Audit Report

Sei Chain and CosmWasm

v1.0

May 19, 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 Sei Labs Inc to perform a security audit of Sei Chain and CosmWasm.

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://github.com/sei-protocol/sei-chain

Commit hash: `8cde96a2924d1e529a99c47693884e07d86e26d1`

https://github.com/sei-protocol/sei-cosmwasm

Commit hash: `76fa3638fdc62750335d04b1ad3c899a4189d37c`

The audit was performed only on the `packages/sei-cosmwasm` directory.

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

Sei chain is a blockchain using Cosmos SDK and Tendermint Core. One of the core modules of the Sei chain is the dex module – it provides a built-in central limit order book (CLOB). Decentralized applications building on Sei can build on top of the CLOB by deploying CosmWasm smart contracts. Other Cosmos-based blockchains can leverage Sei's CLOB as a shared liquidity hub and create markets for any asset.

Sei CosmWasm aims to deliver packages to support smart contract querying and messages to the modules of the Sei chain. It also includes an example contract that can be used to test package behavior locally and may be used as a reference for implementation details on Sei chain integration.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states, or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium-High	Whilst the recommended Cosmos SDK design patterns are followed in most cases, the codebase is large and complex as it has a unique design with significant specialized functionalities and optimization, such as optimistic block processing and parallel execution.
Code readability and clarity	Medium-High	-
Level of documentation	Medium	Technical documentation could have been more comprehensive
Test coverage	Medium	Go test reports a 60.6% code coverage excluding simulation packages and automatically generated files.

Summary of Findings

No	Description	Severity	Status
1	Unbounded loops in the `nitro` package can lead to a DoS	Critical	Resolved
2	Not registered message types prevent users from sending transactions	Major	Resolved
3	`MsgContractDepositRent` should set a minimum amount to avoid potential spamming	Major	Resolved
4	Tick sizes are wrongly updated when a newly registered pair already exists	Major	Resolved
5	Processing of `MsgPlaceOrders` and `MsgSend` messages in `EndBlocker` may be exploited to halt the chain if mispriced	Major	Resolved
6	Contract claiming is not possible due to a logic error	Major	Resolved
7	Past minting emissions would never happen if the chain is down on the required release date	Minor	Resolved
8	`Amino` codec must be registered to support end users with hardware devices like Ledger	Minor	Resolved
9	Insufficient validation of addresses	Minor	Resolved
10	`GetContract` error type used in an incorrect way in `HandleDepositOrRefund`	Minor	Resolved
11	Fund transfer in `EndBlocker` goes against best practices	Minor	Resolved
12	Events are not emitted consistently across the application, which may impact the ability of third-party applications to be integrated	Minor	Resolved
13	`ChangeAdmin` does not validate that the new admin is different to the current one, which may cause errors in contracts and applications relying on this logic	Minor	Resolved
14	Inflation rate is set to 0%, while the documentation indicates an intended default inflation rate of 13%	Minor	Resolved

15	`UnregisterContract` should remove associated pairs after the contract is removed to avoid consuming unnecessary disk space on validators	Minor	Resolved
16	Inefficient implementation of UpdateQuantityTickSize and UpdatePriceTickSize	Informational	Resolved
17	Strict parsing of boolean CLI arguments may degrade user experience	Informational	Resolved
18	Inefficient implementation of `CancelOrders`	Informational	Resolved
19	Inefficient implementation of `PlaceOrders`	Informational	Resolved
20	Redundant checks are inefficient	Informational	Resolved
21	Uninformative error message	Informational	Resolved
22	Misleading short description in `ContractDepositRent` command	Informational	Resolved
23	Use of deprecated module and functions	Informational	Acknowledged
24	Prefer `map[T]struct{}{}` over `map[T]bool` for performance reasons	Informational	Resolved
25	Unchecked error in `SubmitFraudChallenge` function might lead to inefficiencies	Informational	Resolved
26	Missing usage description for transaction and query CLI commands	Informational	Resolved
27	Unnecessary use of signed integers is inconsistent and may be prone to programming errors in the future	Informational	Resolved
28	Resolve TODOs before release	Informational	Resolved
29	Miscellaneous comments	Informational	Resolved

