Security Audit Report

# Neptune Updates 2

v1.1

December 23, 2024

# Table of Contents

## License

# Disclaimer

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 Cryptech Developments Ltd. to perform a security audit of Neptune Protocol.

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/cryptechdev/neptune-protocol |
| Commit | `61e5ae244005ca5bee09609dca69dfcb28d03415` |
| Scope | In scope of this audit were all changes since our previous audit, which was performed at commit `addc118b4b21be17ecdf2e1705dd7f208ed3946d`, except the `swap` feature and the `querier` contract. |
| Fixes verified at commit | `57403acb40211f76ae75316a51cf17d59aaeb3e0`<br><br>The following checksums were generated using `cosmwasm/optimizer` version 0.15.1: |

| | |
|---|---|
| | - `26239ed795adbe5f6de00f96363743cd4b745bfa466a80608c2b7ecd34408fa7` - `flash_loan_receiver.wasm`<br>- `4d03f72da921d8dfccd4706b12bc0f67f84d62cfe7bb8c2682b294962a493f59` - `interest_model.wasm`<br>- `f5d6196ff8f337f5363431ce879578582c3209a962008d1fba1d923463c76918` - `market.wasm`<br>- `4ee8331334091c38e05207567f72418835b1e1642f8ba9fbc47e50a82de7dccf` - `price_oracle.wasm`<br>- `baa6867d097eb139e5af93f808d19897d4f5befd7d074e2057821e24c17ae482` - `token.wasm`<br><br>The checksum for `querier.wasm` is not included in this report.<br><br>Note that 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

Neptune Protocol is a decentralized lending protocol with a novel PID-transformed interest rate curve. This report focuses only on the updates since our previous audit, covering the newly introduced features and modifications.

# 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**, **Partially Resolved**, 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** | Due to the novelty of the interest rate mechanism, the reliance on different off-chain components, and a large feature set (with features such as flash loans), the code base is non-trivial in several places. |
| Code readability and clarity | **Medium-High** | - |
| Level of documentation | **Low-Medium** | The documentation is limited. |
| Test coverage | **Medium** | cargo-llvm-cov reports the following coverage:<br>- `73.93%` function coverage<br>- `83.12%` line coverage<br>- `64.67%` region coverage |

## Summary of Findings

| No | Description | Severity | Status |
|---|---|---|---|
| 1 | Incorrect validation of flash loan transactions | Major | Resolved |
| 2 | Access control inconsistency in flash loans | Major | Resolved |
| 3 | Potential denial of service via unbounded token unlocks | Minor | Resolved |
| 4 | Inefficient handling for zero amounts | Minor | Resolved |
| 5 | Additional funds sent to the contract are lost | Informational | Acknowledged |
| 6 | Imprecise debug assertions | Informational | Acknowledged |
| 7 | Unlimited bonding and unbonding can lead to self-griefing | Informational | Acknowledged |
| 8 | Potential information leakage in error messages | Informational | Acknowledged |
| 9 | Inconsistent attribute keys and hardcoded strings | Informational | Resolved |
| 10 | “Migrate only if newer” pattern is not followed | Informational | Acknowledged |
| 11 | Redundant computation in flash loan execution | Informational | Resolved |
| 12 | Overuse of `fold` and numerical selectors | Informational | Resolved |
| 13 | Missing sanity check for `tokens_per_weight` in `validateParams` | Informational | Resolved |
| 14 | Use of magic numbers decreases maintainability | Informational | Resolved |

# Detailed Findings

## 1. Incorrect validation of flash loan transactions

**Severity: Major**

In `contracts/token/src/query.rs`, the `get_flash_loan_weighted_stake` function aggregates all flash loan weights derived from stakes of the user, into a single coefficient used to determine the user's permission to flash loan. This is implemented in `contracts/market/src/execute.rs:181-186`.

