisterVendor() shows a return value of false for both scenarios.

A.12: Wrong Comments

On **L172**, the comment for the function executeTask() should be 'executes the task'.

Remediation

Fix the comment-code relationships as described above.

Fixed In

https://github.com/money-on-chain/RDOC-Contract-Internal/pull/33 [pending]

Status

Resolved

A.13: Naming Convention

Description

The naming convention used in the codebase is not clear and reduces the readability of the entire codebase.

Remediation

Follow industry standard naming conventions and practices to make code easier to read.

Status

Acknowledged

Functional Tests

Test	Expectation	Actual	Verdict
Deploy and initialize contracts	PASSED	PASSED	PASSED
Deploy libraries and inter-contract dependencies	PASSED	PASSED	PASSED
Upgrade contracts and change implementation afterward	PASSED	PASSED	PASSED
MoCConnector_v0116: Add new token address to the whitelist	FAILED	FAILED	PASSED

Closing Summary

In this report, we have considered the security of the IOV Labs codebase. We performed our audit according to the procedure described above.

Some issues of High, Medium, Low, and Informational severity were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

The IOV Labs team implemented newer functionality that allows to set gas prices as described in this **proposal** to mitigate (not completely eliminate) possible risks to users.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in IOV Labs smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of IOV Labs smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the IOV Labs to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

850+
Audits Completed

$30B
Secured

$30B
Lines of Code Audited

Follow Our Journey

Audit Report
September, 2023

For

IOVlabs

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com