Detailed Findings

1. Unbounded loops in the `nitro` package can lead to a DoS

Severity: Critical

In `x/nitro/keeper/msg_server.go:75` and `x/nitro/keeper/prover.go:24`, the user-provided fields `msg.EndSlot` and `proof.Hash` may lead to an unbounded loop or an arbitrarily computationally expensive transaction which can lead to a DoS. This is problematic since the `Validate` function in `x/nitro/keeper/prover.go:24` does not charge gas to the sender of the transaction, making the attack free of charge. It is important to note that both these code paths are called by the `SubmitFraudChallenge` method in `x/nitro/keeper/msg_server.go:56`, which is currently not registered on the `Router` in `x/nitro/module.go:107`.

Recommendation

We recommend ensuring that the `proof.Hash` length is capped. Additionally, we recommend ensuring that the difference between `msg.EndSlot` and `msg.StartSlot` is limited.

Status: Resolved

2. Not registered message types prevent users from sending transactions

Severity: Major

`MsgContractDepositRent` and `MsgUnregisterContract` are not registered in `RegisterInterfaces` and `RegisterCodec` in `x/dex/types/codec.go`. Hence, the module will not route the messages to an appropriate message server. Thus, users will not be able to send transactions with the aforementioned message types.

Recommendation

We recommend registering `MsgContractDepositRent` and `MsgUnregisterContract` accordingly.

Status: Resolved

3. `MsgContractDepositRent` should set a minimum amount to avoid potential spamming

Severity: Major

In `x/dex/keeper/msgserver/msg_server_contract_deposit_rent.go:12`, there is no minimum limit to the amount that can be sent, and there is no gas charged for the operation, by default. This can be a cost-effective attack vector for the protocol, in which multiple transactions are sent with negligible amounts to consume CPU cycles with minimal cost. This may be used to slow down block production or even halt the chain.

Recommendation

We recommend running benchmarks to determine a sensible minimum deposit amount and setting it in `ContractDepositRent`.

Status: Resolved

4. Tick sizes are wrongly updated when a newly registered pair already exists

Severity: Major

In `x/dex/keeper/msgserver/msg_server_register_pairs.go:36`, any errors returned from the `k.AddRegisteredPair` function are silently ignored. This implies that the execution will continue even if the creator of a contract is trying to register a pair that already exists.

This unhandled error has the unintended consequence that `SetPriceTickSizeForPair` and `SetQuantityTickSizeForPair` are called in lines `37-38`, updating the current tick sizes of an already existing pair. Since events for that pair are not emitted, this could break any integrations with off-chain components and third-party applications relying on those events.

Recommendation

We recommend handling the case by returning an error and halting the execution of the message, or alternatively, continuing with the next iteration of the loop.

Status: Resolved

5. Processing of `MsgPlaceOrders` and `MsgSend` messages in `EndBlocker` may be exploited to halt the chain if mispriced

Severity: Major

Due to the processing of `MsgPlaceOrders` and `MsgSend` messages in the `EndBlocker`, a user can send transactions with multiple `MsgPlaceOrders` and `MsgSend` messages that are successfully included in a block, but then fail during the `EndBlocker` execution. An attacker may exploit this feature if the computation needed in the `EndBlocker` is not properly priced in the transaction. If a network has zero or low fees, an attacker could send hundreds of thousands of `MsgPlaceOrders` and `MsgSend` messages at comparatively low cost for the computation required which can slow down block production up to the point where the blockchain halts.

Recommendation

We recommend consuming additional gas in the handlers for `MsgPlaceOrders` message for the computation performed in the `EndBlocker`.

Status: Resolved

6. Contract claiming is not possible due to a logic error

Severity: Major

In the `ContractDepositRent` function, a new owner can claim a contract if `contract.RentBalance == 0`. However, in `x/dex/keeper/msgserver/msg_server_contract_deposit_rent.go:28`, the `SendCoins` operation is executed from the old `creatorAddr`, not from `msg.Sender`. Thus, this function will always fail when called by any address which is not the current owner, rendering the claim functionality unusable.

Recommendation

We recommend substituting `creatorAddr` with `msg.Sender` in line `28`, to execute the transfer of funds from the new potential owner and not the current contract creator.

