Audit Report

Amulet Neutron PoS Strategy

v1.0

February 7, 2024

Table of Contents

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.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

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

Introduction

Purpose of This Report

Oak Security has been engaged by the Neutron Audit Sponsorship Program and Gaia Projects Limited to perform a security audit of the Amulet Neutron PoS Strategy smart contract.

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/Amulet-Dev/protocol
Commit	`9e0dd27dd996d6a003bc5b48b488c237d349fcd6`
Scope	All contracts were in scope.
Fixes verified at commit	`856de1fa616ca7fb73541893784c94dbe04a5a26` Note that changes to the codebase beyond fixes after the initial audit have not been in the scope of our fixes review.

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 audit covers the Neutron PoS strategy contract of the Amulet protocol, a decentralized finance lending protocol deployed on the Neutron chain. The strategy contract interacts with the hub and vault contract to allow users to earn staking rewards and yields executed by the strategies.

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	**High**	The project's architecture is novel compared to other CosmWasm codebases within the ecosystem. The goal of the architecture is to build platform-agnostic smart contracts with as few external dependencies as possible, which creates complexity due to requiring core API traits, wrappers, and parsers to be implemented for interacting with the core logic. Additionally, integration with Neutron's ICA and ICQ features increases the complexity.
Code readability and clarity	**Low-Medium**	Although common programmatic patterns were used, the non-standard architecture made execution paths harder to follow.
Level of documentation	**Medium**	Documentation is available in the strategies.md file.
Test coverage	**Low**	`cargo tarpaulin` reports a test coverage of `30.75%`.

Summary of Findings

No	Description	Severity	Status
1	Unusable `Reconcile` and `ReceiveUnbonded` messages break the reconciliation process	Critical	Resolved
2	Pending deposits are incorrectly deducted twice and stuck in contract	Critical	Resolved
3	The delegation slot is not decreased when losses are detected	Critical	Resolved
4	Reconciliation height is not updated after completion	Critical	Resolved
5	Excess funds consumed due to storage not being cleared	Critical	Resolved
6	Interchain accounts cannot be accessed if the channel closes	Critical	Resolved
7	Incorrect ICQ balance query ID used breaks the reconciliation process	Major	Resolved
8	Incorrect `UnbondReadyStatusResponse` returned causes the vault contract to fail to start unbonding	Major	Resolved
9	Potential reconciliation failure due to uncapped fee amount	Major	Resolved
10	Rewards are not increased for zero fees or empty fee recipient	Major	Resolved
11	Delegation rewards computed will be zero if `MsgUndelegate` is called	Major	Partially Resolved
12	Incorrect claim amount calculation	Major	Resolved
13	Incorrect ICQ deposit amount is provided	Minor	Resolved
14	Actual unbonding accounting can be anonymously inflated	Minor	Resolved
15	Lack of input validation	Minor	Partially Resolved
16	State transition can be optimized	Informational	Acknowledged

17	Missing validation during `SetNextAdmin`	Informational	Acknowledged
18	Incorrect error returned during vault authorization	Informational	Acknowledged
19	Remove debugging messages	Informational	Acknowledged
20	Fix spelling errors	Informational	Resolved

Detailed Findings

1. Unusable `Reconcile` and `ReceiveUnbonded` messages break the reconciliation process

Severity: Critical

In `contracts/strategy/neutron-pos/msg.rs:69-74`, the `strategy` contract allows calling the `Reconcile` and `ReceiveUnbonded` messages. However, both messages cannot be called properly due to the validation in `contracts/strategy/neutron-pos/contract.rs:225-227`. The validation currently permits only the `Deposit` and `Donate` messages to attach funds and triggers an error if funds are attached to other messages.

This becomes problematic because both the `Reconcile` and `ReceiveUnbonded` messages require funds to be attached to cover fees in `contracts/strategy/neutron-pos/execute.rs:430` or when receiving the unbonding amount in line `457`. Furthermore, the `is_payable` function in `contracts/strategy/neutron-pos/msg.rs:106` does not consider `Reconcile` and `ReceiveUnbonded` as "payable" messages, leading to an error in `contracts/strategy/neutron-pos/contract.rs:226` if funds are attached to these messages.

