Audit Report

Mars Rover v2

v1.0

Sep 14, 2023

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT IS ADDRESSED EXCLUSIVELY TO THE CLIENT. THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THE CLIENT OR THIRD PARTIES.

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Delphi Labs, Ltd. to perform a security audit of the Mars Rover v2 smart contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following targets:

Repository	https://github.com/mars-protocol/rover
Commit	`6013cbca21a343bbea695aac6b178c7be83948b2`
Scope	This audit covers the changes since our previous audit, which was performed on commit `d58f03cbdeeacc87226526b5e7c202ab989883d9`
Identifier	In this report, all paths pointing to this repository are prefixed with `rover:`
Fixes verified at commit	`3b7dbec0d2b41efedd0477c5381bdc3964e95a6a` Note that changes to the codebase beyond fixes after the initial audit have not been in scope of our fixes review.

Repository	https://github.com/mars-protocol/red-bank
Commit	`1b4b3f7feebcf0e228cb92ba06ac5214437a53fe`
Scope	This audit covers the changes since our previous audit, which was performed on commit `7149f580bf6f7593d2ccaa4c09a2d63dc482d5ff`
Identifier	In this report, all paths pointing to this repository are prefixed with `red-bank:`
Fixes verified at commit	`c810de1b737ef603d735734bceb2dc6af1f73462` Note that changes to the codebase beyond fixes after the initial audit have not been in scope of our fixes review.

Repository	https://github.com/mars-protocol/red-bank
Commit	`182ff1fe38bf81efd7608770051ca96395d6415b`
Scope	This audit covers the migration approach implemented in pull request #317, which includes the commits from `b4ca20fe044b41b755884659dd29de8cb73dcd96` to `182ff1fe38bf81efd7608770051ca96395d6415b`. This is intended to support Osmosis migration from version 1.0.0 of the `incentives` contract.
Identifier	In this report, all paths pointing to this pull request are prefixed with `incentive-migration:`

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Mars Protocol is a multi-chain money market built on CosmWasm that leverages the Cosmos ecosystem's interoperability and composability. The audit scope includes updates to Mars Rover and Red Bank smart contracts since our previous audit.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium-High	Most actions in the credit manager contract integrate with the red bank contract, increasing the overall complexity. Furthermore, most contracts are updated to support having multiple account positions created by the `account-nft` contract.
Code readability and clarity	Medium-High	-
Level of documentation	High	-
Test coverage	Medium-High	-

Summary of Findings

No	Description	Severity	Status
1	Lendings in the red bank cannot be liquidated when `ActionKind::Default` circuit breakers trigger	Critical	Resolved
2	Incorrect protocol fee calculation during liquidation	Major	Resolved
3	Incorrect contract name assertion prevents successful migration	Major	Resolved
4	Incomplete state migration in `account-nft` contract	Major	Resolved
5	Circuit breakers may block liquidations, risking the protocol's solvency	Major	Resolved
6	Inaccurate health computation due to default collateral limit	Major	Resolved
7	Credit manager migration requires the reward collector `account-id`	Minor	Resolved
8	Contract version is not updated	Minor	Resolved
9	Users cannot exit vaults for assets that are no longer whitelisted	Minor	Resolved
10	Credit manager contract is not defined when instantiating the health contract	Minor	Resolved
11	Insufficient validation of the interest rate module	Minor	Resolved
12	Insufficient price source validation for liquidity token pricing	Minor	Acknowledged
13	Potential failure of `incentives` contract migration due to unbounded iteration	Informational	Acknowledged
14	Conflict between documentation and implementation of the borrow rate calculation	Informational	Resolved
15	Use of hardcoded versions during migration	Informational	Resolved
16	Inefficient execution of messages	Informational	Resolved
17	Use of custom logic instead of standard libraries	Informational	Acknowledged

18	`update_nft_config` emits the same attribute value as `update_config`	**Informational**	**Resolved**
19	`get_route` function can be used to reduce code complexity	**Informational**	**Resolved**
20	`WHITELIST_COUNT` is not exposed through the configuration query	**Informational**	**Resolved**
21	Missing maximum validation in `query_all_asset_params` and `query_all_vault_configs` queries	**Informational**	**Resolved**

