Audit Report

Neutron

v0.3

December 7, 2022

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 P2P Staking to perform a security audit of Neutron.

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 repositories:

https://github.com/neutron-org/neutron (referred to as `neutron` below)

Commit hash: `7e9751768e533ea9908bbe6fcc6d490747bf0cca`

https://github.com/neutron-org/neutron-contracts (referred to as `neutron-contracts` below)

Commit hash: `e9b2a9d9179d3c136ba024870e762bcb046b0f8c`

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

Neutron is a blockchain network that brings Smart Contracts into the Cosmos ecosystem using CosmWasm. Neutron uses the IBC protocol leveraging Interchain Accounts in order to perform custom cross-chain queries and transactions. Its security is provided by the Cosmos Hub network using Interchain Security.

The audit scope comprehends the Cosmos SDK Neutron chain and its CosmWasm bindings and example 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	Proposal's tally returns wrong vote results	Critical	Resolved
2	An error triggered during the handling of an `Ack` `IBC` message will make the channel unusable and spam the network	Critical	Resolved
3	`IBC` events loop in `Sudo` handler could drain relayer's funds	Critical	Resolved
4	Attackers are able to spam the network with `IBC` messages using the `ibc-transfer` module	Major	Resolved
5	Unbounded iteration in `ValidateBasic` may cause node timeout	Major	Resolved
6	Unbounded iteration in `PerformSubmitTx` could be used by an attacker to slow down or halt the chain	Major	Resolved
7	Unbounded iteration in `EndBlocker` when calculating vote power could be used by an attacker to slow down or halt the chain	Major	Resolved
8	Attackers could steal funds from the `ibc-transfer` contract	Major	Acknowledged
9	`Fee` struct could be simplified to avoid manipulations	Major	Resolved
10	Unbounded messages loop could lead the execution out of gas	Minor	Resolved
11	Contracts are not compliant with `CW2` migration specification	Minor	Resolved
12	Relayers overriding `ACKNOWLEDGEMENT_RESULTS` could lead to inconsistency	Minor	Resolved
13	DAO contract upgrades require a chain upgrade	Minor	Resolved
14	Incomplete `MsgRegisterInterchainQuery` message validation	Minor	Resolved

15	`Sudo` handler is not handling multiple transactions correctly under some circumstances	Minor	Acknowledged
16	`UpdateInterchainQuery` message does not support transaction queries	Minor	Resolved
17	Lack of validation for the `GenesisState` of the `feerefunder` module	Minor	Resolved
18	`ContractFailure` not being removed could cause RPC nodes to be spammed	Minor	Resolved
19	`AppModule` contains an unused `sudoHandler` attribute	Informational	Resolved
20	`KVKeysFromString` function is unused	Informational	Resolved
21	Additional documentation required for `SudoMsg` handlers in the `ibc-transfer` contract	Informational	Resolved
22	Inconsistent naming of query function may cause confusion	Informational	Resolved
23	Missing event emission may negatively impact off-chain components	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium	-
Level of documentation	Medium	The documentation has high variance of quality and completeness between different modules.
Test coverage	Low	17.4% test coverage for Neutron Cosmos SDK appchain. 53.17% test coverage for Neutron CosmWasm contracts. No unit tests are in place.

Detailed Findings

1. Proposal's tally returns wrong vote results

Severity: Critical

In `neutron:x/gov/keeper/tally.go:79-81`, when calculating the result of a governance proposal, the execution is not accounting for `NoWithVeto` votes.

The current implementation returns a failed proposal result if the following inequation is true:

$$\frac{OptionNo}{totalTokensVoted - OptionAbstain} \geq tallyParams.Threshold$$

It is evident that `NoWithVeto` votes are not counted and treated as `Yes` votes in the example in Appendix 1.

This implies that proposals that should fail will be unintendedly successful.

Recommendation

We recommend adding `NoWithVeto` votes to `No` votes in the previous inequation as follows:

$$\frac{OptionNo + OptionNoWithVeto}{totalTokensVoted - OptionAbstain} \geq tallyParams.Threshold$$

Status: Resolved

The Neutron team decided to remove the custom `gov` module and restore the default one from Cosmos SDK.

