## Audit Report

# Nolus Money Market

v1.0

January 20, 2023

# 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 Nolus Platform AG to perform a security audit of the Nolus Money Market CosmWasm 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://gitlab-nomo.credissimo.net/nomo/smart-contracts

Commit hash: `f52531d38a89b8cfa0c2d958142664f1f4f8da11`

## 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

Nolus Money Market is a DeFi lending protocol built on top of the Nolus chain. It enables users to lend and borrow assets as well as NLS token holders to earn protocol fees.

The audit scope includes the `treasury`, `dispatcher`, `timealarms`, `lpp`, `leaser`, `oracle`, `profit` and `lease` CosmWasm smart contracts.

# 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 | Lack of authorization for privileged operations in the `lpp` contract allows users to manipulate the market | Critical | Resolved |
| 2 | Price feeding mechanism could be permanently inhibited through market alarms | Critical | Resolved |
| 3 | Time alarms could be permanently disabled | Critical | Resolved |
| 4 | Lack of instantiation authorization in the `lease` contract allows attackers to instantiate it with arbitrary parameters and hence prevent liquidation and use arbitrary prices | Critical | Externally Resolved |
| 5 | Incomplete Neutron messages support may lead to inconsistent state | Major | Acknowledged |
| 6 | Lack of alternate recipient address validation may lead to loss of user's funds | Minor | Resolved |
| 7 | Oracle is not resilient to feeders errors or manipulation attempts | Minor | Resolved |
| 8 | Contract ownership cannot be transferred | Minor | Externally Resolved |
| 9 | Lack of price validation could lead to a division by zero error | Minor | Resolved |
| 10 | It is not possible to migrate `lease` contracts | Minor | Externally Resolved |
| 11 | Time alarms can be created for the past | Minor | Resolved |
| 12 | Lack of validation for the `UpdateParameters` message | Minor | Resolved |
| 13 | Lack of validation of the `Dispatcher` config parameters | Minor | Resolved |
| 14 | Lack of validation of the `leaser` instantiation parameters | Minor | Resolved |
| 15 | Potential algebraic overflows | Informational | Resolved |

| | | | |
|---|---|---|---|
| 16 | Overflow checks not enabled for release profile | Informational | Acknowledged |
| 17 | Custom access controls implementation | Informational | Acknowledged |
| 18 | Wrong variable name for `APR` | Informational | Resolved |
| 19 | Typographical error | Informational | Resolved |
| 20 | Additional logic included in the `RequestLoan`'s `new` function | Informational | Acknowledged |
| 21 | Outstanding `TODO`s are present in the codebase | Informational | Acknowledged |

## Code Quality Criteria

| Criteria | Status | Comment |
|---|---|---|
| Code complexity | Medium-High | The intensive use of traits, struct methods and closures, even when not strictly needed increases the code complexity. |
| Code readability and clarity | Low-Medium | Polymorphism and OOP concepts that are not explicitly designed for Rust, but that the language supports due to its flexibility, reduce the readability of the code. |
| Level of documentation | Low-Medium | The documentation has a high variance of quality and completeness between different contracts. |
| Test coverage | High | `cargo tarpaulin` reports 90.70% code coverage. |

# Detailed Findings

## 1. Lack of authorization for privileged operations in the `lpp` contract allows users to manipulate the market

**Severity: Critical**

The `lpp` contract does not implement an authorization logic for the execution of privileged operations and does not have the concept of a contract owner/administrator.

This implies that anyone can send `UpdateParameters` messages to update critical contract parameters like the `base_interest_rate`, `utilization_optimal` and `addon_optimal_interest_rate`, which can be used to manipulate the market.

### Recommendation

We recommend defining an administrator role and implementing authorization logic to restrict the execution of specific messages to that role.

**Status: Resolved**

## 2. Price feeding mechanism could be permanently inhibited through market alarms

**Severity: Critical**

The `oracle` allows contracts to subscribe to `MarketAlarms` in order to be notified when price changes trigger a particular condition.

When a `Feeder` sends the `FeedPrices` message to update an asset price, a `PriceAlarm` message is dispatched and sent to all subscribing contracts.