Status: Resolved

7. Past minting emissions would never happen if the chain is down on the required release date

Severity: Minor

In `x/mint/types/minter.go:56`, the `GetScheduledTokenRelease` function validates if the date of the block is equal to the `scheduledReleaseDate`, and gets

executed in the `BeginBlock` of the `epoch` module. If the date matches, new mint emissions of the chain token will be performed. However, the function does not consider if the date `scheduledReleaseDate` is in the past, but the emission of the given epoch has not happened yet. Therefore, in case the Sei chain is down for more than 24 hours, the release of tokens is skipped, altering the emission plan.

Recommendation

We recommend accounting for past planned release dates that have not actually occurred, to cover the scenario where the chain is down for a prolonged period of time.

Status: Resolved

8. `Amino` codec must be registered to support end users with hardware devices like Ledger

Severity: Minor

In `x/dex/types/codec.go:50`, `Amino` should be used to register all interfaces and concrete types for the `dex` module. This is necessary for JSON serialization to support hardware devices like Ledger since these devices do not support proto transaction signing.

Recommendation

We recommend registering all interfaces and concrete types for the `dex` module using `Amino` to support JSON serialization.

Status: Resolved

9. Insufficient validation of addresses

Severity: Minor

The address of the creator, as well as the address of the contract in `validateCancels` the function in `x/dex/keeper/msgserver/msg_server_cancel_orders.go` are only checked against zero-length and not for validity of the address.

Recommendation

We recommend using the `AccAddressFromBech32` function to ensure the string forms a valid address.

Status: Resolved

10. `GetContract` parse error not handled explicitly

Severity: Minor

In `x/dex/keeper/contract.go:37`, the method `GetContract` returns an error either when the contract is not present in the storage, or when the contract is found but cannot be parsed properly. However, the calling contexts in `x/dex/keeper/msgserver/msg_server_register_contract.go:95`, `107`, and `193` assume that the contract does not exist upon either error and does not handle parsing errors.

Recommendation

We recommend checking that the error variant signals that the contract does not exist and handling parse errors by returning them.

Status: Resolved

11. Fund transfer in `EndBlocker` goes against best practices

Severity: Minor

When a user sends a transaction using the `MsgPlaceOrders` message, the `transferFunds` function in `x/dex/keeper/msgserver/msg_server_place_orders.go:15` is called to create a `DepositInfoEntry` with the user's fund amount in `MemState`. The amount is not actually moved from the sender's account until the `GetDepositSudoMsg` function is called in the `EndBlock` function in `x/dex/keeper/abci/end_block_deposit.go:33`. If a user sends a transaction with multiple messages of `MsgPlaceOrders` and `MsgSend`, the transaction would be successfully included in a block but the deposit might fail during the `EndBlocker` execution since insufficient funds may be available at that time. Such asynchronously failing messages can negatively impact the user experience. It is best practice to fail fast by directly transferring funds from a sender's account during message execution. After that, the deposit can still be handled in the `EndBlock` function.

Recommendation

We recommend moving funds from a sender's account to an escrow account during message execution.

Status: Resolved

12. Events are not emitted consistently across the application, which may impact the ability of third-party applications to be integrated

Severity: Minor

Cosmos events are emitted by applications to notify third-party applications such as block explorers about events that occur on the chain. Not emitting events consistently across the application may impact the ability of these applications to parse the activity that happens in Sei, impacting user and developer experience.

The following instances of messages and functionality lack emission of events:

- `x/epoch/module.go:188`
- `x/dex/keeper/msgserver/msg_server_unregister_contract.go:12`

Recommendation

We recommend emitting Cosmos events consistently across the application, covering all messages and ABCI-related events.

Status: Resolved

13. `ChangeAdmin` does not validate that the new admin is different from the current one, which may cause errors in contracts and applications relying on this logic

Severity: Minor

The `ChangeAdmin` function in the `tokenfactory` module in `x/tokenfactory/keeper/ msg_server.go:135` permits the admin of a token denom to change its address to a new one. However, there is no check that `NewAdmin` is different from the current one. An event of a change of admin is emitted nonetheless, which may cause unexpected behavior in applications relying on this information.