Consequently, the reconciliation process cannot be triggered because Neutron IBC fees cannot be sent, and the unbonded funds cannot be sent back to the contract, preventing the contract from working as expected.

Recommendation

We recommend modifying the implementation to include `Reconcile` and `ReceiveUnbonded` in the list of “payable” functions.

Status: Resolved

2. Pending deposits are incorrectly deducted twice and stuck in contract

Severity: Critical

In `contracts/strategy/neutron-pos/reconcile/state_machine.rs:156`, the `restart_delegate_tx` function calls the `ReducePendingDepositByInflightDeposit` command with the in-flight deposit amount when retrying the `DelegateTx::Failed` reconciliation phase for `StrategyState::NoDeposits`. This is problematic because the pending deposit is

already reduced during the `TransferTx::Pending` phase, as seen in `contracts/strategy/neutron-pos/reconcile/state_machine.rs:125`.

Consequently, the pending deposits will be repeatedly deducted if the `DelegateTx::Pending` keeps failing, causing users' deposits to be stuck in the contract.

Recommendation

We recommend removing `ReducePendingDepositByInflightDeposit` in the `restart_delegate_tx` function.

Status: Resolved

3. The delegation slot is not decreased when losses are detected

Severity: Critical

In `contracts/strategy/neutron-pos/reconcile/state_machine.rs:371-381`, the `start_reconcile_process` function determines whether a loss has occurred (i.e., slashing penalty applied to the validator) by checking the difference between queried delegation balance and expected delegation balance. The expected delegation balance comes from the `slot_delegated` function in `contracts/strategy/neutron-pos/reconcile.rs:470`, which records the amount of funds delegated.

The problem occurs when a slashing penalty occurs, and the delegation amount is not deducted from the `detected_loss` amount. The `IncreaseDetectedLosses` command does not handle this, as it only increases the `slot_detected_losses` for the total deposits to be synced later.

Assume a scenario where there is a delegation of 100 funds. In the first reconciliation, a slashing penalty of 10 is applied, and the remaining funds delegated on-chain is 90 (100-90). However, the `slot_delegated` amount remains at 100.

In the second reconciliation, no slashing action occurred, so the ICQ result returns 90. This is problematic because the reconciliation process will compute the detected loss to be 10 (100-90) in `contracts/strategy/neutron-pos/reconcile/state_machine.rs:376`. The correct detected loss value should be 0 because no slashing penalty is actually applied.

Consequently, the detected loss will be incorrect, causing a loss of funds for the users.

Recommendation

We recommend modifying the `IncreaseDetectedLosses` command to reduce the `slot_delegated` amount by the detected loss.

Status: Resolved

4. Reconciliation height is not updated after completion

Severity: Critical

In `contracts/strategy/neutron-pos/reconcile/state_machine.rs:469-473`, the `after_withdraw_tx` function does not call the `SetLastReconcileHeightToCurrentHeight` command when setting the `ActiveDepositsPhase` to `Idle`. This is problematic because when the reconciliation process is called again, the `start_reconcile_process` function in line `343` will use the outdated `last_reconcile_height` value, causing the fees computation, loss detection, and unbonded requests to be triggered again.

Consequently, triggering reconciliation with the outdated reconciled height will cause larger fees to be computed, excess deduction of slashing penalty, and unnecessary unbonded requests to be issued, resulting in a loss of funds.

Recommendation

We recommend adding the `SetLastReconcileHeightToCurrentHeight` command in `contracts/strategy/neutron-pos/reconcile/state_machine.rs:469`.

Status: Resolved

5. Excess funds consumed due to storage not being cleared

Severity: Critical

In `contracts/strategy/neutron-pos/reconcile/state_machine.rs:623-627`, the `restart_delegate_tx` function sets the in-flight deposit to include the fees if the fee recipient is not specified. This happens during the `DelegateTx::Failed` phase with pending in-flight deposits and fees.

