Audit Report

Mars Periphery 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 the Mars Periphery 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 GitHub repository:

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

Commit hash: `130c3c600f16ce7e751d769a4699f96fa5eda61e`

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 lending money market platform on the Terra blockchain. The scope of this report pertains to the periphery elements of the project including its airdrop, lockdrop and auction contracts. The airdrop contract facilitates a Mars airdrop for active Terra users, lockdrop allows for bootstrapping the maUST reserves, and the auction contract facilitates MARS-UST Astroport pool initialization during the protocol launch.

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	Users are unable to withdraw funds once admin deposited all funds in the Red Bank	Major	Resolved
2	Malicious users can cheat lockup rewards without locking their funds if claim is enabled before withdrawal timeline	Minor	Resolved
3	Users that claimed airdrops will miss out on additional airdrops	Minor	Acknowledged
4	Missing validation on Config parameters can lead to human errors and unexpected behavior	Minor	Resolved
5	Missing MARS rewards validation may cause auction participants to lose their deserved rewards	Minor	Acknowledged
6	Querying the state of the contract may return false information	Minor	Resolved
7	Typo in variable names might cause errors in the future	Informational	Resolved
8	Overflow checks not set for most packages	Informational	Resolved

Code Quality Criteria

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

Detailed Findings

1. Users are unable to withdraw funds once admin deposited all funds in the Red Bank

Severity: Major

In the Lockdrop contract, users are able to withdraw funds as long as it's under the configured withdrawal timeline. The withdrawal function will decrease the contract's total locked UST based on the user's withdrawal amount (`contracts/lockdrop/src/contract.rs:395`). Once the admin deposits all funds to the Red Bank via `DepositUstInRedBank`, the contract's total locked UST will be set to 0 (`L791`). Since it only accounts for the deposit timeline instead of the withdrawal timeline (`L465-472`), users will be unable to withdraw their locked funds even if the withdrawal window is still open.

We set the severity to Major since the users' funds are still recoverable after a certain timeline.

Recommendation

We recommend checking the lockdrop withdrawal window to be closed before allowing admin to execute `DepositUstInRedBank`.

Status: Resolved

2. Malicious users can cheat lockup rewards without locking their funds if claim is enabled before withdrawal timeline

Severity: Minor

In the Lockdrop contract, users can execute `ClaimRewardsAndUnlock` to claim rewards in the condition that claims are enabled (`contracts/lockdrop/src/contract.rs:686-688`). Claims can only be enabled from the Auction contract via `AddLiquidityToAstroportPool` which accounts in deposit and withdrawal windows only for the Auction contract itself (`contracts/auction/src/contract.rs:440-445`).

Since both Auction and Lockdrop contracts are independently initialized, there's a possibility that admin might enable claims during the deposit or withdrawal phase of Lockdrop contract. If this happens, a malicious user can potentially cheat the lockup rewards via an attack scenario below:

1. Execute `DepositUst` to deposit a large amount of funds into any lockup position
2. Claim rewards via executing `ClaimRewardsAndUnlock` with `lockup_to_unlock_duration_option` as `None`

3. In `contracts/lockdrop/src/contract.rs:703-715`, code will enter `calculate_mars_incentives_for_lockup` internal function. Notice that `lockup_info.ust_locked` variable will be used to calculate user's position rewards via `calculate_weight`.
4. Due to the code logic that *only* updates user's MARS rewards when it's 0, `user_info.total_mars_incentives` will not be updated in the future (`contracts/lockdrop/src/contract.rs:704`). The malicious user will have a definite reward.
5. Execute `WithdrawUst` to remove all funds locked under that lockup position (funds that can be removed highly depends on window timeline). In `L382`, the corresponding locked UST amount will decrease, however, the user's MARS rewards will still remain the same.

We set the severity to minor since this will only be exploitable if the timeline is configured incorrectly/far between Lockdrop and Auction contract which will influence the execution of `EnableClaims`.

Recommendation

We recommend adding validation to make sure deposit and withdrawal timeline are closed when executing `ClaimRewardsAndUnlock` (eg. using `is_withdraw_open`). This would prevent users from withdrawing their locked UST once their MARS rewards are set.

