Security Audit Report

ZIGChain

v1.0

August 25, 2025

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 GmbH has been engaged by Highend Technologies LLC to perform a security audit of the ZIGChain Cosmos SDK implementation.

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/ZIGChain/zigchain
Commit	`a0584a1329b55109ff1d612636534f7bada3d01f`
Scope	All modules were in scope.
Fixes verified at commit	`18a9e5d1a42dc9564be09b6feb2e6dbf2f40983e` 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

The scope of the audit for ZIGChain features the following custom modules:

- The `x/dex` module enables users to create pools, provide or withdraw liquidity, and swap tokens.
- The `x/factory` module allows users to create custom native tokens with granular access controls via the bank and metadata admin, while enforcing the max supply.
- The `x/tokenwrapper` module facilitates the bridging between ZIG tokens on Axelar (originating from Ethereum) and native ZIG tokens in ZIGChain. Tokens are wrapped or unwrapped accordingly when transferring or receiving from ZIGChain.

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	-
Code readability and clarity	Medium-High	-
Level of documentation	High	Detailed documentation is available at https://docs.zigchain.com and in the respective `README` files.
Test coverage	Low-Medium	`go cover` reports a coverage of `45.2%`, which excludes proto-generated files. For more information on how we compute test coverage, please see this section in the appendix.

Summary of Findings

No	Description	Severity	Status
1	Incorrect token amount can be added to the pool reserves	Critical	Resolved
2	Malicious chains can send IBC packets to steal funds from the `tokenwrapper` module	Critical	Resolved
3	Disabling the `tokenwrapper` module prevents IBC packets from being transferred	Major	Resolved
4	Incorrect token refund mechanism may result in users receiving the wrong tokens	Major	Resolved
5	Insufficient `uzig` coins in the module balance may cause IBC incoming transfers to fail	Major	Resolved
6	Packet Forward Middleware (PFM) is not implemented	Major	Resolved
7	Fees are not incurred during `MsgSwapExactOut`	Major	Resolved
8	Burning tokens does not increase minting capacity	Major	Resolved
9	Incomplete `GenesisState` validation in the tokenwrapper module may lead to an invalid state	Minor	Resolved
10	Incorrect permission handling denies metadata access to bank admins	Minor	Resolved
11	Lack of cross-validation between `DenomList` and `DenomAuthList` may lead to an invalid state	Minor	Resolved
12	Incomplete validation in the `Pool` function may allow invalid addresses	Minor	Resolved
13	`DenomAuth` validation function incorrectly rejects empty admin strings	Minor	Resolved
14	Centralization issue regarding the token wrapper's operator	Minor	Acknowledged
15	Insufficient validation of `CreateFeeDenom` in `UpdateParams` may prevent denom creation	Minor	Resolved
16	Incorrect pagination handling in `DenomsByAdmin` may lead to inaccurate results	Minor	Resolved

17	Overwriting fields in `ValidateBasic` will not persist during actual execution	Minor	Resolved
18	Imbalanced liquidity deposits can lead to immediate loss	Minor	Resolved
19	Outdated dependencies with potential risks	Minor	Acknowledged
20	Input string modification before validation violates best practices	Minor	Resolved
21	Insecure default keyring backend	Minor	Resolved
22	Missing `MINIMAL_LIQUIDITY` lock when minting first depositor share tokens	Minor	Resolved
23	Missing upper bound check on `decimalDifference`	Minor	Resolved
24	Deviation from specifications in `BurnTokens` may result in unauthorized burning	Informational	Resolved
25	Over-protective validations reduce code clarity	Informational	Resolved
26	Unnecessary `scaleFactor` implementation reduces calculation precision	Informational	Resolved
27	Swapped parameters in denom validation error messages	Informational	Resolved
28	Commented security logic creates code confusion and maintenance issues	Informational	Resolved
29	Error when withdrawing validator commission during export is ignored	Informational	Resolved
30	Token wrapping rejects amounts that convert to zero	Informational	Resolved
31	Unused governance authority for module parameters	Informational	Resolved
32	Administrative operations lack event emission	Informational	Resolved
33	Error handling can be improved using the `%w` formatter	Informational	Resolved
34	Documentation incorrectly lists allowed characters in subdenoms	Informational	Resolved
35	One-step operator and admin transfer may lead to lost admin privileges	Informational	Resolved