Detailed Findings

1. Lendings in the red bank cannot be liquidated when `ActionKind::Default` circuit breakers trigger

Severity: Critical

In `rover:contracts/credit-manager/src/liquidate_lend.rs:43-45`, the `liquidate_lend` function withdraws the liquidatee's asset lent in the red bank as part of the liquidation process. When a liquidation occurs, price queries are dispatched as `ActionKind::Liquidation` to ensure liquidations can still be executed during extreme market conditions. This design differs from `ActionKind::Default`, as the `max_confidence` and `max_deviation` circuit breakers will trigger and revert the transaction, possibly hindering the liquidation process.

However, the red bank contract performs an `ActionKind::Default` price query during withdrawals. In `red-bank:contracts/red-bank/src/withdraw.rs:77-86`, the `assert_below_liq_threshold_after_withdraw` function internally calls `get_user_positions_map` with `is_liquidation` parameter as `false`, resulting in the asset price query being executed as `ActionKind::Default` instead of `ActionKind::Liquidation`.

Consequently, liquidating lent positions in the red bank contract will fail when one of the `max_confidence` or `max_deviation` circuit breakers is triggered, potentially causing bad debt to accrue and risking the protocol's solvency.

Recommendation

We recommend modifying the red bank contract to use the `ActionKind::Liquidation` price query when the credit manager contract performs a withdrawal during liquidations.

Status: Resolved

2. Incorrect protocol fee calculation during liquidation

Severity: Major

In `red-bank:packages/liquidation/src/liquidation.rs:100`, the `calculate_liquidation_amounts` function computes the liquidation bonus amount by multiplying the `collateral_amount_to_liquidate` value with the liquidation bonus percentage. This is incorrect because the `collateral_amount_to_liquidate` value had already the liquidation bonus applied in line `84`.

Consequently, the protocol fee amount computed in line `103` will be inaccurate because it uses an incorrect liquidation bonus value, causing the protocol to receive erroneous fee amounts.

The client has independently detected this issue during this audit.

Recommendation

We recommend calculating the protocol fee without double computing the liquidation bonus.

Status: Resolved

3. Incorrect contract name assertion prevents successful migration

Severity: Major

In `rover:contracts/account-nft/src/contract.rs:94`, the `migrate` function does not prepend the "crates.io:" identifier to the `CONTRACT_NAME` when performing a contract migration. This is problematic because the previously deployed version and the contract instantiation process prepends that identifier to the contract name. Consequently, migrations will fail because they expects the contract name to not include that identifier.

This issue is also present in the following contracts:

- `rover:contracts/credit-manager/src/migrations/helpers.rs:13`
- `red-bank:contracts/oracle/osmosis/src/migrations.rs:13`

Recommendation

We recommend prepending the "crates.io:" identifier to the `CONTRACT_NAME` in the migrate and `set_contract_version` functions.

Status: Resolved

4. Incomplete state migration in `account-nft` contract

Severity: Major

In `rover:contracts/account-nft/src/state.rs:5`, the `CONFIG` state uses `Item<NftConfig>` for version `2.0.0`. As version `1.0.0` uses `Item<Config>`, a state migration is required in order to support the new `health_contract_addr` field in the `NftConfig` struct. However, the migrate handler does not handle this in `rover:contracts/account-nft/src/contract.rs:88`.

Consequently, the `burn` and `update_config` functions will fail to load the `CONFIG` state successfully, preventing the contract from working as intended.

Recommendation

We recommend performing the relevant state migrations for the `CONFIG` storage state.

Status: Resolved

5. Circuit breakers may block liquidations, risking the protocol's solvency

Severity: Major

In `rover:contracts/credit-manager/src/execute.rs:77`, the account's health state is queried with `ActionKind::Default` before dispatching any actions. The `ActionKind::Default` query is subjected to the `max_confidence` and `max_deviation` price validations, as seen in `red-bank:contracts/oracle/base/src/pyth.rs:80-81`. If one of the validations fails, the circuit breakers will be triggered, preventing the account from dispatching actions during cases of extreme volatility.