Recommendation

We recommend validating that the `newAdmin` is different from the current one in a denom of the `tokenfactory` module.

Status: Resolved

14. Inflation rate is set to 0%, while the documentation indicates an intended default inflation rate of 13%

Severity: Minor

In `x/mint/types/minter.go:26`, it is stated that the intended default initial inflation rate should be 13%. However, in line `21`, the initial inflation rate is set to `sdk.NewDec(0)`.

Recommendation

We recommend updating line `21` to reflect the intended inflation rate.

Status: Resolved

15. `UnregisterContract` should remove associated pairs after the contract is removed to avoid consuming unnecessary disk space on validators

Severity: Minor

Once the contract is removed in the `UnregisterContract` function in `x/dex/keeper/msgserver/msg_server_unregister_contract.go:12`, its associated states are not deleted. An attacker can exploit storage by registering contracts, and then unregistering them multiple times. The cost of such an attack is comparatively low as rent is refunded to the contract owner after unregistration.

Recommendation

We recommend removing all associated states including associated pairs when a contract is removed.

Status: Resolved

16. Inefficient implementation of `UpdateQuantityTickSize` and `UpdatePriceTickSize`

Severity: Informational

The implementation of `UpdateQuantityTickSize` in `x/dex/keeper/msgserver/msg_server_update_quantity_tick_size.go:11` iterates through the tick size list, performing validation and then iteration through the same list happens again in `SetQuantityTickSizeForPair`. A similar implementation is performed to update price tick size.

Recommendation

We recommend using the same loop for validating and setting quantity tick size and price tick size for pairs.

Status: Resolved

17. Strict parsing of boolean CLI arguments may degrade user experience

Severity: Informational

In `x/dex/client/cli/tx/tx_register_contract.go`, the arguments `[need hook]` and `[need order matching]` only check against the string `true`, which degrades the user experience. For example, if a user capitalizes the word, the flag is going to be set to false.

Recommendation

We recommend using `strconv.ParseBool` to parse the boolean.

Status: Resolved

18. Inefficient implementation of `CancelOrders`

Severity: Informational

The implementation of `CancelOrders` in `x/dex/exchange/cancel_order.go` iterates through all orders, calling `cancelOrder(cancel, orderbook.Longs)` and `cancelOrder(cancel, orderbook.Shorts)` for each one. This is inefficient, since an order is always just in either the long or short orderbook.

Recommendation

We recommend checking the `PositionDirection` for each cancel order and calling either `cancelOrder(cancel, orderbook.Longs)` or `cancelOrder(cancel, orderbook.Shorts)`, respectively.

Status: Resolved

19. Inefficient implementation of `PlaceOrders`

Severity: Informational

The implementation of `transferFunds` within the `PlaceOrders` function is inefficient. In `x/dex/keeper/msgserver/msg_server_place_orders.go:25-29`, there is a loop

validating that the fund amounts are different from `Nil`. However, this iteration is redundant as the same validation is performed in line `38`.

Recommendation

We recommend removing redundant validations, especially if they involve iterations, as they may impact performance.

Status: Resolved

20. Redundant checks are inefficient

Severity: Informational

The codebase contains redundant checks that lead to inefficiencies:

- The `ContractAddr` in `x/dex/keeper/msgserver/msg_server_register_contract.go:71` is a duplicate validation as it is being checked in line `74`.
- The `ValidateBasic` function in `x/dex/client/cli/tx/tx_cancel_orders.go:61` is a duplicate validation as it is invoked through the `GenerateOrBroadcastTxCLI` function.
- The `ValidateBasic` function in `x/dex/client/cli/tx/tx_contract_deposit_rent.go:38` is a duplicate validation as it is invoked through the `GenerateOrBroadcastTxCLI` function.
- The `ValidateBasic` function in `x/dex/client/cli/tx/tx_place_orders.go:100` is a duplicate validation as it is invoked through the `GenerateOrBroadcastTxCLI` function.
- The `ValidateBasic` function in `x/dex/client/cli/tx/tx_register_contract.go:52` is a duplicate validation as it is invoked through the `GenerateOrBroadcastTxCLI` function.
- The `ValidateBasic` function in `x/dex/client/cli/tx/tx_register_pairs.go:40` is a duplicate validation as it is invoked through the `GenerateOrBroadcastTxCLI` function.
- The `ValidateBasic` function in `x/dex/client/cli/tx/tx_unregister_contract.go:32` is a duplicate validation as it is invoked through the `GenerateOrBroadcastTxCLI` function.
- The `ValidateBasic` function in `x/dex/client/cli/tx/tx_update_price_tick_size.go:40` is a duplicate validation as it is invoked through the `GenerateOrBroadcastTxCLI` function.
- The `ValidateBasic` function in `x/dex/client/cli/tx/tx_update_quantity_tick_size.go:40` is a duplicate validation as it is invoked through the `GenerateOrBroadcastTxCLI` function.