2. An error triggered during the handling of an `Ack` IBC message will make the channel unusable and spam the network

Severity: Critical

Neutron uses `ORDERED` channels, which means that there is a sequencer that keeps track of the currently waiting `Ack` message.

This message is handled in the `Acknowledgement` function defined in https://github.com/cosmos/ibc-go/blob/77c10be63204a52ec53b1e8ef91a76bae140d5ed/modules/core/keeper/msg_server.go#L588-L647 that is responsible for incrementing the `NextSequenceAck` and call the `IBCModule`'s `HandleAcknowledgement` method defined in `neutron:x/interchaintxs/keeper/ibc_handlers.go:16-47`.

If it returns an error, the execution will revert and `NextSequenceAck` will not be incremented. This could happen for various reasons:

- `Sudo` handlers in the smart contract return an error because of a bug or an invalid input data
- `Sudo` handlers are not defined in the smart contract

This implies that the channel will be unusable and that the relayer will continue sending the same failing `Ack` message.

Also, an attacker could use a significant amount of deployed smart contracts and transactions with failing `Sudo` handlers in order to let relayers spam the network with `Ack` messages and trigger the `BroadcastTxCommit` timeout and the `NewTxTimeoutHeightDecorator` defined in the `AnteHandler`.

This can prevent that node from processing further `ABCI` messages such that it has to pause and contact peers to get the latest correct blocks. If a significant number of them hit the timeout and halt simultaneously, block production may slow down or even stop.

Recommendation

We recommend implementing logic to manage failures in the `HandleAcknowledgement` in order to stop the relayer from continuously re-propose the same failing `Ack` message.

Status: Resolved

3. `IBC` events loop in `Sudo` handler could drain relayer's funds

Severity: Critical

A malicious hacker could develop a CosmWasm smart contract that implements an `IBC` events loop in the `Sudo` handler.

For example a contract that in the `Response Sudo` handler, executes another transaction that will trigger the same `Response` handler and so on.

Since there is not an aggregate gas counter for all the different transactions of the `IBC` events flow that could break the loop with an out of gas error, the execution can run until all the relayer's funds are drained.

Also, this behavior could be used to congest and slow down the chain.

Recommendation

We recommend implementing an aggregate gas counter for each `IBC` flow in order to break possible loops.

Status: Resolved

4. Attackers are able to spam the network with IBC messages using the ibc-transfer module

Severity: Major

In

- neutron:internal/sudo/sudo.go:100-103,
- neutron:internal/sudo/sudo.go:132-135, and
- neutron:internal/sudo/sudo.go:166-169,

the execution is checking that the received Acknowledgement or Timeout Packet is related to an IBC transaction originated from an existing CosmWasm smart contract address and returning an error otherwise.

Since it is possible to send an ibc-transfer module's transfer transaction also from a non contract address, for example using the neutrond CLI, with the following command:

```
neutrond tx ibc-transfer transfer
```

an attacker could send a big number of small value transfer messages in order to spam the network with Acknowledgement packets and let them fail in the guard implemented in the mentioned lines.

A huge number of such messages could congest nodes and make them unable to process blocks before the BroadcastTxCommit timeout.

Also, the NewTxTimeoutHeightDecorator will discard all messages with an elapsed heightTimeout, which could be used in a particular event in order to manipulate it.

This can prevent that node from processing further ABCI messages such that it has to pause and contact peers to get the latest correct blocks. If a significant number of them hit the timeout and halt simultaneously, block production may slow down or even stop.

Recommendation

We recommend removing the guard or restricting transfer transactions in the ibc-transfer module to only smart contracts.

Status: Resolved

5. Unbounded iteration in `ValidateBasic` may cause node timeout

Severity: Major

The `ValidateBasic` function for `MsgRegisterInterchainQuery` in `x/interchainqueries/types/tx.go:73-100` includes an unbounded iteration that may be exploited to cause a node timeout.

The function loops over `TransactionsFilter`, a caller-supplied slice that is not checked for duplicate entries and does not have a defined size upper-bound.

It is best practice to keep `ValidateBasic` logic simple as gas is not charged when it is executed. It should only perform all necessary stateless checks to enable middleware operations (for example, parsing the required signer accounts to validate a signature by a middleware) without impacting performance in the `CheckTx` phase. Other validation operations must be performed when handling a message in a module's `MsgServer`.