Since an `O(n^2)` unbounded loop through `MarkedAlarms` is needed to check and fire alarms, a significant number of subscriptions could lead to out of gas errors. Consequently, feeders will be permanently inhibited from providing new prices.

A malicious actor could develop a contract with a computationally expensive `PriceAlarm` handler and subscribe potentially multiple `MarketAlarms` in order to cause the `FeedPrices` message to always run out of gas.

Even without malicious intent, this issue is likely to occur since `lease` contract instances that users can instantiate from the `leaser` contract subscribe to alarms in the `oracle`.

### Recommendation

We recommend rethinking the alarm design with a pull-over-push approach. For example, off-chain services could be incentivized to execute alarms in a timely manner.

**Status: Resolved**

## 3. Time alarms could be permanently disabled

**Severity: Critical**

The `timealarms` contract allows other contracts to subscribe to `TimeAlarms` in order to be notified at a certain timestamp.

The `Notify` transaction is used to trigger all stored alarms with a timestamp less than the actual time and execute the `TimeAlarm` transaction in the respective contracts.

Since there is not a defined maximum number of stored alarms and a maximum amount of gas that `TimeAlarm` handlers can consume, the execution of `Notify` may run out of gas if a significant amount of alarms are registered. Consequently, no time alarm could be triggered anymore.

A malicious actor could intentionally deploy computationally expensive contracts that subscribe to `TimeAlarms`, consuming more gas than available and hence effectively disabling time alarms.

### Recommendation

We recommend rethinking the time alarm design. For example, off-chain services could be incentivized to execute time alarms in a timely manner. Alternatively, the execution permission of the `AddAlarm` message could be restricted to administrators/governance.

**Status: Resolved**

## 4. Lack of instantiation authorization in the `lease` contract allows attackers to instantiate it with arbitrary parameters and hence prevent liquidation and use arbitrary prices

**Severity: Critical**

During the execution of the `InstantiateMsg` of the `lease` contract, it is not enforced that its instantiator is the `leaser` contract.

This implies that an attacker could instantiate a `lease` contract without the intermediation of the `leaser`, which allows the attacker to provide arbitrary parameters. For example, he could

set a custom `oracle` address or `liability` configurations. This gives the attacker the ability to prevent liquidation and set arbitrary prices.

**Recommendation**

We recommend implementing authorization to ensure that `info.sender` is equal to the `leaser` contract address.

**Status: Externally Resolved**

The deployment of contracts on the Nolus chain is permissioned and requires a governance proposal, hence it is not possible for an attacker to instantiate a `lease` contract in this environment.

## 5. Incomplete Neutron messages support may lead to inconsistent state

**Severity: Major**

Contracts implement Neutron message specifications but are not implementing their specific `Sudo` handlers used for handling message responses.

This could lead to different inconsistencies and partial state updates in the protocol in case of errors or timeouts in the controlled chain.

Some examples of the aforementioned behavior exhibited in the code:

- In `contracts/profit/src/profit.rs:63`, a non acknowledged alarm can lead to profits not being transferred.
- In `contracts/lease/src/liquidation/alarm.rs:127`, not properly rescheduled alarms can impact the timing of liquidations.
- In `contracts/lpp/src/contract/borrow.rs:57`, failed attempts to transfer funds using `BankMsg::Send` through an interchain transaction are not handled.

**Recommendation**

We recommend completing the Neutron message support implementing required handlers as specified in the official documentation.

https://docs.neutron.org/neutron/interchain-txs/overview

**Status: Acknowledged**

## 6. Lack of alternate recipient address validation may lead to loss of user's funds

**Severity: Minor**

When sending a `ClaimRewards` message to the `lpp` contract, the user can specify an alternate reward recipient in the `other_recipient` parameter.

However, in `contracts/lpp/src/contract/rewards.rs:34`, during the execution of the `try_claim_rewards` function, there is no validation in place to check that the `other_recipient` address is valid.

An invalid address will lead to the permanent loss of the user's rewards.

**Recommendation**

We recommend validating the `other_recipient` address.

**Status: Resolved**

## 7. Oracle is not resilient to `feeders` errors or manipulation attempts

**Severity: Minor**

The `FeedPrices` message defined in `contracts/oracle/src/contract/exec.rs:61` enables `feeders` to submit their oracle prices on chain.