The issue is that the fee is not cleared after adding to the in-flight deposit. Assume the first reconciliation is called with an invalid fee recipient on the remote chain, so it fails. In the second reconciliation call, the reconciler does not provide the fee recipient. This causes the in-flight deposit amount to be updated to include the fees, as seen in line `626`.

The second reconciliation fails, and a third reconciliation is required. Since the in-flight fees are not removed from the storage, and the in-flight deposit includes it, excess funds will be sent to the remote chain in line `632` if the reconciler provides a fee recipient or line `627` if not.

Consequently, the sent funds will unknowingly consume excess ICA balance instead of the delegation rewards, potentially causing the `TransferInflightUnbonded` command to fail due to insufficient funds.

Recommendation

We recommend adding the `ClearInflightFee` command in line `625`.

Status: Resolved

6. Interchain accounts cannot be accessed if the channel closes

Severity: Critical

The strategy contract creates interchain accounts on the remote chain for each validator to perform staking operations. The interchain messages will be processed in the same order flow due to the ordered channel and close if an error occurs in the sudo handler or a timeout occurs.

The issue arises since there are no entry points for the admin to call the `RegisterInterchainAccount` message to recover access to the interchain account in case the channel closes. This can happen when the `SudoMsg::Timeout` message gets entered or when the `reconcile` function called by the `handle_response` function fails, see `contracts/strategy/neutron-pos/sudo.rs:116`.

Consequently, the admin cannot recover access to the interchain account, preventing the protocol from working as expected.

Recommendation

We recommend implementing an entry point for the admin to call `RegisterInterchainAccount` if the channel closes due to an error or timeout.

Status: Resolved

7. Incorrect ICQ balance query ID used breaks the reconciliation process

Severity: Major

In `contracts/strategy/neutron-pos/reconcile.rs:406`, the `last_balance_icq` function loads the balance query ID using the

`slot_delegation_icq_id` storage. This is incorrect because the loaded query is for the delegation ICQ query ID, not the balance query ID.

Consequently, the function will error in line `425`, preventing validator slots with active deposits from being reconciled properly and breaking the whole reconciliation process.

Recommendation

We recommend using `slot_balance_icq_id` to load the balance ICQ query ID.

Status: Resolved

8. Incorrect `UnbondReadyStatusResponse` returned causes the vault contract to fail to start unbonding

Severity: Major

In `contracts/strategy/neutron-pos/query.rs:13`, the `unbond_ready_status` function returns `UnbondReadyStatus::Later` with the `hint` as `None` if no validator is unbonding. This is problematic because the vault contract only allows unbonding to be initiated if the returned status is `UnbondReadyStatus::Ready`.

Consequently, the vault contract cannot trigger an unbonding request to the strategy contract, preventing the contract from working as expected.

Recommendation

We recommend returning `UnbondReadyStatus::Ready` if there are no previously requested delegations to unbond.

Status: Resolved

9. Potential reconciliation failure due to uncapped fee amount

Severity: Major

In `contracts/strategy/neutron-pos/reconcile/state_machine.rs:411`, the `split_rewards` function computes the remaining rewards by deducting the withdrawn rewards and computed fee. This is problematic because the fee can grow arbitrarily (and exceed the `HUNDRED_PERCENT_BPS` value) based on the passed block height number, as seen in lines `358-360`.

Consequently, an overflow error will occur, breaking the reconciliation process.

Recommendation

We recommend implementing a max fee basis point configuration and enforcing a maximum limit of incentivization fees for the reconciler.

Status: Resolved

10. Rewards are not increased for zero fees or empty fee recipient

Severity: Major

In `contracts/strategy/neutron-pos/reconcile/state_machine.rs:559`, the `after_transfer_tx` function does not add the `IncreaseReceivedRewards` command when the fee BPS is zero or the fee recipient is not set. This is problematic because the delegation rewards will not be accounted for when redelegated with the `DelegateInflightDeposit` command.

