Audit Report

Mars Protocol Core Smart Contracts

v1.0

February 17, 2022

Table of Contents

License

Disclaimer

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 to perform a security audit of Mars protocol 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 behaviour.
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 GitHub repository:

https://github.com/mars-protocol/mars-core

Commit hash: `94183e9555122a5cdfc8ca967e190215f61f8726`

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

The submitted contracts implement Mars, a DeFi money market protocol built on the Terra blockchain.

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 improve 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 a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	Deactivated market assets would cause forced liquidation on borrowers	Critical	Acknowledged
2	Staking xMars rewards can be sandwiched by an attacker, skimming its value before accruing to stakers	Critical	Acknowledged
3	Malicious smart contracts can avoid liquidation attempts	Critical	Resolved
4	Disabled collateral assets can be liquidated	Critical	Resolved
5	Incorrect slashing calculation will lead to loss of user funds	Critical	Resolved
6	Users will be unable to claim rewards if too much assets are added	Minor	Acknowledged
7	New Mars incentives set to a wrong address could block incentive rewards in any assets irrevocably	Minor	Acknowledged
8	Initializing vesting contract wrongly would cause inconsistent state	Minor	Resolved
9	Duplicate accounts creation would cause inconsistent total supply	Minor	Acknowledged
10	Sent tokens other than denoms are lost	Minor	Resolved
11	Expiration can be set to past timestamp and block height	Minor	Acknowledged
12	Misconfigured required proposal threshold would lead to unfair voting process	Minor	Resolved
13	Incorrect error messages returned to user	Informational	Resolved
14	Overflows checks not set for profile release in packages/mars-core/Cargo.toml	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium-High	-
Code readability and clarity	Medium-High	-
Level of documentation	Medium	-
Test coverage	Medium	-

Detailed Findings

1. Deactivated market assets would cause forced liquidation on borrowers

Severity: Critical

In `mars-red-bank/src/account.rs:61`, the function `get_user_position` is used internally to calculate the borrower's debt, average liquidation threshold and most importantly the health factor which determines whether the borrower can be liquidated. The function iterates over all assets in that the borrower has a position, without accounting in the market's active state. If a market is deactivated, borrowers will be unable to perform any actions on it including repaying their loans.

This would cause problems if the market asset the borrower is borrowing had been deactivated by the contract owner. The debt of the inactive market would still be included when calculating the overall health factor for the borrower. Although the deactivated market asset prevents liquidation attempts, a liquidator can still bypass it by liquidating other active collateral assets. Hence, the borrower would be subjected to forced liquidation due to the inability to repay their loans.

Recommendation

We recommend excluding inactive market assets during `get_user_position`.

Status: Acknowledged

The Mars team states that the disabling of assets would be only on extreme events, such as an exploit, in which case protecting the protocol solvency as a whole is preferred rather than making individual users whole.

2. Staking xMars rewards can be sandwiched by an attacker, skimming its value before accruing to stakers

Severity: Critical

Protocol rewards can be swapped and transferred by anyone to the Staking contract. A combination of public messages executed in one transaction can be leveraged by an attacker to skim the rewards of xMars stakers as shown below:

1. Withdraw Mars rewards (or `ma_mars`) from the Red Bank to the Protocol Collector using `execute_withdraw_from_red_bank` in `mars-protocol-rewards-collector/src/contract.rs:187`.
2. Sell assets if you have Mars/maMars inventory.

3. Call `execute_swap_asset_to_uusd` in L321 in the Protocol Rewards Collector, pushing the price of Mars down and converting it to UST.
4. Buy assets again (MARS) in the market.
5. Call `execute_distribute_protocol_rewards` in L216 sending a portion of the UST to staking contract
6. Call `execute_swap_uusd_to_mars` in the Staking contract, `mars-staking/src/contract.rs:448`, pushing the price of Mars up.
7. Sell MARS bought in step 4.

This can be also exploited with rewards denominated in assets other than Mars, but the severity of this performed on Mars rewards is greater due to steps 6 and 7.

Recommendation

We recommend either rate-limiting/restricting the access of these public functions, or allowing the Protocol Rewards Collector to send rewards directly in the denomination they are withdrawn from the Red Bank, and swapping them back to Mars on the Staking contract without the intermediate swap to UST.

Status: Acknowledged

The Mars team is working on ideas to solve this on future versions of the protocol that the community could adopt. In the meantime, the community would have to evaluate the risk of adding Mars as a borrowable asset to the money market. At first most fees will be collected in uusd which would only be swapped to Mars in the case of staking which would make the window for this attack small.

3. Malicious smart contracts can avoid liquidation attempts

Severity: Critical

In the Red Bank contract, uncollateralized loan limits for a specific asset can be given to a user (or smart contract) via `UpdateUncollateralizedLoanLimit` message. This is problematic because it allows users with an outstanding collateralized debt balance to have their uncollateralized loan limit to increase.

