Audit Report

Mars Periphery

v0.4

October 21, 2022

Table of Contents

License

THIS WORK IS LICENSED UNDER A CREATIVE COMMONS ATTRIBUTION-NODERIVATIVES 4.0 INTERNATIONAL 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.

COPYRIGHT OF THIS REPORT REMAINS WITH THE AUTHOR.

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 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/periphery

Commit hash: `6b08edfd521c9b455b49eeb1b2e3329373636060`

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 code features Mars Periphery smart contracts to be deployed on Mars Hub.

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.

Summary of Findings

No	Description	Severity	Status
1	Liquidated and excess funds are stuck in the liquidation filterer contract	Critical	Resolved
2	Liquidation filterer contract cannot process multiple liquidations efficiently	Major	Resolved
3	Smart contracts holding tokens on Terra classic cannot claim their airdrop	Major	Acknowledged
4	`liquidation-filterer` contract is not compliant with CW2 Migration specification	Minor	Resolved
5	`InstantiateMsg` is not validated in the `airdrop` contract	Minor	Resolved
6	`Schedule` structs are not validated in the `vesting` contract	Minor	Resolved
7	liquidation-filterer contract could be optimized	Informational	Resolved
8	Overflow checks not enabled for release profile	Informational	Resolved
9	Contracts should implement a two step ownership transfer	Informational	Acknowledged
10	Custom access controls implementation	Informational	Acknowledged
11	Incorrect comment for `LiquidateMany` message	Informational	Resolved
12	Voting power is suboptimal	Informational	Acknowledged
13	Voting power is valid on the same block schedule created	Informational	Acknowledged

Code Quality Criteria

Criteria	Status	Comment
Code complexity	**Medium**	-
Code readability and clarity	**Medium-High**	-
Level of documentation	**Medium-High**	-
Test coverage	**High**	`cargo tarpaulin` reports 94.76% code coverage.

Detailed Findings

1. Liquidated and excess funds are stuck in the liquidation filterer contract

Severity: Critical

In `contracts/liquidation-filterer/src/contract.rs:84`, the `execute_liquidation` function does not properly handle collateral transfer after a liquidation attempt. We outline three possible scenarios that causes funds to be stuck in the contract:

Firstly, the liquidation logic forces the caller to provide funds in line `99`, but there is a possibility that the user to liquidate is not liquidatable. In this case, the provided funds are not refunded.

Secondly, the red bank refunds excess funds to the liquidation filterer contract (see `contracts/red-bank/src/execute.rs:1083-1088`). There is no way to withdraw these excess funds though.

Lastly, the red bank contract will transfer Mars tokens to the contract after a successful liquidation attempt. There is no logic in the contract that allows a withdrawal in return for the liquidated collateral. As a result, the liquidated collaterals are inaccessible.

Recommendation

We recommend implementing refund/withdrawal functionality.

Status: Resolved

2. Liquidation filterer contract cannot process multiple liquidations efficiently

Severity: Major

In `contracts/liquidation-filterer/src/contract.rs:114`, when `Liquidation` messages are constructed, if at least two of them have the same `debt_denom` or `user_address`, the transaction will return an error.

The former happens because the sub-messages reuse the total funds with the same denom. Since the first message already includes all funds, the contract will not have enough funds to process the subsequent liquidation messages with the same denom.

The latter fails because the contract would send duplicate liquidation messages to the red bank contract, and since liquidating a user twice is not possible, the transaction would fail too.

Additionally, the liquidation filterer contract does not tolerate failures. The whole liquidation execution would fail if any liquidation attempts were frontrunned by another liquidator. As a result, undercollateralized positions might not be liquidated efficiently during high volatility and activity in the market.

Recommendation

We recommend reworking the liquidation filterer contract to include the following suggestions:

- Correctly calculate the required amount of collateral for all liquidation attempts.
- Dedupe the borrower's address to prevent duplicate liquidations.
- Continue the overall liquidation process even if one liquidation fails by handling the failures with sub-messages (elaborated more in the next section).

Status: Resolved

3. Smart contracts holding tokens on Terra classic cannot claim their airdrop

Severity: Major

In the scripts, the Merkle root is generated from historical data of Terra classic. If a smart contract holds tokens on Terra, the Merkle root is generated with the smart contract's address, but there is no way that a smart contract can sign a message. In `contracts/airdrop/src/contract.rs`, the way users prove their ownership of Terra Address won't allow smart contracts to claim airdrops.

Recommendation

We recommend explicitly specifying the mechanism for smart contracts to claim airdrops.

Status: Acknowledged

The Mars team mentioned that there are four multisig contract addresses that held Mars on Terra Classic, and they will have nominated addresses from them to receive the airdrop.

4. `liquidation-filterer` contract is not compliant with CW2 Migration specification

Severity: Minor

The `liquidation-filterer` contract does not adhere to the CW2 Migration specification standard.

This may lead to unexpected problems during contract migration and code version handling.

Recommendation

We recommend following the CW2 standard in all the contracts. For reference, see https://docs.cosmwasm.com/docs/1.0/smart-contracts/migration.