Recommendation

We recommend removing the aforementioned redundant checks.

Status: Resolved

21. Uninformative error message

Severity: Informational

The errors in `x/dex/ante.go:59` and `71` provide no information whether `priceTickSize` or `quantityTickSize` does not fulfill the multiple of tick size criterion.

Recommendation

We recommend changing the error to “price needs to be multiple of price tick size” and “quantity needs to be multiple of quantity tick size”, respectively.

Status: Resolved

22. Misleading short description in `ContractDepositRent` command

Severity: Informational

The “Short” description for `CmdContractDepositRent` in `x/dex/client/cli/tx/tx_contract_deposit_rent.go:16` is misleading.

Recommendation

We recommend changing the short description from “Unregister exchange contract” to “Contract deposit rent”.

Status: Resolved

23. Use of deprecated module and functions

Severity: Informational

Deprecated code is used in the codebase:

- The module `github.com/golang/protobuf` is deprecated. Use the `google.golang.org/protobuf` module instead.
- The function `EmitEvents`, used in `x/dex/contract/abci.go` and `x/dex/contract/execution.go` is deprecated. Use `EmitTypedEvents` instead.

- The function `Title`, used in `x/dex/types/wasm.utils.go` is deprecated since it does not handle Unicode punctuation. Use `golang.org/x/text/cases` instead.

Recommendation

We recommend replacing deprecated modules and functions as mentioned above.

Status: Acknowledged

24. Use `map[T]struct{}{}` over `map[T]bool` to increase efficiency

Severity: Informational

In `x/dex/types/wasm/block_hooks.go:59`, `map[uint64]bool` is used as a hash set. An alternative data type can be faster and more efficient in terms of memory.

Recommendation

We recommend changing the data type to `map[uint64]struct{}{}`. See https://itnext.io/set-in-go-map-bool-and-map-struct-performance-comparison-5315b4b107b for more details.

Status: Resolved

25. Unchecked error in `SubmitFraudChallenge` function might lead to inefficiencies

Severity: Informational

In `x/nitro/keeper/msg_server.go:76`, there is no check for an error on `server.GetTransactionData`. In case of an error, the transaction was not found but still added into the slice of transactions. This will be caught later on in the replay function, so this issue has no security implications. However, it is inefficient to perform extra computation on input that is known to fail.

Recommendation

We recommend not adding transactions in the case of an error to the slice.

Status: Resolved

26. Missing usage description for transaction and query CLI commands

Severity: Informational

The transaction and query CLI commands of the dex module are missing a long message to describe their usage, which could be helpful for users and external developers.

Recommendation

We recommend specifying a long message for all transaction and query CLI commands. Each command should provide a description of how to correctly use the command.

Status: Resolved

27. Unnecessary use of signed integers is inconsistent and may be prone to programming errors

Severity: Informational

In the `sei-cosmwasm` package, there are messages and types definitions for CosmWasm smart contracts to interact with the Sei Cosmos SDK chain. Some of the fields in these types are defined as signed integers instead of unsigned integers, which is inconsistent and error prone. Examples can be found in:

- `packages/sei-cosmwasm/src/query.rs:71`
- `packages/sei-cosmwasm/src/msg.rs:49` and `53`

Recommendation

We recommend using unsigned integers whenever possible, to maximize consistency and reduce potential future programming errors.

