Audit Report

Mars v2 on Neutron

v1.0

July 15, 2024

Table of Contents

Table of Contents **2**
License **3**
Disclaimer **4**
Introduction **5**
Purpose of This Report 5
Codebase Submitted for the Audit 5
Methodology 7
Functionality Overview 7
How to Read This Report **8**
Code Quality Criteria **9**
Summary of Findings **10**
Detailed Findings **11**
1. Total liquidity tokens are incorrectly increased, causing lower rewards 11
2. Attackers can bind vault account ID to forcefully cause a loss for users 11
3. Liquidatee’s staking rewards are lost 12
4. Users cannot stake new liquidity tokens on Astroport 12
5. Potentially outdated configurations stored in the vault 13
6. Incorrect liquidity tokens unstaked for ActionAmount::Exact 13
7. Using AstroportSpot can introduce price manipulation risk 14
8. Vault unlock may revert because of cooldown end timing of a position 14
9. Duplicate code 15
10. Inconsistent use of u32::from and as u32 for type conversion 15
11. Redundant check for !rewards.is_empty() in claim_lp_rewards 15
12. Incorrect token referenced in comment 16
13. Unnecessary address lookup 16
14. Missing validations 16
15. Redundant code setting accumulated_pnl and accumulated_fee to zero 17

License

This audit has been performed by

Oak Security GmbH

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

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by Mars Protocol Foundation to perform a security audit of the Mars on Neutron v2 codebase.

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 target:

Repository	https://github.com/mars-protocol/contracts
Commit	`3d59ad32e34ac64a821ddee5f5e3b017c04a7312`
Scope	mars-oracle-wasm: ● `contracts/oracle/wasm/src/price_source.rs` ● `contracts/oracle/wasm/src/lp_pricing.rs` mars-credit-manager: ● `contracts/credit-manager/src/stake_astro_lp.rs` ● `contracts/credit-manager/src/unstake_astro_lp.rs` ● `contracts/credit-manager/src/claim_astro_lp_rewards.rs`

	● `contracts/credit-manager/src/liquidate_astro_lp.rs` ● `contracts/credit-manager/src/execute.rs` ● `contracts/credit-manager/src/deposit.rs` ● `contracts/credit-manager/src/hls.rs` ● `contracts/credit-manager/src/liquidate.rs` ● `contracts/credit-manager/src/swap.rs` mars-incentives: ● `contracts/incentives/src/astro_incentives.rs` ● `contracts/incentives/src/query.rs` mars-health: ● `contracts/health/src/compute.rs` mars-rover-health-computer: ● `packages/health-computer/src/health_computer.rs` mars-vault: ● `contracts/vault/*` mars-account-nft ● `contracts/account-nft/src/execute.rs` ● `contracts/account-nft/src/helpers.rs` mars-zapper-astroport ● `contracts/v2-zapper/astroport/src/lp_pool.rs` mars-rewards-collector-base: ● `contracts/rewards-collector/base/src/contract.rs` mars-swapper-base: ● `contracts/swapper/base/src/contract.rs` mars-red-bank: ● PR https://github.com/mars-protocol/contracts/pull/372
Fixes verified at commit	`01364db126c077abd6dc70799692de4551fc08d8` During the fix review, the following additional pull requests were reviewed, which introduced additional features and addressed issues: ● https://github.com/mars-protocol/contracts/pull/404 ● https://github.com/mars-protocol/contracts/pull/405 ● https://github.com/mars-protocol/contracts/pull/406 ● https://github.com/mars-protocol/contracts/pull/409 Note that besides these pull requests, only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

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 audited update introduces various new features, namely LP token pricing, integration with Astroport incentives, managed vaults, support for trading any assets, providing/withdrawing liquidity in Astroport, and external routing.

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**	The audited codebase is large with several external dependencies/integrations, which increases the complexity.
Code readability and clarity	**High**	-
Level of documentation	**High**	-
Test coverage	**High**	92.3% test coverage

Summary of Findings

No	Description	Severity	Status
1	Total liquidity tokens are incorrectly increased, causing lower rewards	Critical	Resolved
2	Attackers can bind vault account ID to forcefully cause a loss for users	Critical	Resolved
3	Liquidatee’s staking rewards are lost	Major	Resolved
4	Users cannot stake new liquidity tokens on Astroport	Major	Resolved
5	Potentially outdated configurations stored in the vault	Major	Resolved
6	Incorrect liquidity tokens unstaked for `ActionAmount::Exact`	Major	Resolved
7	Using `AstroportSpot` can introduce price manipulation risk	Minor	Acknowledged
8	Vault unlock may revert because of cooldown end timing of a position	Informational	Resolved
9	Duplicate code	Informational	Acknowledged
10	Inconsistent use of `u32::from` and `as u32` for type conversion	Informational	Resolved
11	Redundant check for `!rewards.is_empty()` in `claim_lp_rewards`	Informational	Resolved
12	Incorrect token referenced in comment	Informational	Resolved
13	Unnecessary address lookup	Informational	Resolved
14	Missing validations	Informational	Partially Resolved
15	Redundant code setting `accumulated_pnl` and `accumulated_fee` to zero	Informational	Resolved

Detailed Findings

1. Total liquidity tokens are incorrectly increased, causing lower rewards

Severity: Critical

In contracts/incentives/src/astro_incentives.rs:241-247, the decrement_staked_lp function increases the ASTRO_TOTAL_LP_DEPOSITS state amount instead of decreasing it. This is incorrect because this function is called when users withdraw their liquidity tokens in contracts/incentives/src/astro_incentives.rs:91.