Consequently, the total deposits will not record the delegation rewards when the `Sync` message is called, preventing the funds from being unbonded.

Recommendation

We recommend adding the `IncreaseReceivedRewards` command similar to line `568`.

Status: Resolved

11. Delegation rewards computed will be zero if `MsgUndelegate` is called

Severity: Major

In `contracts/strategy/neutron-pos/reconcile.rs:523`, the `get_withdrawal_success_ctx` function computes the withdrawn rewards by parsing the `MsgWithdrawDelegatorReward` response after the idle reconciliation phase for `StrategyState::ActiveDeposits` succeeds. Before dispatching the message with the `WithdrawRewards` command in `contracts/strategy/neutron-pos/reconcile/state_machine.rs:400`, the `UndelegateInflightUnbond` command is called first in line `397` if there are pending delegations to unbond. In other words, `MsgUndelegate` is called before `MsgWithdrawDelegatorReward`.

This is problematic because when `MsgUndelegate` is called, rewards are automatically withdrawn. When the `MsgWithdrawDelegatorReward` is called after that, zero rewards will be accrued, causing the reconciliation process to incorrectly determine the withdrawn rewards as zero.

Consequently, the withdrawn rewards will not be redelegated but will eventually be distributed back to the contract with the `TransferInflightUnbonded` command.

This issue also affects the `MsgDelegate` message and subsequent `DelegateInflightDeposit` calls.

Recommendation

We recommend calling `MsgWithdrawDelegatorReward` before `MsgDelegate` and `MsgUndelegate` messages to compute the delegation rewards correctly.

Status: Partially Resolved

This issue is partially resolved because the `WithdrawRewards` command is not called before `MsgDelegate`.

12. Incorrect claim amount calculation

Severity: Major

In `contracts/strategy/neutron-pos/execute.rs:241-244`, the `calculate_actual_claim_amount` function computes the claim amount by multiplying the requested amount with the expected unbonding amount and dividing it by the actual unbonding amount. This is incorrect because the result will be larger if the expected unbonding amount is not fulfilled, which allows claiming a higher amount than intended.

Recommendation

We recommend switching the numerator and denominator in the proposed formula.

Status: Resolved

13. Incorrect ICQ deposit amount is provided

Severity: Minor

In `contracts/strategy/neutron-pos/contract.rs:137`, the required deposit amount is computed based on the queried deposit fee multiplied by the `ICQS_PER_SLOT`. This is required because after the interchain account is created with the `RegisterInterchainAccount` message in line `210`, the `new_register_balance_query_msg` and `new_register_delegator_delegations_query_msg` functions will be called to register interchain queries, as seen in `contracts/strategy/neutron-pos/sudo.rs:64-76`. Since registering an interchain query requires a fee to be paid, it is important to ensure that the fees are provided correctly.

A problem occurs when there is more than one interchain account that gets registered with the `RegisterInterchainAccount` message. In this case, the total ICQ fees should be multiplied according to the initial validator set (see `contracts/strategy/neutron-pos/contract.rs:133`). However, this is not enforced during the contract instantiation phase.

Consequently, instantiating the contract with more than one validator will fail, preventing the contract from being instantiated successfully.

We classify this issue as minor because the contract instantiator can recover by modifying and uploading a new code ID to instantiate the contract.

Recommendation

We recommend computing the required ICQ deposit fees according to the validator set.

Status: Resolved

14. Actual unbonding accounting can be anonymously inflated

Severity: Minor

The `handle_receive_ubonded` function of the `strategy` contract does not perform any kind of access control, and it allows for funds to be sent along, directly increasing `total_actual_received`. Tampering with this value directly affects the result of `calculate_actual_claim_amount` in `contracts/strategy/neutron-pos/execute.rs:241-244`.