Status: Resolved

5. `InstantiateMsg` is not validated in the `airdrop` contract

Severity: Minor

In `contracts/airdrop/src/contract.rs:28`, when handling the `InstantiateMsg`, no validation of the `merkle_root` length and sent funds is performed.

Recommendation

We recommend implementing validation logic that ensures that the `merkle_root` has the correct length and that funds with the `umars` denom are sent to the contract.

Status: Acknowledged

The Mars team states that the contract will be instantiated in the chain's genesis state which will be carefully reviewed.

6. `Schedule` structs are not validated in the `vesting` contract

Severity: Minor

In

- `contracts/vesting/src/contract.rs:37`,
- `contracts/vesting/src/contract.rs:102`, and
- `contracts/vesting/src/contract.rs:65`,

the `Schedule` structs are handled without being validated.

This could lead to situations where the `start_time` timestamp value is less than the current timestamp or the `cliff` value is greater than or equal to `duration`.

Recommendation

We recommend implementing a validation logic for `Schedule` structs.

Status: Acknowledged

7. `liquidation-filterer` contract could be optimized

Severity: Informational

The `liquidation-filterer` contract is designed to be able to trigger a set of liquidations in the red bank. In order to accomplish this, the `LiquidateMany` message takes a list of `Liquidate` elements, iterates through all of them, performs a query in order to check if they are liquidatable, and then forwards the `Liquidate` messages to the red bank.

Since the list of `Liquidate` elements could be of a relevant cardinality and each iteration performed both a query and a transaction, there is a possibility that the execution goes out of gas and reverts all the intended liquidations. Also, transactions can change the health status of a position, which invalidates previous query results.

Recommendation

We recommend implementing a different logic to optimize the transaction execution. For example, instead of adding the overhead of checking the health of a position before sending liquidation messages, `SubMsg` could be used with the `ReplyOn` field to handle errors.

Status: Resolved

8. Overflow checks not enabled for release profile

Severity: Informational

The following packages and contracts do not enable `overflow-checks` for the release profile:

- `contracts/liquidation-filterer/Cargo.toml`
- `contracts/airdrop/Cargo.toml`
- `contracts/vesting/Cargo.toml`

While enabled implicitly through the workspace manifest, future refactoring might break this assumption.

Recommendation

We recommend enabling overflow checks in all packages, including those that do not currently perform calculations, to prevent unintended consequences if changes are added in future releases or during refactoring. Note that enabling overflow checks in packages other than the workspace manifest will lead to compiler warnings.

Status: Resolved

9. Contracts should implement a two step ownership transfer

Severity: Informational

The contracts within the scope of this audit allow the current owner to execute a one-step ownership transfer. While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to the incorrect address. A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Acknowledged

The Mars team states that they may address this in a future version.

10. Custom access controls implementation

Severity: Informational

Contracts implement custom access controls. Although no instances of broken controls or bypasses have been found, using a battle-tested implementation reduces potential risks and the complexity of the codebase.

Also, the access control logic is duplicated across the handlers of each function, which negatively impacts the code's readability and maintainability.

Recommendation

We recommend using a well-known access control implementation such as `cw_controllers::Admin` (https://docs.rs/cw-controllers/0.14.0/cw_controllers/struct.Admin.html).

Status: Acknowledged

The Mars team states that they plan to implement this together with the two-step ownership transfer described above.

11. Incorrect comment for `LiquidateMany` message

Severity: Informational

In `contracts/liquidation-filterer/src/msg.rs:17`, the comment for the `LiquidateMany` message is "*Set emission per second for an asset to holders of its maToken*". This is incorrect because the message's functionality is to liquidate multiple undercollateralized positions from the red bank.

Recommendation

We recommend correcting the comment.

Status: Resolved

12. Voting power is suboptimal

Severity: Informational

In `contracts/vesting/src/contract.rs:199`, the voting power is calculated suboptimally. The Mars hub calculates the voting power from staked tokens (hub validators), locked tokens in the vesting contract, and unlocked tokens from the vesting contract that are not yet withdrawn `Ok(Some(position)) => position.total - position.withdrawn`. Users can vote on a proposal and immediately withdraw the withdrawable tokens.

Recommendation:

We recommend following the curve-style voting power calculations, where voting power decreases as time increases (because the time to unlock vested tokens decreases).

Status: Acknowledged

13. Voting power is valid on the same block schedule created

Severity: Informational

In `contracts/vesting/src/contract.rs:197`, the `query_voting_power` functionality determines a user's voting power by checking the `POSITIONS` storage state. Since positions are directly updated in the same block when `create_position` is called, this might allow an exploit similar to the Beanstalk exploit.

With that said, it is currently not exploitable since only the contract owner can create positions. However, suppose the contract owner is another smart contract allowing permissionless position creation. In that case, an attacker can flash loan a big amount of Mars tokens to create a large position for themselves which comes with high voting power.

We consider this a minor issue because it may lead to a vulnerability if the architecture changes in the future.

Recommendation

We recommend updating the user's voting power a block after the position is created for the user.

Status: Acknowledged