Recommendation

We recommend simplifying the checks that are performed during the `ValidateBasic` function. A possible solution could be implementing a length upper-bound for `TransactionsFilter`.

Status: Resolved

6. Unbounded iteration in `PerformSubmitTx` could be used by an attacker to slow down or halt the chain

Severity: Major

In `neutron:wasmbinding/message_plugin.go:204`, `SubmitTx` is performing an unbounded iteration over `submitTx.Msgs`.

An attacker could craft a message with a significant number of `Msgs` with the intention of spamming the network and impact the block production time triggering the `BroadcastTxCommit` timeout and the `NewTxTimeoutHeightDecorator` defined in the `AnteHandler`.

This can prevent the node from processing further `ABCI` messages such that it has to pause and contact peers to get the latest correct blocks. If a significant number of them hit the timeout and halt simultaneously, block production may slow down or even stop.

Recommendation

We recommend placing a cap on the number of messages that the user can send in `SubmitTx`, or otherwise consume gas in each iteration.

Status: Resolved

7. Unbounded iteration in `EndBlocker` when calculating vote power could be used by an attacker to slow down or halt the chain

Severity: Major

In `neutron:x/gov/keeper/voting.go:40-51`, in order to calculate voting powers, the `EndBlocker` is running an unbounded iteration over a slice containing all voting user votes.

An attacker could use a significant number of addresses with a small amount of tokens in order to grow the slice length and impact the block production time, eventually triggering the `BroadcastTxCommit` timeout and the `NewTxTimeoutHeightDecorator` defined in the `AnteHandler`.

This can prevent the node from processing further `ABCI` messages such that it has to pause and contact peers to get the latest correct blocks. If a significant number of them hit the timeout and halt simultaneously, block production may slow down or even stop.

Recommendation

We recommend not performing unbounded iterations in `BeginBlocker` or in the `EndBlocker`. Instead, logic could be implemented in the DAO contract in order to store aggregated vote data.

Status: Resolved

The Neutron team decided to remove the custom `gov` module and restore the default one from Cosmos SDK.

8. Attackers could steal funds from the `ibc-transfer` contract

Severity: Major

In `neutron-contracts:contracts/ibc_transfer/src/contract.rs:43`, `execute_send` is called in order to complete an `IBC-20` transaction from source chain to sink chain, but this function does not verify that there are sufficient funds in the contract to support an `IBC` transfer.

Because the `execute_send` function lacks an authorization check, an attacker can frontrun the legit user and steal funds by sending them from the contract to its own address on the sink chain.

We classify this issue as major instead of critical since the `ibc-transfer` contract is meant to be an example.

Recommendation

We recommend either having the `msg.sender` send money during the `execute_send` function call or implementing an authorization check in order to confirm the ownership of the funds held within the contract.

Status: Acknowledged

The Neutron team states that the `ibc-transfer` contract is meant to be used only as an example and for testing purposes. Warnings about the involved risks of missing authorization have been added in code and documentation.

9. `Fee` struct could be simplified to avoid manipulations

Severity: Major

In `neutron:proto/feerefunder/fee.proto:17`, `Fees` definition includes `Coins` as an attribute in order to track users' and relayers' payments. The `checkFees` function defined in `neutron:x/feerefunder/keeper/keeper.go:169` ensures that the user pays the right amount, checking that coins are over a threshold. However, there's no validation for the denom. This can be abused, and only one coin with almost no value could be used to pass the validation checks.

Recommendation

We recommend either specifying and enforcing the minimum fee per denom. Alternatively, fees could be limited to one coin by replacing `Coins` with `Coin`, as this replacement is still compliant with the `Fee` interface, see https://github.com/cosmos/ibc/tree/main/spec/app/ics-029-fee-payment#fee-middleware-contract.

Status: Resolved

10. Unbounded messages loop could run out of gas

Severity: Minor

In `neutron-contracts:contracts/neutron_interchain_queries/src/contrac`

`t.rs:312`, the `recipient_deposits_from_tx_body` function iterates over the `tx_body.messages` to filter out the required transaction.

However, because the number of messages is unknown, this could lead to an unbounded loop execution, which could cause an out-of-gas error.

This implies that the smart contract may never be able to consume the supplied dataset corresponding to the query.