Consequently, the computed rewards will be less than intended due to the inflated number of liquidity tokens (see contracts/incentives/src/helpers.rs:226-243), causing a loss of rewards for stakers.

Recommendation

We recommend modifying the decrement_staked_lp function to use checked_sub.

Status: Resolved

2. Attackers can bind vault account ID to forcefully cause a loss for users

Severity: Critical

In contracts/vault/src/execute.rs:22, the bind_credit_manager_account function can only be called by the credit manager contract to bind the vault account ID. After binding it, users can interact with the vault to deposit and redeem their tokens with the accrued reward.

The issue is that anyone can call the CreateCreditAccountV2 message while specifying the vault address to bind it to (see contracts/credit-manager/src/execute.rs:69-87), allowing them to manage the funds deposited in the vault. Suppose an attacker gets control of the vault's account ID, and users start depositing funds. In that case, attackers can steal funds by purposely borrowing a huge amount of tokens and using another address to liquidate the vault for profit.

Recommendation

We recommend modifying the `create_credit_account` function so that only the contract owner of the vault can bind the account ID.

Status: Resolved

3. Liquidatee's staking rewards are lost

Severity: Major

In `contracts/credit-manager/src/liquidate_astro_lp.rs:47`, the `liquidate_astro_lp` function sends the `UnstakeAstroLp` message to liquidate the borrower's collateral from the incentives contract. The incentives contract will compute the rewards and transfer them to the credit manager contract, as seen in `contracts/incentives/src/astro_incentives.rs:346-353`.

The issue is that the credit manager contract ignores the accrued rewards, causing a loss of rewards for the borrower. Instead, the borrower should receive the accrued rewards for the staked period even though they are liquidated. For comparison, the `stake_lp` function accrues user rewards after querying the incentives contract in `contracts/credit-manager/src/stake_astro_lp.rs:17-36`.

Recommendation

We recommend accruing the liquidatee's staking rewards in the `liquidate_astro_lp` function.

Status: Resolved

4. Users cannot stake new liquidity tokens on Astroport

Severity: Major

In `contracts/incentives/src/query.rs:95`, the `query_unclaimed_astro_lp_rewards` function dispatches a `PendingRewards` query message to Astroport to retrieve the number of rewards accrued by the incentives contract. This function is called to accrue user rewards before increasing their liquidity token balance, which influences the reward amount.

The issue is that the `PendingRewards` query assumes the incentives contract has staked liquidity tokens previously before calling the function. Otherwise, an error will occur when retrieving the position (see the `query_pending_rewards` and `load_position` functions). This means that the `StakeAstroLp` message will always fail for new liquidity tokens as Astroport assumes the incentives contract to have an existing balance, which is not the case.

Consequently, users cannot stake their liquidity tokens on Astroport, breaking the protocol's intended functionality.

Recommendation

We recommend only performing the query if the total amount of liquidity tokens deposited exceeds zero.

Status: Resolved

5. Potentially outdated configurations stored in the vault

Severity: Major

In `contracts/vault/src/instantiate.rs:37-46`, the vault contract's `init` function sets the `ORACLE`, `HEALTH`, and `ACCOUNT_NFT` addresses to the query response of the credit manager contract. This is problematic because if the credit manager's contract owner updates these values to new addresses in `contracts/credit-manager/src/update_config.rs:31-91`, they will not be reflected in the vault contract.

For example, if the oracle contract is unmaintained and updated to a new address, the vault contract could potentially be exposed to stale and incorrect prices, causing incorrect accounting.

Recommendation

We recommend removing the `ORACLE`, `HEALTH`, and `ACCOUNT_NFT` states and always query the credit manager contract for the latest addresses.

Status: Resolved

6. Incorrect liquidity tokens unstaked for `ActionAmount::Exact`

Severity: Major

In `contracts/credit-manager/src/unstake_astro_lp.rs:32-41`, the `unstake_lp` function computes the amount of liquidity tokens to unstake by deducting the the amount from `ActionAmount::Exact` from the user's balance. This is incorrect because when the user specifies `ActionAmount::Exact`, it indicates that the user wants to withdraw the exact amount of liquidity tokens.

For example, if the user's balance is 1000 tokens and `ActionAmount::Exact` is specified as 100, the unstaked amount is incorrectly 900 instead of 100 tokens.