Status: Resolved

28. Resolve TODOs before release

Severity: Informational

There are multiple TODOs in the codebase that may refer to sensitive or importante logic. Instances are:

- `x/dex/keeper/msgserver/msg_server_register_contract.go:29`
- `cmd/seid/cmd/root.go:373`
- `x/dex/ante.go:54`
- `x/dex/keeper/query/grpc_query_get_historical_prices.go:22`
- `x/dex/module.go:221`

- app/ante.go:79
- x/nitro/keeper/msg_server.go:109
- x/epoch/module.go:174

Recommendation

We recommend resolving all TODOs in the codebase before production release, especially those which may relate to sensitive or important logic.

Status: Resolved

29. Miscellaneous comments

Severity: Informational

Across the codebase, various instances of unused code and misleading comments have been found.

Recommendation

The following are some recommendations to improve the overall code quality and readability:

- Remove unused errors in x/dex/types/errors.go with error codes 5, 7, 9, 11, 1100, 1101, and 1102.
- Resolve the grammar issue in the description of the error with code 2 in x/dex/types/errors.go.
- Remove the unnecessary duplication of fund.Amount.isNil check in x/dex/keeper/msgserver/msg_server_place_orders.go.
- Remove the redundant alias fmt in x/dex/types/params.go:4.
- Remove commented code in x/dex/client/cli/tx/tx.go:10.
- Remove unused constants in x/dex/types/wasm/settlement.go:8.
- Remove unused constants in x/dex/types/keys.go:166.
- Remove unused constant DefaultIndex in x/dex/types/genesis.go.
- Remove unused constant ContractAddressAddressLength in x/dex/migrations/v2_to_v3.go.
- Remove unused variable DexPerPairWhitelistedKeys in x/dex/contract/whitelist.go.
- Remove unused variable DefaultRelativePacketTimeoutTimestamp in x/dex/client/cli/tx/tx.go.
- Remove unused function GetContractPositionEffect in x/dex/types/wasm/utils.go.
- Remove unused function NewAddAssetMetadata in x/dex/types/gov.go.
- Remove unused function NewPlainKeeper in x/dex/keeper/keeper.go:34.
- Remove unused functions TwapPrefix, GetKeyForHeight, Cancel, and AccountActiveOrdersPrefix in x/dex/types/keys.go.
- Remove unused functions SetDefaultPriceTickSizeForPair and SetDefaultQuantityTickSizeForPair in x/dex/keeper/tick.go.

- Remove unused function `NewParams` in `x/dex/types/params.go`.
- Remove unused parameter `config` of `ValidateGenesis` in `x/dex/module.go`.
- Remove unused parameter `queryRoute` of `GetQueryCmd` in `x/dex/client/cli/query/query.go`.
- Remove unused parameter `ctx` of `ValidateBasics` in `x/dex/keeper/msgserver/msg_server_register_contract.go`.
- Remove unused function `GetStakingDependencyGenerator` in `aclmapping/staking/mappings.go:18`.
- Remove unused function `PtrCopier` in `utils/pointer.go:3`.
- Remove unused functions `ConvertWholeToMicroDenom` and `ConvertMicroToWholeDenom` in `utils/denom_conversion.go:9` and `14`.
- Remove unused parameter `req.Height` in `x/dex/keeper/query/grpc_query_match_result.go:12`.
- Remove the if statement in `app/app.go:261` which is always true.
- Remove commented logic in `x/tokenfactory/keeper/msg_server.go:106`.
- Remove commented logic related to the batch verifier in `app/ante.go:71` and `778`.
- Fix misleading comment in `x/nitro/keeper/msg_server.go:55`.
- Fix misleading comment in `x/dex/module.go`, `getAllContractInfo` does not process contracts with zero rent balance.
- Fix misleading comment in `x/mint/keeper/keeper.go:121` and `128` as the function is not currently used in `BeginBlocker`.
- Fix misleading comment in `x/mint/keeper/migrations.go:18`.
- Fix misleading comment in `x/tokenfactory/keeper/createdenom.go:12`.
- Return the correct type message instead of `TypeMsgRecordTransactionData` in `x/nitro/types/msgs.go:70`.

Status: Resolved