Status: Resolved

3. Users that claimed airdrops will miss out on additional airdrops

Severity: Minor

The current implementation of airdrop claims allows only one claim per user due to the condition in `contracts/airdrop/src/contract.rs:326`. At the same time, the contract owner has the ability to update Merkle roots in `L198-200`. Taken together, if a user was assigned additional airdrops, they would not be able to claim those additional airdrops. Even without an update, the current design does not allow a user to claim multiple airdrop leaves from the same Merkle tree.

Recommendation

We recommend changing the check whether a user already claimed an airdrop to compare the amount they already claimed with the amount they try to claim. That will allow users that claimed in the past to claim higher amounts if the Merkle tree is updated.

Status: Acknowledged

The Mars team states that doing another airdrop through the same contract is not in scope. If the community decides to do one, they should consider instantiating a new contract.

4. Missing validation on Config parameters can lead to human errors and unexpected behavior

Severity: Minor

The `Auction`, `Lockdrop`, and `LP Staking` contracts are missing validation checks in some of its Config numerical values.

For example, in `contracts/auction/src/contract.rs:61`, there is no check that `msg.init_timestamp` corresponds to a future value (i.e. `msg.init_timestamp>= env.block.time`). This validation is present in the Lockdrop contract in `lockdrop/src/contract.rs:45`.

An example in the LP Staking contract is that in `contracts/lp_staking/src/contract.rs:37-39`, there is no validation that `init_timestamp + cycle_duration <= till_timestamp`.

Examples of missing validation checks in the Lockdrop contract are:

- No validation that there are repeated values in `msg.lockup_durations` in line 67
- No check that `msg.seconds_per_duration_unit` is non-zero.

Examples of missing validation checks in the Auction contract are:

- No check that `config.mars_vesting_duration` is non-zero, which would cause division by zero error.

Recommendation

We recommend adding validation checks on the relevant instantiate functions and `Config` parameters to minimize human errors by the admin or deployer.

Status: Resolved

5. Missing MARS rewards validation may cause auction participants to lose their deserved rewards

Severity: Minor

In the Auction contract, anyone can deposit MARS to increase the overall reward for users via `IncreaseMarsIncentives`. If no one deposited any MARS token rewards before the admin executed `AddLiquidityToAstroportPool`, it is not possible to continue depositing MARS tokens due to LP token shares already minted (`contracts/auction/src/contract.rs:199-L203`). This would cause auction participants to receive zero Mars token rewards in return for their MARS token delegation.

Recommendation

We recommend adding validation to verify MARS tokens are added during `AddLiquidityToAstroportPool`. (eg. verifying `config.mars_rewards != 0`)

Status: Acknowledged

The Mars team states that they will make sure Mars incentives are set before adding liquidity to the pool during the deployment process.

6. Querying the state of the contract may return false information

Severity: Minor

In the LP staking contract, in lines `lp_staking/src/contracts.rs:315-352`, there is a max function that takes the timestamp provided by the user and the current timestamp. Therefore, if the user-provided timestamp is in the past, the returned information will be false, as it will correspond to the current timestamp. If the user, which can be a 3rd party protocol, relies on this query to do further calculations, this might cause unintended consequences.

Recommendation

We recommend explicitly displaying an `Error` when the `query_state` and `query_staker_info` are called with a timestamp from the past.

Status: Resolved

7. Typo in variable names might cause errors in the future

Severity: Informational

In the Lockdrop contract, `contracts/lockdrop/src/contract.rs:832`, there is a typo in the variable `xmars_accured`, which may cause development errors in the future.

Recommendation

Even if this typo error does not affect the business logic of the contract, we recommend that it gets fixed to `xmas_accrued` to minimize confusion and enhance the future development experience.

Status: Resolved

8. Overflow checks not set for most packages

Severity: Informational

The following `Cargo.toml` files do not enable overflow-checks for the release profile:

- `packages/mars-periphery/Cargo.toml`
- `contracts/airdrop/Cargo.toml`
- `contracts/auction/Cargo.toml`
- `contracts/lockdrop/Cargo.toml`
- `contracts/lockdrop/Cargo.toml`

Recommendation

Even though this check is implicitly applied to all packages from the workspace's `Cargo.toml`, we recommend also explicitly enabling overflow checks in every individual package. That helps prevent unintended consequences when the codebase is refactored in the future.

Status: Resolved