However, this function uses the wrong field of `BondDurationSettings` structure: instead of `flash_loan_weight` it refers to `health_weight`. This error seems to stem from code duplication between the functions `get_flash_loan_weighted_stake`, `get_health_weighted_stake`, and `get_gov_weighted_stake`.

As a consequence, the validation of flash loan amounts functions incorrectly: flash loan transactions are accepted or rejected depending on the health weights and not on dedicated flash loan weights.

### Recommendation

We recommend using the correct field `flash_loan_weight`, as well as removing code duplications and improving code readability.

**Status: Resolved**

## 2. Access control inconsistency in flash loans

**Severity: Major**

In `contracts/market/src/execute.rs:176-186`, the updated `execute_borrow_flash_loan` function introduces a mechanism that allows unrestricted flash loans for whitelisted addresses and enforces a stake-based limit for others. However, the message execution remains restricted to only `FlashLoanWhitelist` in `packages/neptune-money-market/src/market.rs:184`, contradicting the access control logic.

The update intends to allow both whitelisted users (who skip the ratio check) and users who pass the ratio check to execute the function. Whitelisted users would have the privilege of skipping the ratio check. However, the `BorrowFlashLoan` message cannot be called by anyone who is not whitelisted.

### Recommendation

We recommend allowing anyone to execute the `BorrowFlashLoan` message if that is the intended behavior.

**Status: Resolved**

## 3. Potential denial of service via unbounded token unlocks

**Severity: Minor**

In `contracts/token/src/execute.rs:533-594`, the `execute_create_token_unlock` function allows any user to create an unlimited number of token unlocks for any address without restrictions. This could lead to a scenario where a malicious actor creates a large number of active unlocks for a target address, potentially resulting in increased gas costs when the target user attempts to claim their unlocks.

While there is no direct loss of funds for the victim, the increased gas fees associated with iterating through a long vector of unlocks during the claiming process could make it expensive or even impossible to claim tokens, especially if the unlocks are set for very long durations. The impact of this issue is limited, as it requires a malicious user to cover relevant gas fees and would necessitate separate attacks against each victim, providing little incentive for such behavior.

This issue has been independently reported by the client.

### Recommendation

We recommend implementing access control restrictions in `packages/neptune-money-market/src/token.rs:278` for the `CreateTokenUnlock` message to limit its usage to trusted roles. Alternatively, we suggest considering the implementation of unlock creation limits to reduce the feasibility of such attacks.

**Status: Resolved**

## 4. Inefficient handling for zero amounts

**Severity: Minor**

The `execute_unbond` and `execute_rebond` functions in `contracts/token/src/execute.rs:104` and `218` do not check for a zero `amount` input. If a user attempts to unbond or rebond zero tokens, the functions proceed without returning an error. This results in the following:

- **Unnecessary state changes:** The functions emit events and update state variables (`state`, `bonded`, `unbonding`) even though no actual unbonding occurs.
- **Wasted gas:** The user pays gas for a transaction that effectively does nothing.
- **Potential for confusion:** The emitted events and state changes might mislead users or indexing services into believing an unbonding or rebonding action took place.

This issue occurs because the code iterates through the `bonded` list and decrements `amount_remaining` until it reaches zero. If the initial `amount` is zero, `amount_remaining` will also be zero, and the loop will terminate without triggering any error.

### Recommendation

We recommend adding a check at the beginning of the `execute_unbond` and `execute_rebond` functions to ensure that `amount` is not zero. If it is, the functions should return an appropriate error message.

**Status: Resolved**

## 5. Additional funds sent to the contract are lost

**Severity: Informational**

In `contracts/token/src/contract.rs`, within the `execute` function, handling of the following messages does not validate `info.funds`:

- `Unbond`
- `Rebond`
- `ClaimUnbonded`
- `ClaimRewards`
- `ClaimTokenUnlock`
- `ReclaimTokenUnlock`
- `Cascade`

These messages are not supposed to receive any funds. However, if the user accidentally attaches funds to them, they will stay in the contract without the possibility of a refund.