Since its associated handler `try_feed_prices` in `contracts/oracle/src/contract/feed.rs:92` does not perform any validation on inputs, it allows `feeders` to store incorrect data, for example a price equal to zero or multiple prices in the same block.

Also, the `get_price` function in `packages/marketprice/src/feed.rs:55`, which is responsible for providing the price of an asset, is returning the latest feeded one, without performing any calculation with other stored data points.

By not performing any checks, nor transforming the data through any function that might catch outliers and data points that are not representing the real market price, the protocol allows the price to be manipulated by just introducing one corrupted observation.

### Recommendation

We recommend validating inputs provided from `feeders` and implementing a strategy in order to avoid the aforementioned issues. For example, a time-weighted average price or a model to eliminate outliers could be used.

**Status: Resolved**

## 8. Contract ownership cannot be transferred

**Severity: Minor**

The `treasury`, `dispatcher`, `leaser`, `profit` and `oracle` contracts do not implement an ownership transfer mechanism.

Since the defined `owner` has the right to modify critical contract parameters and execute privileged messages, a compromise of its account would have devastating consequences for the protocol.

It is best practice to offer functionality to transfer the ownership to another account.

### Recommendation

We recommend implementing a mechanism to transfer the contract ownership to another account following the practices recommended in the issue "Custom access control implementation" below.

**Status: Externally Resolved**

The deployment of these contracts on the Nolus chain is performed at genesis and owner-related operations can only be executed through governance.

## 9. Lack of price validation could lead to a division by zero error

**Severity: Minor**

In `contracts/oracle/src/contract/feed.rs:67`, there is no validation to ensure that prices are not equal to zero.

Since throughout the codebase there are multiple calls to the `inv` function with the price value, there is a risk that the protocol performs a division by zero.

### Recommendation

We recommend validating the provided price to be greater than zero.

**Status: Resolved**

### 10. It is not possible to migrate `lease` contracts

**Severity: Minor**

In `contracts/leaser/src/cmd/borrow.rs:27-34`, the `lease` contract instantiate message is constructed with the `admin` parameter set to `None`.

This implies that the instantiated contract could not be migrated in case of a bug or a vulnerability disclosure.

#### Recommendation

We recommend setting the `admin` parameter to a trusted address, for example the `leaser` contract administrator.

**Status: Externally Resolved**

The Nolus chain where the contracts in scope of this audit are deployed leverages `authz` policies to allow governance to perform migrations if needed.

### 11. Time alarms can be created for the past

**Severity: Minor**

In `packages/time-oracle/src/alarms.rs:60-69` the `add` function is not enforcing that a new time alert's timestamp is in the future.

#### Recommendation

We recommend implementing a guard that returns an error if the provided timestamp is in the past.

**Status: Resolved**

### 12. Lack of validation for the `UpdateParameters` message

**Severity: Minor**

In `contracts/lpp/src/contract/config.rs:9-22` the `try_update_parameters` function is updating the contract parameters without validating them.

Since `base_interest_rate`, `utilization_optimal` and `addon_optimal_interest_rate` are `Percent` values, they should be validated to be in the `[0,1)` interval.

We classify this issue as minor since only the contract owner can set those values.

### Recommendation

We recommend implementing a validation mechanism for the mentioned parameters.

**Status: Resolved**

## 13. Lack of validation of `Dispatcher` config parameters

**Severity: Minor**

In `contracts/dispatcher/src/contract.rs:45-50`, the `tvl_to_apr` parameter is not validated during the `dispatcher` contract instantiation.

This could lead to incorrect `APR` calculations and to not covered `TVL` intervals.

We classify this issue as minor since only the contract owner can set this parameter.

### Recommendation

We recommend enforcing the `tvl_to_apr isValid` method call in order to validate parameter correctness.

**Status: Resolved**

## 14. Lack of validation of the `leaser` instantiation parameters

**Severity: Minor**

In `contracts/leaser/src/contract.rs:29`, during the `leaser` contract instantiation, not all the parameters are validated, such as `liability` and `repayment`.

As a result, incorrect configurations could be used in the contract, such as a `Liability` instance with an incorrect attribute relationship, for example, a `healthy` factor smaller than the `initial` factor or greater than the `max` factor.