36	Unnecessary metadata validation may cause confusion	Informational	Resolved
37	Missing underflow protection in the token burned amount calculation	Informational	Resolved
38	Use of magic string for factory denom delimiter reduces maintainability	Informational	Resolved
39	Missing denom `MaxSupply` check may lead to incorrect states	Informational	Resolved
40	Misleading error messages and comments	Informational	Resolved
41	Insufficient validation in `ValidateBasic` may lead to panic	Informational	Resolved
42	Possibility to bypass fees by swapping very low amounts	Informational	Resolved
43	Potential non-unique pool UIDs due to denom separator collision	Informational	Resolved
44	Presence of debugging artifact in production code	Informational	Resolved
45	Magic numbers used for pool formula validation	Informational	Resolved
46	Unclear “invalid” state variable values in `DefaultGenesis`	Informational	Resolved
47	Potential division by zero during token balance calculation	Informational	Resolved
48	Unused `ibcv2TransferStack` initialization may cause confusion	Informational	Resolved
49	Hardcoded pool UID string construction reduces maintainability	Informational	Resolved
50	Error events emitted for non-error cases may cause confusion	Informational	Resolved
51	Incorrect comments and unused types may cause confusion	Informational	Resolved
52	Unused functions and errors implemented in the codebase	Informational	Resolved
53	Incorrect use of `SignerCheck` may cause confusion	Informational	Resolved
54	Incorrect base and quote values returned after liquidity removal	Informational	Resolved

55	`Pool` removal message is not implemented	**Informational**	**Resolved**
56	Miscellaneous comments	**Informational**	**Resolved**

Detailed Findings

1. Incorrect token amount can be added to the pool reserves

Severity: Critical

When adding liquidity in `x/dex/keeper/msg_server_add_liquidity.go:45-52`, the base and quote tokens provided by the caller are sorted and validated with the pool reserves. Ideally, `pool.Coins[0].Denom` is the `base` denom while `pool.Coins[1].Denom` is the `quote` denom.

If the user provides `msg.Quote` as the quote denom and `msg.Base` as the base denom (essentially in a different order than the pool assets), the `sdk.NewCoins` operation on line `45` will automatically sort them.

However, the sorted coins are not used when increasing the pool assets in lines `138-139`. Instead, the user-provided `msg.Base` token, which is the quote token, will be incorrectly added to the base token reserves amount.

Consequently, attackers can add liquidity in the incorrect order, tricking the pool into thinking it has more assets than intended, and performing a swap to steal valuable assets from it, resulting in a loss of funds for the liquidity providers.

Please see the proof of concept in the appendix to reproduce the issue.

Recommendation

We recommend adding the pool reserves based on the sorted tokens:

```
pool.Coins[0] = pool.Coins[0].AddAmount(sortedCoins[0].Amount)
pool.Coins[1] = pool.Coins[1].AddAmount(sortedCoins[1].Amount)
```

Status: Resolved

2. Malicious chains can send IBC packets to steal funds from the `tokenwrapper` module

Severity: Critical

In `x/tokenwrapper/module/module_ibc.go:181`, the packet's source port and channel are validated to be the configured IBC settings from Axelar. This is problematic because malicious chains can fake the source port and channel to spoof themselves as Axelar when connecting with ZIGChain.

The malicious chain can craft the `FungibleTokenPacketData` packet such that it sends the module denom with the attacker's address as the receiver. This will cause the worthless

funds sent by the malicious chain to be locked and replaced with legitimate `uzig` tokens (see `x/tokenwrapper/module/module_ibc.go:213-240`), causing a loss of funds for the `tokenwrapper` module.

Recommendation

We recommend validating the packet's destination port and channel.

Status: Resolved

3. Disabling the `tokenwrapper` module prevents IBC packets from being transferred

Severity: Major

Disabling the `tokenwrapper` module should only affect the sending and receiving of `uzig` / Axelar-Zig tokens via IBC. However, in `x/tokenwrapper/module/module_ibc.go:308-311`, when the `tokenwrapper` is disabled, no tokens can be sent out to any channel via IBC.

This results in a complete halt of IBC token transfers (out), affecting all tokens and channels, and not only the sending of `uzig` tokens.