Even though this is a major issue, we classify it as minor since the affected contract is only an example contract.

Recommendation

We recommend limiting the amount of messages that are executed and, if there are more messages than the limit permits, emitting the unprocessed messages so the relayer can re-submit them.

Status: Acknowledged

The Neutron team states that the mentioned contract is meant to be used only as an example and for testing purposes. In production, contract developers have to take responsibility for handling discarded messages.

11. Contracts are not compliant with `CW2` migration specification

Severity: Minor

The following contracts do not adhere to the `CW2` migration specification:

- `neutron-contracts:contracts/reflect`,
- `neutron-contracts:contracts/neutron_interchain_txs`,
- `neutron-contracts:contracts/neutron_interchain_queries`, and
- `neutron-contracts:contracts/ibc-transfer`.

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

Recommendation

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

Status: Resolved

12. Relayers overriding `ACKNOWLEDGEMENT_RESULTS` could lead to inconsistency

Severity: Minor

In

- `neutron-contracts:contracts/neutron_interchain_txs/src/contract.rs:384`,
- `neutron-contracts:contracts/neutron_interchain_txs/src/contract.rs:405`, and
- `neutron-contracts:contracts/neutron_interchain_txs/src/contract.rs:425`,

if the relayer submits the data for the same `seq_id` and `channel_id` again to the chain, this may lead to an inconsistent result since the `ACKNOWLEDGEMENT_RESULTS` might be overridden.

We classify this issue as minor since the affected contract is only an example contract.

Recommendation

We recommend deleting the `Sudo` message payload that corresponds to the actual `seq_id` and `channel_id` tuple as soon as the submitted data has been processed on-chain.

Status: Resolved

13. DAO contract upgrades require a chain upgrade

Severity: Minor

In `neutron:x/gov/keeper/tally.go:13`, the DAO contract's `code_id` is hardcoded.

Consequently, if this contract needs to be updated to fix an issue or to introduce a feature, it will not be possible without updating the chain binary and distribuite the new executable to validators.

Recommendation

We recommend having the `code_id` of the DAO contract in the module's `Params` in order to enable the governance to update it.

Status: Resolved

The Neutron team decided to remove the custom `gov` module and restore the default one from Cosmos SDK.

14. Incomplete `MsgRegisterInterchainQuery` message validation

Severity: Minor

In `neutron:x/interchainqueries/types/tx.go:73-100`, during the validation of the `MsgRegisterInterchainQuery` message in the `ValidateBasic` function, there aren't any checks for the `keys` field if `QueryType` is `KV`.

This implies that an invalid `MsgRegisterInterchainQuery` can successfully pass the `ValidateBasic` checks and be processed in the message handler.

Recommendation

We recommend implementing a check in `MsgRegisterInterchainQuery`'s `ValidateBasic` method in order to ensure that it is correctly constructed.

Status: Resolved

15. `Sudo` handler is not handling multiple transactions correctly under some circumstances

Severity: Minor

In `neutron-contracts:contracts/ibc_transfer/src/contract.rs:131`, a query searches for transfer transactions with a recipient equal to the provided one and a `min_height` larger than a specific value.

In a realistic scenario, there might be several transfer transactions where the receiver matches the given recipient under the same `min_height`.

In fact, when the sudo handler submits the result of the `tx_query`, it always expects it will be one, which is not necessarily the case. This assumption could cause the handler to improperly decode the data parameter and lose the transaction data, which would produce inaccurate results.

We classify this issue as minor since the affected contract is only an example contract.

Recommendation

We recommend not relying on the mentioned assumption and using a multi transaction decoding method instead.

Status: Acknowledged

16. `UpdateInterchainQuery` message does not support transaction queries

Severity: Minor

In lines `neutron:x/interchainqueries/keeper/msg_server.go:118-151`, when handling an `UpdateInterchainQuery` message, there is no check in order to perform a different behavior for different query types.

The function assumes that the given query is of the `KV` type one so it will be possible to save `KV` query parameters like `keys` in transaction queries.

Consequently, it will not be possible to update `transactions_filter` of transaction queries.

Recommendation

We recommend implementing a condition in order to support the update of both types of queries.