Similarly, in `contracts/market/src/contract.rs`, variable `info.funds` is not validated when handling the following messages:

- `Borrow`
- `AssertFlashLoanRepaid`
- `BorrowFlashLoan` (public if the whitelist is empty)
- `WithdrawCollateral`
- `DistributeInterest`
- `ClaimWarChest`

- `Cw20HookMsg::Redeem`

While blockchains generally do not protect users from sending funds to wrong accounts, reverting extra funds increases the user experience.

### Recommendation

We recommend checking that messages contain only the expected funds or none if there is no payment required, by utilizing standard CosmWasm functions `nonpayable` and `must_pay`.

**Status: Acknowledged**

## 6. Imprecise debug assertions

**Severity: Informational**

In `packages/neptune-money-market/src/token.rs:80-93`, the `Bonded` struct is declared. Implementations of `Ord` and `PartialOrd` traits are generated automatically for this structure by using `derive` macro annotation. Automatic implementation of these traits takes into account all fields of the structure: `cooldown`, `cascade`, `last_stake_acc`, `amount`. When two instances of this type have first fields equal, the comparison continues using second, third and fourth fields.

Now, in `contracts/token/src/execute.rs:967-987` the function `insert_bond` is defined. This function finds a place for insertion using a custom comparator which first compares fields `cooldown`, then `cascade`, and then `last_stake_acc`. The field `amount` does not affect the result of the comparison.

Finally, this distinction between automatically generated comparator logic and manually implemented in `insert_bond` becomes important when the order is checked using expressions like `debug_assert!(is_sorted(bonded.iter()))` in lines `contracts/token/src/execute.rs`:`75`, `187`, `305`, `813`, `907`, `968`, and `1050`. These assertions use automatically generated implementations of `Ord` and `PartialOrd` traits.

Since the construction of the `BONDED` storage vector and assertions use different comparators, these assertions can fail during valid operation of the smart contract in the development environment.

### Recommendation

We recommend unifying the comparison logic between `insert_bonded` and the calls to `is_sorted`.

**Status: Acknowledged**

## 7. Unlimited bonding and unbonding can lead to higher gas consumption

**Severity: Informational**

The current design of the bonding and unbonding functions allows users to execute an unlimited number of bonds and unbonds over a given duration. This unrestricted behavior creates a potential self-griefing scenario. If a user performs a large number of bonds and unbonds across different blocks within the same duration, they will create multiple unconsolidated entries (due to different `last_stake_acc` values) until they claim rewards, which triggers consolidation. While funds will not get stuck, as unbonding can be done in multiple transactions, users have to pay high gas fees for performing further bonding and unbonding actions for the same duration.

### Recommendation

We recommend implementing UI warnings to let users know that multiple bondings for the same duration will result in much higher gas fees.

**Status: Acknowledged**

## 8. Potential information leakage in error messages

**Severity: Informational**

The use of the recently introduced `Located` type in error reporting captures the file and line number where an error occurs. While helpful for debugging, including this information in error messages exposed to users could reveal internal code structure and degrade the user experience by providing unhelpful information.

### Recommendation

We recommend reviewing the use of `Located` type in error messages. Consider logging location information internally for debugging while providing more generic error messages to users.

**Status: Acknowledged**

## 9. Inconsistent attribute keys and hardcoded strings

**Severity: Informational**

The following instances illustrate code paths where consistency can be improved:

- In `contracts/token/src/execute.rs`, the `execute_bond`, `execute_unbond`, and `execute_claim_rewards` functions use the attribute key "duration". This is inconsistent with the existing DURATION constant. We recommend replacing "duration" with the DURATION constant in execute_bond, execute_unbond, and execute_claim_rewards for consistency.