Recommendation

We recommend removing the code in `x/tokenwrapper/module/module_ibc.go:308-311`, as the required checks are already performed in lines `331-338`.

Status: Resolved

4. Incorrect token refund mechanism may result in users receiving the wrong tokens

Severity: Major

In `x/tokenwrapper/module/module_ibc.go:284-442`, the tokenwrapper module intercepts outgoing `uzig` tokens and replaces them with Axelar-Zig IBC vouchers when sending to Axelar.

However, when IBC transfers fail due to acknowledgement errors or timeouts, the refund mechanism is incomplete. In `x/tokenwrapper/module/module_ibc.go:261-281`, the `OnAcknowledgementPacket` and `OnTimeoutPacket` functions only pass messages down the IBC stack without handling the token conversion back to `uzig`. This results in users

receiving Axelar-Zig IBC vouchers instead of their original `uzig` tokens. Additionally, the `TotalTransferredOut` amount is not properly reset.

Consequently, this issue causes users to receive incorrect tokens after failed IBC transfers to the Axelar chain, preventing them from immediately recovering their original `uzig` tokens.

We classify this issue as major since the users can send the tokens back to Axelar, and then send the tokens back to ZIGChain and recover their `uzig` tokens.

Recommendation

We recommend updating the `OnAcknowledgementPacket` and `OnTimeoutPacket` functions to properly handle refunds for failed Axelar-Zig token transfers. After passing messages down the IBC stack, these functions should detect refund scenarios (timeouts and `Acknowledgement_Error` responses) and process them similarly to `OnRecvPacket`.

The refund process should include:

1. Calling `LockTokens` for the refunded IBC vouchers,
2. Converting to the correct decimal amount for `uzig`,
3. Unlocking `uzig` tokens to the user via `UnlockTokens`,
4. Subtracting the amount from `TotalTransferredOut`.

Status: Resolved

5. Insufficient `uzig` coins in the module balance may cause IBC incoming transfers to fail

Severity: Major

In `x/tokenwrapper/module/module_ibc.go:237`, the call to `UnlockTokens` may fail if there are not enough `uzig` coins in the module balance. This could prevent the successful execution of `OnRecvPacket`, resulting in users not receiving their `uzig` tokens after transferring Axelar-Zig from the Axelar blockchain.

Additionally, after the Incorrect token refund mechanism may result in users receiving wrong tokens issue is fixed, if insufficient `uzig` tokens are available in the module balance when the error acknowledgement or the timeout are posted on chain, the `UnlockTokens` call in the step 3 of the recommended fix will fail and prevent user refunds.

In such cases, users would need to wait for the module to be funded with additional `uzig` tokens, before a relayer can trigger the refund mechanism through `AcknowledgePacket` or `TimeoutPacket` , as no automatic refund mechanism is implemented.

Recommendation

We recommend monitoring the amount of `uzig` tokens in the tokenwrapper module account to ensure there are always sufficient tokens to accommodate users transferring their tokens from Axelar.

Status: Resolved

6. Packet Forward Middleware (PFM) is not implemented

Severity: Major

In `app/ibc.go:122-130`, `app.PacketForwardKeeper` is initialized and then added as `ICS4Wrapper` to `app.TransferKeeper` in line `137`.

Further, in lines `183-189`, the PFM is added to `transferStack`. However, in `app/ibc.go:191`, the PFM is discarded because it is not included in the `tokenwrapper` middleware.

Finally, in line `201`, `app.PacketForwardKeeper` is replaced by `transferICS4Wrapper` as the `ICS4Wrapper` in `app.TransferKeeper`.

Consequently, the PFM is not being utilized as intended in the final ICS4Wrapper stack, and the intended functionality of the PFM is not achieved in the chain.

Recommendation

We recommend updating the code to include both the PFM and the IBC callback stack in the `tokenwrapper` middleware, or removing the PFM initialization if it is not used.

Status: Resolved

7. Fees are not incurred during `MsgSwapExactOut`

Severity: Major

In `x/dex/keeper/msg_server_swap_exact_out.go:221`, the `incomingBase` variable computes the required incoming token the caller needs to provide after taking into account the swap fees.

However, since `feeRatePerHundredThousand.Quo(scalingFactor)` is evaluated first, and `feeRatePerHundredThousand` is less than the `scalingFactor`, the actual computed fee will be rounded to zero. This will result in a loss of fees for liquidity providers, enabling users to swap tokens without incurring any fees.