According to the litepaper, uncollateralized debt limits will be issued to whitelisted smart contracts after the council decides on the risk framework. A malicious smart contract can deposit and take up a huge loan before the proposal is approved. Once the proposal is approved, the corresponding `UNCOLLATERALIZED_LOAN_LIMITS` for the specific asset is updated. In case the liquidation threshold for the asset is reached, the malicious smart contract will be free from liquidation attempts due to `CannotLiquidateWhenPositiveUncollateralizedLoanLimit` error message as seen in `contracts/mars-red-bank/src/contract.rs:1216-1223`. This would cause loss of funds for the lenders.

Recommendation

We recommend checking for outstanding collateralized debts before setting a new uncollateralized limit for smart contracts in `UpdateUncollateralizedLoanLimit`.

Status: Resolved

4. Disabled collateral assets can be liquidated

Severity: Critical

In the Red Bank contract, users can enable or disable their collateral assets via `UpdateAssetCollateralStatus` message. The health factor is verified to be higher than 100% when disabling the collateral asset to prevent sudden liquidation attempts. This implies that users should be able to control which collateral assets are "allowed" for liquidation and vice versa.

However, `execute_liquidate` does not account for whether the collateral asset specified by the liquidator is open for liquidation or not. As seen in `contracts/mars-red-bank/src/contract.rs:1371-1380`, the collateral asset's bit is unset without accounting for whether the asset is being used as collateral. This means that the borrowers' disabled collateral asset can still be liquidated which would cause unexpected loss of funds.

Recommendation

We recommend verifying whether the `collateral_asset` specified by the liquidator is being used as collateral before continuing the liquidation process, eg. `contracts/mars-red-bank/src/accounts.rs:142`.

Status: Resolved

5. Incorrect slashing calculation will lead to loss of user funds

Severity: Critical

In `contracts/mars-staking/src/contract.rs:377`, the `slash_percentage` is determined via the amount of Mars to retrieve divided by the total staked Mars as a `Decimal`. The `slash_percentage` is then multiplied in `L386` to mutate the value with this formula: `global_state.total_mars_for_claimers = global_state.total_mars_for_claimers * slash_percentage;`.

This will be problematic when stakers decide to claim their staked Mars via `execute_claim`. In `L313`, `apply_slash_events_to_claim` is called to determine the amount stakers can claim after deducting the value from existing slash events. The formula is however different from `L386` as in `L563` it deducts the claim amount such as

`claim.amount = claim.amount * (Decimal::one() - slash_event.slash_percentage);`.

To illustrate, let's imagine the following scenario with user as Alice (for simplicity let's also assume the exchange rate for Mars and xMars is 1:1):

1. There is a total of `10000` Mars token staked in the staking contract
2. Alice sends `1000` xMars token to staking contract and executes `execute_unstake` to claim their Mars token, this would cause the value of `global_state.total_mars_for_claimers` to `1000` as seen in `L275-277`. This means Alice would be eligible to claim `1000` Mars after the cooldown period.
3. A slashing event happens and the contract owner executes `execute_transfer_mars` to retrieve `2000` Mars from staking contract, this would cause the value of `slash_percentage` to be `0.2` (`2000/10000`). In `L386`, the value of `global_state.total_mars_for_claimers` would become `200` (`1000*0.2`).
4. Cooldown period ends and Alice tries to claim her Mars token via `execute_claim`. In `L313`, `apply_slash_events_to_claim` is called and `claim.amount` becomes `800` (`1000*(1-0.2)`).
5. The operation would fail in `L316-318` since `200` minus `800` would cause an underflow error. Alice would need to wait for another user to execute `execute_unstake` so the value of `global_states.total_mars_for_claimers` would be enough for her to claim. Due to the incorrect slashing calculation, some of the user's funds are lost.

Recommendation

We recommend modifying `L386` into `global_state.total_mars_for_claimers * (Decimal::one() - slash_percentage);`. Mimicking the above scenario, this would cause `global_state.total_mars_for_claimers` to become `800` instead of `200`. When Alice decides to claim her share of Mars token, the underflow issue would not happen again. The slashing penalty is also applied since Alice's original `1000` Mars is decreased to `800` Mars.

Status: Resolved

6. Users will be unable to claim rewards if too much assets are added

Severity: Minor

In the Incentives contract, users can call `ClaimReward` to retrieve their accrued Mars rewards. The message then calls the internal `compute_user_unclaimed_rewards` function which triggers a loop from all the asset incentives. Asset incentives are unbounded as they cannot be removed once added via `SetAssetIncentive` message. On a long enough timeframe, if many assets get added into the protocol, this could introduce out of gas

errors and block the claim of rewards for a given user. This issue is also present in the `UserUnclaimedRewards` query message.

The same issue is also present in `contracts/mars-staking/src/contract.rs:530`, where the user may run out of gas if the list of slash events to apply grows to a certain point over time.

Recommendation

We recommend adding pagination to both calls to avoid any potential out of gas errors in the future. Alternatively, a thorough benchmarking of the unbounded messages by the team would be recommended.

Status: Acknowledged

The Mars team has profiled this function, and since slashing is going to be an extreme scenario, consider this to be a non-issue.