Status: Resolved

17. Lack of validation for the `GenesisState` of the `feerefunder` module

Severity: Minor

In `neutron:x/feerefunder/types/genesis.go:20`, the `Validate` function is not validating the `FeeInfos` and `Params` parameters. Consequently, `Payer` could be an incorrect address and the `PacketId` and `Fee` structs may not be well-constructed.

Also, the `Validate` function for `Params` in `neutron:x/feerefunder/types/params.go:43-45` is not implemented and always returns `nil`.

Recommendation

We recommend implementing validation for the `GenesisState` and its `Params` attribute of the `feerefunder` module.

Status: Resolved

18. `ContractFailure` not being removed from the store could cause RPC nodes to be spammed

Severity: Minor

In `neutron:x/contractmanager/keeper/failure.go`, the `AddContractFailure` function adds a failure entry to the store. However, once consumed by the client, there is no way to remove those entries.

Failures can be queried through the `Failures` query defined in `neutron:x/contractmanager/keeper/grpc_query_failure.go:14`. This method allows the caller to specify a pagination parameter.

Nevertheless, since there is no cap on the pagination parameter, this can be maliciously defined to be close to the highest number possible with the following combination

$$offset = 0 \ \wedge \ limit = uint64::Max$$

This combination of parameters will cause `query.Paginate` to load many results, the majority of them potentially already consumed. Since anybody can call this endpoint, it could be an attack vector over RPCs to spam nodes.

Recommendation

We recommend capping the number of results that can be requested from this endpoint by enforcing a maximum number for the `limit` parameter. Alternatively, failures that have already been consumed could be flagged and removed from the store through a time dependent trigger.

Status: Resolved

19. `AppModule` contains an unused `sudoHandler` attribute

Severity: Informational

In `neutron:x/interchainqueries/module.go:106`, the `AppModule` struct defines a `sudoHandler` attribute that is never used.

Recommendation

We recommend removing the unused attribute.

Status: Resolved

20. `KVKeysFromString` function is unused

Severity: Informational

In `neutron:x/interchainqueries/types/types.go:96`, the function `KVKeysFromString` is defined but never used in the code.

Recommendation

We recommend removing the mentioned unused function.

Status: Resolved

21. Additional documentation required for `SudoMsg` handlers in the `ibc-transfer` contract

Severity: Informational

In `neutron-contracts:contracts/ibc_transfer/src/contract.rs:165`, `match` is not implemented for all the `SudoMsg` variants and a `todo!` macro is used instead.

This will lead to the issue described in "An error triggered during the handling of an Ack IBC message will make the channel unusable and spam the network"

Since this is a contract that should be used as an example and reference for other devs, it should follow all the best practices defined in the Neutron documentation.

Recommendation

We recommend implementing a `match` statement for all `SudoMsg`s and providing more comments and documentation.

Status: Resolved

22. Inconsistent naming of query function may cause confusion

Severity: Informational

In `neutron-contracts:contracts/neutron_interchain_queries/src/contract.rs:206`, according to its literal meaning, the `query_transfers_amount` function should return the total amount of tokens that have been moved, but instead it returns the total number of transfers that have been made so far.

Recommendation

We recommend changing the `query_transfers_amount` function's logic to reflect its name literal meaning or renaming the function to reflect how it is meant to operate.

Status: Resolved

23. Missing event emission may negatively impact off-chain components

Severity: Informational

In `neutron:x/feerefunder/keeper/keeper.go:121`, no event is emitted when the timeout fee is distributed, whereas an event gets emitted when the acknowledgement fee is distributed.

This may negatively impact off-chain components such as data collectors and indexers.

Recommendation

We recommend emitting an event during the timeout fee distribution.

Status: Resolved

Appendix

1. Example data demonstration for issue “Proposal tally is not accounting no with veto votes”

Example proposal’s vote result:

- `OptionYes`: 40
- `OptionNo`: 20
- `OptionNoWithVeto`: 25
- `OptionAbstain`: 15

With those votes, the proposal should fail so the following inequation should be verified:

$$\frac{OptionNo}{totalTokensVoted - OptionAbstain} \geq tallyParams.Threshold$$

Substituting with example data:

$$\frac{20}{100 - 15} \geq 0.5 \Rightarrow 0.235 \geq 0.5$$

The inequation is not verified and the proposal is marked as successful.