However, the circuit breakers might unintentionally block liquidation attempts. This behavior is inconsistent with the documentation, as one of the objectives is to let the protocol always be open for liquidation. To illustrate the scenario, assume a liquidator deposits ATOM as the debt asset to their account in preparation for liquidation attempts while also borrowing other assets.

When the ATOM's price experiences high volatility, the `max_confidence` or `max_deviation` circuit breakers can trigger, causing the `ActionKind::Default` health state query to fail. If the liquidator wants to liquidate an undercollateralized position's debt with ATOM, the `Liquidate` action cannot be dispatched due to the previous health state query. Consequently, the liquidator cannot liquidate unhealthy positions using their account's debt assets, ultimately risking the protocol's solvency.

Additionally, borrowers cannot improve the health of their positions during extreme market circumstances, exposing them to liquidation risks.

Recommendation

We recommend revising the health state query logic to ensure that liquidation attempts are not accidentally blocked during times of extreme volatility. At the very least, borrowers should be able to improve the health of their positions, independent of market conditions.

Status: Resolved

The client implemented a fix that allows deposit and self-repayment to be performed without health checks, enabling users to improve the health of their positions during high market volatility. This fixes the issue as liquidators can repay any outstanding debt irrespective of

market conditions before performing a liquidation, such that no health factor checks are triggered.

6. Inaccurate health computation due to default collateral limit

Severity: Major

When computing an account's health, the user's lending positions are queried using `None` as the `limit` parameter in `rover:contracts/credit-manager/src/query.rs:69`. In `red-bank:contracts/red-bank/src/query.rs:185`, the red bank contract interprets this request as requesting a maximum of five collaterals by default (`DEFAULT_LIMIT`) when the `limit` parameter is unwrapped.

Consequently, the health computation will be unable to include the excess assets if the account lent more than five assets in the red bank contract, causing incorrect health LTV computation.

Recommendation

We recommend dynamically configuring the `limit` parameter based on the length of the whitelisted coins in the red bank contract.

Status: Resolved

7. Credit manager migration requires the reward collector `account-id`

Severity: Minor

In `rover:contracts/credit-manager/src/migrations/v2_0_0.rs:42`, the migrate handler requires the caller to provide the reward collector's address and account id and store it in the `REWARDS_COLLECTOR` storage. This requirement is invalid because there is no entry point for the rewards collector contract to mint an account id for themselves before the version `2.0.0` update. If the owner misconfigures the account id to another user's account, the protocol fee will be distributed to them incorrectly.

Recommendation

We recommend removing `rover:contracts/credit-manager/src/migrations/v2_0_0.rs:42` and instead enabling the owner call the `update_config` function to issue an account id for the rewards collector contract.

Status: Resolved

8. Contract version is not updated

Severity: Minor

The incentive contract inherits the version of the workspace in `red-bank:Cargo.toml`. During initialization, this version is set correctly. However, during migration, the contract version is not updated in `red-bank:contracts/incentives/src/contract.rs:767-769`.

This could lead to confusion during future migrations as it is not clear which version of the contract is deployed in production.

Recommendation

We recommend updating the contract version during the migration of the incentives contract.

Status: Resolved

The client implemented the migration handler to support state migration from the v1.0.0 tag for Osmosis mainnet deployment, aside from fixing the reported issue.

9. Users cannot exit vaults for assets that are no longer whitelisted

Severity: Minor

In the rover credit manager contract, assets sent are verified to ensure that they are whitelisted. This is also performed during subscription to a vault in `rover:contracts/credit-manager/src/vault/enter.rs:41`.

Should an asset be removed from the whitelist, then users would be unable to exit from both locked and unlocked vaults, causing their funds to be locked.

Recommendation

We recommend removing the `assert_vault_is_whitelisted` validation in `rover:contracts/credit-manager/src/vault/exit.rs:22` and `rover:contracts/credit-manager/src/vault/exit_unlocked.rs:26`.