- Throughout contracts/token/src/execute.rs, several instances of hardcoded strings are used for attribute keys and event fields, including "state", "bonded", "unclaimed", "unbonding", "unlock", and "unlocks". Similar to the existing constants in packages/neptune-money-market/src/market.rs, we recommend defining constants for these strings to improve consistency and maintainability.

### Recommendation

We recommend applying the recommendations mentioned above.

**Status: Resolved**

## 10. "Migrate only if newer" pattern is not followed

**Severity: Informational**

The contracts within the scope of this audit are currently migrated without regard to their version. This can be improved by adding validation to ensure that the migration is only performed if the supplied version is newer.

### Recommendation

We recommend following the "migrate only if newer" pattern defined in the CosmWasm documentation.

**Status: Acknowledged**

## 11. Redundant computation in flash loan execution

**Severity: Informational**

In contracts/market/src/execute.rs:135-138, the boolean value whitelisted is computed. This value is used in line 184 in combination with weighted_stake_value to reject the flash loan transaction if the sender is neither in the whitelist nor has enough stake.

However, if the sender is whitelisted (whitelisted is true), weighted_stake_value has no effect, and the computation in lines 140-183 becomes redundant, wasting computational resources and hence gas.

### Recommendation

We recommend optimizing the gas consumption of flash loans by computing the `weighted_stake_value` only when the sender is not whitelisted.

**Status: Resolved**

## 12. Overuse of `fold` and numerical selectors

**Severity: Informational**

The codebase utilizes the `fold` function combinator numerous times. While this higher-order function can be useful in certain scenarios, it often reduces code readability compared to chains of simpler combinators like `map` and `sum`.

Instances where `fold` could be replaced with `map` and `sum` for improved readability:

- `token/src/query.rs`:`109-116`, `137-139`, `158-165`, `175-181`, `198`, and `218`
- `packages/neptune-test/src/market/liquidate.rs`:`779-781`, `786-788`

Additionally, the codebase sometimes uses numerical selectors (`.0` and `.1`) instead of pattern matching, which can reduce readability. Examples can be found in `token/src/query.rs` in lines `194-199` and `214-219`.

### Recommendation

We recommend replacing the aforementioned `fold` usages with chained calls to `map` and `sum`. Additionally, we recommend utilizing pattern matching to replace expressions like `item.0` and `.1` with human-readable variables.

**Status: Resolved**

## 13. Missing sanity check for `tokens_per_weight` in `validateParams`

**Severity: Informational**

In `contracts/token/src/contract.rs:196`, the `validateParams` function validates parameters during the initialization of the token contract. However, it fails to ensure that the `tokens_per_weight` parameter is greater than zero.

If `tokens_per_weight` is initialized as zero, the token calculation per weight mechanism will be ineffective, as no tokens will be allocated per weight unit. This could lead to system-wide failure in token distribution and disrupt all contract operations dependent on this value. Since `tokens_per_weight` cannot be reset after initialization, the only solution would be to redeploy or upgrade the contract.

#### Recommendation

We recommend adding a check to ensure that `tokens_per_weight` is greater than zero during contract initialization.

**Status: Resolved**

### 14. Use of magic numbers decreases maintainability

**Severity: Informational**

Throughout the codebase, hard-coded number literals without context or a description are used. Using such "magic numbers" goes against best practices as they reduce code readability and maintenance as developers are unable to easily understand their use and may make inconsistent changes across the codebase. Instances of magic numbers are listed below:

- In `contracts/market/src/execute.rs:268` and `packages/neptune-money-market/src/market.rs:293-296`, the parameters `interest_fee`, `stake_collateral_ratio`, `staking_health_modifier`, and `stake_flash_loan_ratio` are required to be less than or equal to 100%. This choice is unclear and not documented.
- In `contracts/token/src/contract.rs:72`, the "zero" address `inj1qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqe2hm49` is used, which is specific to the Injective project and might be not obvious.

#### Recommendation

We recommend declaring magic numbers as constants with descriptive names and documenting rationale behind the choice of their specific values.

**Status: Resolved**