Recommendation

We recommend computing the fees for `MsgSwapExactOut` correctly.

Status: Resolved

8. Burning tokens does not increase minting capacity

Severity: Major

In `x/factory/keeper/msg_server_mint_and_send_tokens.go:39-42`, the `totalMinted` value, which is the result of incrementing `currentDenom.Minted` by the amount of new tokens `msg.Token.Amount`, is validated against `currentDenom.MaxSupply`. If the `currentDenom.MaxSupply` value is exceeded, the minting is aborted.

However, the `currentDenom.MaxSupply` value represents the lifetime minting cap, and not the circulating supply cap, which is more common. This means that every time a number of tokens is burned, this number is not added to the minting capacity.

This behaviour can be unexpected for users and be perceived as a DoS when the tokens are supposed to be indefinitely mintable.

Recommendation

We recommend renaming the `currentDenom.MaxSupply` as `MintingCap` to reflect the lifetime and not circulating aspect of the constant. Additionally, a separate constant `MaxSupply` could be introduced to support indefinitely mintable tokens.

Status: Resolved

9. Incomplete `GenesisState` validation in the `tokenwrapper` module may lead to an invalid state

Severity: Minor

In `x/tokenwrapper/types/genesis.go:31-35`, the `Validate` function on `GenesisState` in the `tokenwrapper` module does not currently execute any validator functions. This could result in a potentially invalid state being accepted.

Recommendation

We recommend enhancing the `Validate` function by incorporating the following validator functions to ensure comprehensive validation:

- `ValidateClientID`
- `ValidateSourcePort`

- `ValidateSourceChannel`
- `ValidateDenom`
- `ValidateDecimalDifference`

We also recommend checking that `TotalTransferredIn` and `TotalTransferredOut` are not negative.

Additionally, we recommend verifying that `OperatorAddress` is a valid address.

Status: Resolved

10. Incorrect permission handling denies metadata access to bank admins

Severity: Minor

In `x/factory/keeper/auth.go:49-58`, the check for whether a signer has bank or metadata permissions returns an error if the metadata admin is disabled when checking for metadata permissions. However, a bank admin should inherently have metadata permissions. Consequently, if the metadata admin is disabled and a signer is a valid bank admin, the signer is incorrectly denied metadata permissions.

Recommendation

We recommend modifying the logic to not return an error if the signer is `denomAuth.BankAdmin`, allowing the execution flow to continue and correctly granting permissions.

Status: Resolved

11. Lack of cross-validation between `DenomList` and `DenomAuthList` may lead to an invalid state

Severity: Minor

In `x/factory/types/genesis.go:27-54`, the genesis `DenomList` and `DenomAuthList` are validated independently.

However, there is no check to ensure that each denom in `DenomList` has a corresponding entry in `DenomAuthList`, and vice versa.

This can lead to an invalid state where the denoms and their auth counterpart are mismatched.

Recommendation

Similar to the checks performed in `x/dex/types/genesis.go:79-93`, we recommend implementing cross-validation to ensure that all denoms in `DenomList` have their counterparts in `DenomAuthList`, and vice versa.

Status: Resolved

12. Incomplete validation in the `Pool` function may allow invalid addresses

Severity: Minor

In `x/dex/types/genesis.go:100-142`, the `Validate` function on `Pool` checks all fields except the `Address` field. Consequently, a pool with an invalid address could pass the `Validate` function which could lead to an invalid state.

Recommendation

We recommend adding standard address validation and checking that the address is equal to `authtypes.NewModuleAddress(p.poolID)`.

Status: Resolved

13. `DenomAuth` validation function incorrectly rejects empty admin strings

Severity: Minor

In `x/factory/types/genesis.go:119-135`, the validation function for `DenomAuth` checks that both `BankAdmin` and `MetadataAdmin` are valid using the `AddressCheck` function, which does not allow empty strings.

However, it is a valid state for both `BankAdmin` and `MetadataAdmin` to be empty when they are disabled.

This validation could lead to valid states being incorrectly rejected within genesis.

Recommendation

We recommend modifying the `DenomAuth` validation function to allow empty strings for both `BankAdmin` and `MetadataAdmin`, thereby accommodating valid states where these admins are disabled.