7. New Mars incentives set to a wrong address could block incentive rewards in any assets irrevocably

Severity: Minor

In the Mars Incentives contract, `mars-incentives/src/contracts.rs:82`, the owner of the contract can set new asset incentives with `execute_set_asset_incentive`. However, there is no check that the input `ma_token_address` is a `ma_token` or even cw20. Also, `ASSET_INCENTIVES` cannot be removed from `Storage`.

If there is a human error and a non-cw20 address is submitted as asset incentives, all claimable rewards will be blocked forever as `execute_claim_rewards`, which internally calls `compute_user_unclaimed_rewards` would always panic in the `cw20Balance` call in `mars-incentives/src/contract.rs:410`.

Recommendation

We recommend adding validations in `execute_set_asset_incentive` to double-check that any new input address is a cw20 `ma_token` address.

Status: Acknowledged

The Mars team states that asset incentives are approved by governance so this would require everyone reviewing the proposal to miss the address in question. Also, if the error would happen, no MARS will be lost as rewards will never be accounted for (because no BalanceChange call would happen ever) nor claimed.

8. Initializing vesting contract wrongly would cause inconsistent state

Severity: Minor

In the Vesting contract, configurations such as `unlock_start_time`, `unlock_cliff` and `unlock_duration` are configured during contract initialization. There are no validations in place to verify submitted values are correct, eg. `unlock_start_time` must be a timestamp in the future instead of the past.

Recommendation

We recommend adding the following validations during contract initialization:

- `unlock_start_time` must be a timestamp greater than present time
- `unlock_duration` should be greater than `unlock_cliff`
- Submitted values must not be 0

Status: Resolved

9. Duplicate accounts creation would cause inconsistent total supply

Severity: Minor

In the xMars token contract, accounts are created via `create_accounts` during contract initialization. As shown in `contracts/mars-xmars-token/src/contract.rs:53-62`, the function loops through a set of `accounts` without verifying possible duplications. If the `accounts` contain any repeated address, the previous balance would be overwritten but the total_supply would increase, creating a discrepancy between the two.

Recommendation

We recommend adding a verification for checking duplicate accounts during `create_accounts`.

Status: Acknowledged

The Mars team states that there will be no accounts for xMars on initialization, as xMars need staked Mars in order to be minted. The client prefers to keep the code as close to the cw20 standard as possible. They have also raised the issue on the cw-plus repository.

10. Sent tokens other than denoms are lost

Severity: Minor

In the Red Bank contract, users can deposit native assets via `DepositNative` message. The coins are then parsed from an internal function which is called `get_denom_amount_from_coins`. The function does not limit user sent funds to exactly one type of funds. If the user sent two types of funds (eg. LUNA and UST), the other assets which are not denom would be lost in the contract.

Recommendation

We recommend adding validation to verify that the user only sent one type of funds, eg. checking `info.funds.len` to be only 1.

Status: Resolved

11. Expiration can be set to past timestamp and block height

Severity: Minor

In the Ma-token contract, `IncreaseAllowance` and `DecreaseAllowance` allow specifying an `Expiration` via block height or block timestamp. The specification does not verify whether the supplied `Expiration` value is higher than the current timestamp or block height. Users may set the expiration value to a past timestamp or block height, which causes the approval to fail.

Recommendation

We recommend adding a verification to validate the supplied `Expiration` value to be greater than the current block height or block timestamp.

Status: Acknowledged

The Mars team prefers to keep the code the same as the cw20 standard. They have raised the issue on the cw-plus repository.

12. Misconfigured required proposal threshold would lead to unfair voting process

Severity: Minor

In the Council contract, `proposal_required_threshold` is used to determine the required percentage of votes in order to consider the proposal to be successful. It will then be used in `contracts/mars-council/src/contract.rs:373-376` to determine whether the current voting percentage is greater than the configured proposal required threshold.

This would be problematic if `proposal_required_threshold` is configured to a value below 50%. The proposal would be executed despite the majority disagrees with the

proposal. For example, a configured proposal threshold at 30% would indicate that the proposal would be executed once 31% of the votes are `for_votes`. This is unfair since the `against_votes` will be 69%, which represents the majority of voter's decision.

Recommendation

We recommend adding validation to verify that the value of `config.proposal_required_threshold` must be above `0.5` which corresponds to 50%.

Status: Resolved

13. Incorrect error messages returned to user

Severity: Informational

There are several duplicate and incorrect error messages found during the audit process of the contract:

- `CannotRepayMoreThanDebt` and `CannotLiquidateWhenPositiveUncollateralizedLoanLimit` returns the same error message
- `InvalidHealthFactorAfterWithdraw` and `AssetAlreadyInitialized` returns the same error message

Recommendation

We recommend setting an appropriate error message for the mentioned error message in `contracts/mars-red-bank/src/error.rs`.

Status: Resolved

14. Overflows checks not set for profile release in packages/mars-core/Cargo.toml

Severity: Informational

While set in the project root Cargo.toml, packages/mars-core/Cargo.toml does not enable overflow-checks for the release profile.

Recommendation

While this check is implicitly applied to all packages from the workspace Cargo.toml, we recommend also explicitly enabling overflow checks in every individual package. That helps when/if the project is refactored to prevent unintended consequences.

Status: Resolved