Status: Resolved

10. Credit manager contract is not defined when instantiating the health contract

Severity: Minor

The credit manager contract address is not stored upon instantiation of the health contract. As soon as the health contract is deployed, the health state and values of a user's position can be queried by the credit manager.

However, should this occur, each query would fail during the creation of the new `HealthQuerier` object in `rover:contracts/health/src/querier.rs:20`. Consequently, this would cause the failure of any calculation of position health in the credit manager.

Recommendation

We recommend adding the ability to optionally define the address of the credit manager contract during the instantiation of the health contract. Additionally, if the credit manager contract is not stored during the creation of a `HealthQuerier` object, then a custom error message should be returned gracefully. This should also be applied when querying the contract configuration in `rover:contracts/health/src/contract.rs:71`.

Status: Resolved

11. Insufficient validation of the interest rate module

Severity: Minor

In `red-bank:packages/types/src/red_bank/interest_rate_model.rs`, the interest rate model of the red bank is defined, which uses two constant slope variables to define the interest rate on either side of the optimal utilization rate.

However, during the creation of a new market, it is not validated that `slope_1` is less than `slope_2`, prior to the optimal utilization rate.

Were this to be the case, then the economic model of the red bank would be undermined, which could cause significant losses.

We classify this issue as minor since it can only be caused by the owner.

Recommendation

We recommend performing additional validation of the interest rate model during the creation and update of a market to ensure that the `slope_1` is less than `slope_2`.

Status: Resolved

12. Insufficient price source validation for liquidity token pricing

Severity: Minor

In `red-bank:contracts/oracle/osmosis/src/price_source.rs:617-623`, the current liquidity pool pricing implementation queries an underlying asset's price without proper validation against `price_sources`. This is problematic because it is crucial to ensure that the price source for denom is not set to an AMM spot price.

For example, the `querying_xyk_lp_price_success` test case in `red-bank:integration-tests/tests/test_oracles.rs:163` sets the price source as the Osmosis pool's spot price. If the price sources are not validated explicitly, an oversight by admins can potentially open up a price manipulation attack.

We classify this issue as minor because it can only happen due to a misconfiguration by the contract owner. From a secure coding point-of-view, it is recommended that proper validation checks are performed in the code rather than relying on the owner to prevent misconfigurations.

Recommendation

We recommend explicitly validating that the `price_source` for liquidity token pricing does not use a spot price from an AMM.

Status: Acknowledged

The client mentioned using spot and fixed price sources only during integration testing.

13. Potential failure of `incentives` contract migration due to unbounded iteration

Severity: Informational

In `incentive-migration:contracts/incentives/src/migrations/v2_0_0.rs:198-283`, the `migrate_indices_and_unclaimed_rewards` function performs an unbounded iteration over all keys in `ASSET_INCENTIVES`, `USER_ASSET_INDICES`, and `USER_UNCLAIMED_REWARDS` storages. This is required because the latest version of the `incentives` contract implements a new storage design that differs from the old one.

If there are too many asset incentives, users, or unclaimed rewards to iterate over though, the transaction will fail due to an out-of-gas error, preventing the migration from working correctly.

We classify this issue as informational because a failed migration attempt does not inherently compromise the protocol's security. If the migration fails, the contract admin can upload a fixed contract with an updated migration function.

Recommendation

We recommend performing the state migrations in batches. A pausing mechanism can be implemented to ensure that the migration has been completed before the contract returns back to full operation.

Status: Acknowledged

14. Conflict between documentation and implementation of the borrow rate calculation

Severity: Informational

In `red-bank:packages/types/src/red_bank/interest_rate_model.rs:27`, the `get_borrow_rate` function uses `slope_1` when the current utilization rate is less than or equal to the optimal utilization rate. However, in line `14`, the documentation states that the utilization rate should use `slope_1` when the current utilization rate is less than the optimal utilization rate, not mentioning the case of equality.

We classify this issue as informational as the difference between documentation and implementation does not impact the functionality, but might mislead developers or cause problems in the future.

Recommendation

We recommend ensuring that the implementation and documentation are aligned.