Status: Resolved

14. Centralization issue regarding the token wrapper's operator

Severity: Minor

As detailed in `x/tokenwrapper/README.md:62-68`, the operator address in the token wrapper has the following capabilities:

- Set IBC source port, channel, and denom for Axelar-ZIGChain communication
- Configure the `decimal_difference` parameter to manage token scaling between chains
- Enable/disable the module in case of IBC channel anomalies
- Fund the module wallet with native `uzig` tokens
- Withdraw excess tokens from the module wallet

The above privileges seem too extensive for a single entity (the Operator).

For instance, the Axelar `denom` and `decimal_difference` could be set statically in the code, as they should be known in advance. The IBC source port and channel can be set only once and cannot be changed by the operator once they are set. If necessary, an upgrade or a governance proposal could be used to change the source port and channel.

Additionally, withdrawing excess tokens from the module wallet also means that the operator can withdraw IBC vouchers of the Axelar-Zig tokens, which conceptually belong to the users who hold the `uzig` tokens on ZIGChain. By withdrawing them from the module wallet, the operator would deny some users the ability to transfer their `uzig` tokens out of the chain (to Axelar).

We consider that funding the module wallet could also be permissionless, not reserved only for the operator.

Recommendation

We recommend considering adjusting the operator capabilities to what is truly necessary, rather than centralizing all the above powers in one entity.

Status: Acknowledged

The client states that: "We appreciate the audit team's concerns and would like to address the key points raised:

Security-Driven Design

The operator-controlled funding and withdrawal capabilities are primarily security measures. Maintaining large reserves within the module creates greater risk than using a separate, secured wallet that funds the token wrapper based on operational needs. This approach minimizes attack surface and reduces potential impact.

Emergency Response

Time-sensitive security responses are critical in blockchain environments. While we can detect attacks within seconds, waiting for governance proposals during active attacks would be counterproductive and dangerous. However, we acknowledge and we seem the value of separating pause and unpause privileges - pause functionality could be extended to broader groups or automated tools, while unpause remains more restricted. Adding this implementation.

Implementation Details

The operator wallet is a multisignature wallet secured by the ZIGChain Foundation, ensuring no single individual has unilateral control. We maintain Axelar denomination and decimal_difference as configurable parameters to ensure adaptability for possible future bridge transitions. Since the operator manages bridge funds, keeping IBC configuration under the same role maintains operational consistency."

15. Insufficient validation of `CreateFeeDenom` in `UpdateParams` may prevent denom creation

Severity: Minor

In `x/factory/keeper/msg_update_params.go:12-23`, the `UpdateParams` function does not verify whether the `CreateFeeDenom` is an existing denom. If the request was processed with an invalid `CreateFeeDenom` (a valid name that does not exist on the chain), it could prevent anyone from creating new denoms. This is because creators would not have any of the invalid tokens in their balance.

If this situation occurs, it would necessitate a new governance proposal to rectify the issue, requiring a wait for the voting period before new denoms could be created again.

We classify this issue as minor since the function can only be called via governance, and proposals are expected to be carefully reviewed.

Recommendation

We recommend adding a check to ensure that `req.Params.CreateFeeDenom` has some supply on chain before setting the parameters. This will help prevent the accidental setting of an invalid denom.

Status: Resolved

16. Incorrect pagination handling in `DenomsByAdmin` may lead to inaccurate results

Severity: Minor

In `x/factory/keeper/query_denom_by_admin.go:26-36`, the iteration over the Denom Auths does not follow the standard use of the request's pagination. It uses pagination to access the Denom Auths, but only returns those where the `req.Admin` is both the Bank and Metadata admin.

This approach can result in an incorrect `Total` and may return zero `Denoms` while still providing a `NextKey`, leading to potential confusion and incorrect data handling.

Recommendation

We recommend implementing a dedicated store for `DenomByAdmin`, which seems to be the intended approach as indicated by the comment "*TODO: Pre store this so we don't have to iterate over all denoms*" on line `17`. This change would improve efficiency and accuracy in handling pagination and querying Denom Auths by the admin.

Additionally, in case the query should return the Denom Auths where the `req.Admin` is either the Bank or Metadata admin, we recommend updating the code to match this behaviour.