Recommendation

We recommend using the specified amount from `ActionAmount::Exact` as the amount to be unstaked.

Status: Resolved

7. Using `AstroportSpot` can introduce price manipulation risk

Severity: Minor

In `contracts/oracle/wasm/src/price_source.rs:401`, the `query_price` function includes an option to fetch a spot price using `WasmPriceSource::AstroportSpot`. However, consuming spot prices into Mars protocol can introduce price manipulation risk and should therefore be avoided.

Recommendation

We recommend removing the `AstroportSpot` option from the `WasmPriceSource` enum.

Status: Acknowledged

8. Vault unlock may revert because of cooldown end timing of a position

Severity: Informational

In `contracts/vault/src/execute.rs:209`, it is required that the user-provided vault tokens are equal to the sum of the unlocked vault tokens. This sum is calculated by iterating over all positions where the field `cooldown_end` is less than or equal to the current block timestamp.

This logic can make it very difficult for a user to send the correct amount of tokens when the transaction is initiated at a time close to the `cooldown_end` value of a position. Then, the sum will vary depending on whether the transaction was included in a block that was produced slightly earlier or slightly later than this timestamp.

If the amounts do not match, the user has wasted gas and needs to try unlocking again.

Recommendation

We recommend allowing the user to send more tokens when unlocking. The additional tokens can then be reimbursed to the user.

Status: Resolved

9. Duplicate code

Severity: Informational

In `contracts/oracle/wasm/src/lp_pricing.rs:26-42` the code to fetch the price of `coin0` and `coin1` is duplicated in lines `157-173`. Duplicate code can negatively affect the maintainability of the codebase.

Recommendation

We recommend extracting the code to a helper function to avoid duplication.

Status: Resolved

10. Inconsistent use of `u32::from` and `as u32` for type conversion

Severity: Informational

In `contracts/oracle/wasm/src/lp_pricing.rs:118`, `coin0_decimals` is converted to a `u32` type using `u32::from(coin0_decimals)`, however, in `contracts/oracle/wasm/src/lp_pricing.rs:218` the `coin0_decimals` is cast to a `u32` type using `coin0_decimals as u32`.

Recommendation

We recommend consistently using `u32::from` function to convert to `u32` type since it provides safe, lossless conversion and improved error handling.

Status: Resolved

11. Redundant check for `!rewards.is_empty()` in `claim_lp_rewards`

Severity: Informational

In `contracts/credit-manager/src/claim_astro_lp_rewards.rs:33`, the `claim_lp_rewards` function includes a redundant check `!rewards.is_empty()` since that check was already performed in line `19` of the same function.

Recommendation

We recommend removing the additional check `!rewards.is_empty()` in line `33`.

Status: Resolved

12. Incorrect token referenced in comment

Severity: Informational

In `contracts/vault/src/execute.rs:182`, there is a comment stating `"// check that only the expected base token was sent"`. However, it is the vault token that is being passed to the `cw_utils::must_pay` function below the comment.

Recommendation

We recommend the comment is updated to `"// check that only the expected vault token was sent"`.

Status: Resolved

13. Unnecessary address lookup

Severity: Informational

In `contracts/incentives/src/astro_incentives.rs:290`, the `ensure_eq!` macro checks that the sender is the credit manager. However, the credit manager is passed in as `&addresses[&MarsAddressType::CreditManager]` when, in fact, the `credit_manager_addr` variable could be used since that has been set earlier in line `284`. This unnecessary address lookup is inefficient.

Recommendation

We recommend passing in `credit_manager_addr` to the `ensure_eq!` macro in line `290`.

Status: Resolved

14. Missing validations

Severity: Informational

There are several instances where validation is missing:

- In `contracts/oracle/wasm/src/price_source.rs:219`, the `validate` function returns the `price` variable for `WasmPriceSource::Fixed` option without validating that the price is greater than zero.
- In `contracts/credit-manager/src/stake_astro_lp.rs:26`, if the variable `amt` is greater than a user's LP coin balance, there will be an underflow error in `decrement_coin_balance`.
- In `contracts/credit-manager/src/stake_astro_lp.rs:27`, if `coin_balance` is zero, there will be an underflow error in `decrement_coin_balance`.

- In `contracts/vault/src/instantiate.rs:64`, the `cooldown_period` is not validated to be greater than zero.
- In `contracts/vault/src/instantiate.rs:75`, the `base_token` address is not checked to be a valid address.

Recommendation

We recommend implementing validations for the instances described above.

Status: Partially Resolved

The first two bullet points have been acknowledged and the last three have been resolved.

15. Redundant code setting `accumulated_pnl` and `accumulated_fee` to zero

Severity: Informational

In `contracts/vault/src/performance_fee.rs:72`, the `accumulated_pnl` and `accumulated_fee` are set to `Uint128::zero()`. However, this is redundant since these values will already be set to `Uint128::zero()` in the `default` function.

Recommendation

We recommend removing the redundant code.

Status: Resolved