Given the incorrect order of the formula, as detailed in the Incorrect claim amount calculation issue, an attacker that adds large amounts of funds would affect the functioning of vaults as a smaller claim would be received each time. However, it should be noted that even with the correct formula in place, the behavior of the Vault can be affected by the difference between expected and effective claims.

Recommendation

We recommend implementing access controls in the affected function so that only actual validators can submit funds. In addition, we recommend raising an error if `expected_unbondings` is zero.

Status: Resolved

15. Lack of input validation

Severity: Minor

The contracts within the scope of this audit do not perform input validation before saving data in the storage on the `set_primary_deposit_asset`, `instantiate`, `parse`, and `set_config` functions.

- The PDA's ID is not checked to be a valid denom in `crates/core/strategy.rs:286-300`.
- The slot weight for validators is not checked to be larger than zero.
- Timeouts should not accept zero as a value to properly communicate with a remote chain in `contracts/strategy/neutron-pos/contract.rs:153`, `158`, `163` and in `contracts/strategy/neutron-pos/execute.rs:276`, `280`, `288`, and `292`.

Recommendation

We recommend querying the total supply of the submitted denom to ensure it is a valid denom and applying the validations above.

Status: Partially Resolved

This issue is partially resolved because the slot weight for validators is not validated to be larger than zero.

16. State transition can be optimized

Severity: Informational

The state transitions of the `strategy` contract use `unwrap_or_default` when loading the relevant piece of data for each of the states. However, if there is no stored data, a default of zero will end up not taking any effect but costing gas, for example, passing a `pending_deposit` of zero in the `StartReconcileProcessCtx`.

The same happens for most state transition functions in `contracts/strategy/neutron-pos/reconcile.rs:243`, `256`, `266`, `276`, `287`, `557`, `572`, `596`, and `615`.

Recommendation

We recommend raising an error instead of loading the default value on the state transitions.

Status: Acknowledged

17. Missing validation during `SetNextAdmin`

Severity: Informational

The `SetNextAdmin` message in `crates/shell/admin.rs:54` is used to change the current admin to a new admin. However, during its execution, there is no validation that the proposed `next_admin` differs from the `current_admin`, which should be the sender itself.

While this issue does not pose a direct risk to the protocol, it performs unnecessary operations while the same admin is proposed again.

Recommendation

We recommend checking if `next_admin != current_admin` at the beginning of the `set_next` function or earlier at the `parse` function level.

Status: Acknowledged

18. Incorrect error returned during vault authorization

Severity: Informational

The `vault` function in `crates/core/auth.rs:205`, requires the caller to hold the `vault` role for authentication. However, if the `vault` value, being an optional parameter is not provided, it leads to a situation where the `ok_or` method returns `None` in line `208`, and an `UnauthorizedError` is returned. This situation is misleading as it does not provide the caller with a clean and specific error message because the reason is the lack of a `vault` parameter in the function call, not an issue with authorization itself.

Recommendation

We recommend implementing distinct error messages for the case described above.

Status: Acknowledged

19. Remove debugging messages

Severity: Informational

The codebase has a number of debugging checks, performed through the macros such as `debug`, `dbg`, `debug_assert_eq`, and `debug_assert`. It is best practice to remove these debugging messages before deploying the code in production.

Recommendation

We recommend removing the following debug statements:

- `crates/core/strategy.rs:533`

- `crates/cosmwasm/lib.rs:27`
- `contracts/strategy/neutron-pos/sudo.rs:37`
- `contracts/strategy/neutron-pos/execute.rs:55` and `175`
- `contracts/strategy/neutron-pos/query.rs:20`
- `contracts/strategy/neutron-pos/state.rs:145` and `203`

Status: Acknowledged

20. Fix spelling errors

Severity: Informational

In `contracts/strategy/neutron-pos/execute.rs:449`, the function name is misspelled as `handle_receive_ubonded`.

Additionally, in the `take_detected_losses` function, the variable responsible for holding the `losses` amount is called `rewards`.

Recommendation

We recommend updating the spelling errors as mentioned above.

Status: Resolved