### Recommendation

We recommend validating parameters with the same logic defined in `try_configure`.

**Status: Resolved**

## 15. Potential algebraic overflows

**Severity: Informational**

Protocol contracts are not making use of checked operators in order to guard against algebraic overflows.

For example:

- In `contracts/lpp/src/state/deposit.rs:79`. `globals.balance_nlpn` is incremented by `deposited_nlpn` without checking for overflows. This assignment can be dangerous if the price of `NLP` is in the `(0,1]` range. In that case the entire global value could overflow, which would affect rewards calculation.
- In `contracts/leaser/src/state/leaser.rs:23`, an increment of an integer could cause an overflow eventually.

Since `overflow-checks` are enabled in `Cargo.toml` profile, the execution will panic if an overflow occurs; however it should be better to be handled gracefully with an error message.

### Recommendation

We recommend using checked operators from the standard library instead of the unchecked math operators.

**Status: Resolved**

## 16. Overflow checks not enabled for release profile

**Severity: Informational**

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

- `contracts/treasury/Cargo.toml`
- `contracts/dispatcher/Cargo.toml`
- `contracts/timealarms/Cargo.toml`
- `contracts/lpp/Cargo.toml`
- `contracts/leaser/Cargo.toml`
- `contracts/oracle/Cargo.toml`
- `contracts/profit/Cargo.toml`
- `contracts/lease/Cargo.toml`

While enabled implicitly through the workspace manifest, a 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: Acknowledged**

## 17. Custom access controls implementation

**Severity: Informational**

The `treasury`, `dispatcher`, `leaser`, `profit` and `oracle` 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**

## 18. Wrong variable name for `APR`

**Severity: Informational**

In `contracts/dispatcher/src/state/tvl_intervals.rs:72`, a variable is named `arp`, when it aims to reflect the `APR` value.

### Recommendation

We recommend renaming the variable to `apr`.

**Status: Resolved**

### 19. Typographical error

**Severity: Informational**

In `contracts/leaser/src/error.rs:27`, an error variant is named `IvalidLiability`, while its correct spelling is `InvalidLiability`.

#### Recommendation

We recommend renaming the enum variant to `InvalidLiability`.

**Status: Resolved**

### 20. Additional logic included in the `RequestLoan`'s `new` function

**Severity: Informational**

In `contracts/lease/src/contract/state/request_loan.rs:30` the `new` function contains much more business logic than just creating the structure. Besides creating the structure, it performs validations and executes an `OpenLoanReq`. This goes against best practices

#### Recommendation

We recommend following Rust design patterns and avoid additional logic on the `new` function. See Constructor - Rust Design Patterns for reference.

**Status: Acknowledged**

### 21. Outstanding `TODO`s are present in the codebase

**Severity: Informational**

During the audit, `TODO` comments were found in the following lines:

- `contracts/dispatcher/src/state/tvl_intervals.rs:62`
- `contracts/lease/src/contract/repay.rs:53`
- `contracts/lease/src/contract/state/active.rs:48`
- `contracts/lease/src/contract/state/active.rs:107`
- `contracts/lease/src/contract/state/active.rs:143`
- `contracts/lease/src/lease/mod.rs:206`
- `contracts/lease/src/loan/mod.rs:237`
- `contracts/lpp/src/state/total.rs:78`
- `contracts/oracle/src/state/supported_pairs/mod.rs:27`
- `contracts/profit/src/profit.rs:58`
- `contracts/profit/src/profit.rs:69`
- `packages/currency/src/lease.rs:108`

- `packages/currency/src/lease.rs:129`
- `packages/finance/src/ratio.rs:7`
- `packages/finance/src/fractionable/duration.rs:25`
- `packages/finance/src/fractionable/mod.rs:20`
- `packages/finance/src/fractionable/mod.rs:45`
- `packages/finance/src/fractionable/percent.rs:38`
- `packages/finance/src/price/mod.rs:86`
- `packages/marketprice/src/market_price.rs:86`

This suggests that the contracts might still be under development and not yet ready for mainnet deployment.

### Recommendation

We recommend resolving the `TODO` comments and/or removing them from the codebase.

**Status: Acknowledged**