Status: Resolved

15. Use of hardcoded versions during migration

Severity: Informational

When migrating the credit manager contract in `rover:contracts/credit-manager/src/contract.rs:132`, the values of the old and new versions are hardcoded.

However, the previous and new contract versions are available as constants and from the environment in lines `rover:contracts/credit-manager/src/migrations/v2_0_0.rs:11` and `rover:contracts/credit-manager/Cargo.toml` respectively.

Recommendation

We recommend using the existing hardcoded variables when referencing the previous contract versions throughout the codebase and likewise using the contract version defined in the environment file when referencing the current contract version.

Status: Resolved

16. Inefficient execution of messages

Severity: Informational

For each dispatch action for a particular account, the contract always performs a number of callback messages to ensure that certain conditions are met.

However, during the execution of the callback message `AssertAccountReqs` in `rover:contracts/credit-manager/src/execute.rs:255`, conditions are only checked if the account is of type high-leverage strategy. This causes the creation of a redundant message, increasing the code's complexity and computational resources required.

Recommendation

We recommend performing the check of the account type prior to the creation of the callback message `AssertAccountReqs` and only creating the message if the account is a high-leverage strategy type.

Status: Resolved

17. Use of custom logic instead of standard libraries

Severity: Informational

Throughout the codebase, there are a number of custom conditions that could be simplified by using the `cosmwasm_std` library functions `ensure!`, `ensure_eq!`, and `ensure_ne!`.

For example, in `rover:contracts/credit-manager/src/utils.rs:36`, the condition `if user != owner { … }` is used to return an error if the condition is met.

The following provides a list of non-exhaustive examples:

- `rover:contracts/account-nft/src/contract.rs:94,100`
- `rover:contracts/account-nft/src/execute.rs:80`
- `rover:contracts/credit-manager/src/execute.rs:36`
- `rover:contracts/credit-manager/src/utils.rs:36`
- `rover:contracts/swapper/src/contract.rs:81`

Recommendation

We recommend using standard libraries where possible to perform functions as they assist with code readability and maintenance.

Status: Acknowledged

18. `update_nft_config` emits the same attribute value as `update_config`

Severity: Informational

In `rover:contracts/credit-manager/src/update_config.rs:141`, the `update_nft_config` function emits the `action` key attribute as "`update_config`". This is misleading because the `update_config` function in line `27` emits the same value, potentially confusing event indexers and off-chain listeners.

Recommendation

We recommend modifying the `action` key attribute to "`update_nft_config`" in `rover:contracts/credit-manager/src/update_config.rs:141`.

Status: Resolved

19. `get_route` function can be used to reduce code complexity

Severity: Informational

In `rover:contracts/swapper/base/src/contract.rs:164-170`, the `swap_exact_in` function attempts to load a swap route and errors if there is no route for the `coin_in.denom` and `denom_out` combination. As the `get_route` function in lines `248-255` performs the same functionality, the code complexity can be reduced by using that function.

Recommendation

We recommend using the `get_route` function to reduce code complexity and increase code maintainability.

Status: Resolved

20. `WHITELIST_COUNT` is not exposed through the configuration query

Severity: Informational

In `red-bank:contracts/incentives/src/contract.rs:614-624`, the `query_config` function does not return the `WHITELIST_COUNT` as part of the `ConfigResponse` struct. Consequently, users and other smart contracts would need to perform a raw query to retrieve the `WHITELIST_COUNT` storage value, decreasing user experience.

Recommendation

We recommend including the `WHITELIST_COUNT` response in the `query_config` function.

Status: Resolved

21. Missing maximum validation in `query_all_asset_params` and `query_all_vault_configs` queries

Severity: Informational

In `red-bank:contracts/params/src/query.rs:23` and `50`, the `query_all_asset_params` and `query_all_vault_configs` functions do not perform maximum validations for the caller-specified `limit` parameter. If the caller specified the limit parameter to a very large value, the query might fail due to an out-of-gas error.

Recommendation

We recommend performing maximum limit validations similar to `red-bank:contracts/address-provider/src/contract.rs:153`.

Status: Resolved