Status: Resolved

17. Overwriting fields in `ValidateBasic` will not persist during actual execution

Severity: Minor

In `x/factory/types/message_set_denom_metadata.go:79`, the `msg.Metadata.URIHash` will be forcefully overwritten to an empty hash if the provided `msg.Metadata.URI` is empty. This is to prevent users from updating the `msg.Metadata.URIHash` without updating the `msg.Metadata.URI`.

However, this overwrite will not persist when `MsgSetDenomMetadata` is executed, allowing users to update the `msg.Metadata.URIHash` field without providing the `msg.Metadata.URI` field.

This issue also occurs in the following instances:

- `x/factory/types/msg_create_denom.go:90`
- `x/factory/types/message_update_denom_uri.go:64`

Recommendation

We recommend overwriting the fields during the actual execution.

Status: Resolved

18. Imbalanced liquidity deposits can lead to immediate loss

Severity: Minor

In `x/dex/keeper/msg_server_add_liquidity.go:151-194`, the function `CalculateLiquidityShares` is defined, which is used to process liquidity deposits. However, liquidity deposits are allowed to be asymmetric, meaning that tokens deposited by the user in a different proportion compared to the current pool balance.

An unsuspecting user could experience immediate loss if they were tricked into depositing an imbalanced amount of tokens in the pool, or if they did it by mistake.

A hypothetical scenario could be:

1. A pool holds 100K USDC and 1 BTC
2. Alice adds 100K BTC and 1 USDC
3. The pool now holds: 100,001 BTC + 100,001 USDC
4. Alice receives LP shares representing 50% of the pool, being 50K BTC + 50K USDC

In this scenario, the final value of the liquidity position owned by Alice is $5.05M, which is half of the deposited $10M, despite no pool operations having been performed after the deposit.

Such a theoretical scenario could be implemented as a malicious user interface, where the victim would not notice the swap in the amount of BTC on one side and USDC on the other. The attacker could immediately benefit from the attack by arbitraging: purchasing Bitcoins at a rate of 1 BTC = 1 USDC, and selling on another exchange for a profit.

While there is the possibility of loss of funds, we classify this issue as minor since the exploitation requires the victim to make a mistake or the attacker to entice the victim into making an imbalanced deposit.

Recommendation

We recommend implementing one of the following approaches:

- Restrict permissionless liquidity deposits until a certain level of liquidity in the pool
- Require deposits to maintain the same tokens ratio, or immediately refund excess tokens to the user
- Implement an advanced math formula to determine what degree of imbalance is allowed, given the deposit value and total liquidity in the pool
- Implement "Time-weighted average price" oracles instead of spot price

Status: Resolved

19. Outdated dependencies with potential risks

Severity: Minor

In `go.mod`, the project uses outdated versions of core dependencies while commenting out available newer versions. Multiple Cosmos SDK modules and other critical dependencies exhibit significant version gaps (e.g., `cosmossdk.io/core v0.11.3` while `v1.0.0` is available, `cosmossdk.io/client/v2 v2.0.0-beta.5` while `v2.10.0-beta.3` is available).

This practice of intentionally using older versions may expose the application to known security vulnerabilities and overlook important bug fixes, even if it is done for stability reasons.

Recommendation

We recommend reviewing pinned dependency versions against CVE databases and establishing a regular update schedule with comprehensive testing.

Status: Acknowledged

The client states that "Most of this versions are unreachable due to our current Cosmos SDK version. We tried to update to version 53 unsuccesfully before the submit for the security audit due to some issue during simulation tests. We still will push as far as we can with the current sdk version. Once the fixes for the version 53 are implemented we will move forward with version 53."

20. Input string modification before validation violates best practices

Severity: Minor

In `x/factory/keeper/query_denom_auth.go:49` and `x/factory/keeper/query_ denom_get.go:20`, the code performs `req.Denom =`

`strings.ReplaceAll(req.Denom,"'","/")` before calling `validators.CheckDenomString`. This preprocessing modifies user input before validation, which violates input validation best practices that recommend validating original user input.

While the transformation is predictable and validation still occurs, this approach can lead to user confusion when queries for "*sub'denom*" are processed as queries for "*sub/denom*", and makes the validation logic less transparent.

Recommendation

We recommend validating the original user input directly.

Status: Resolved

21. Insecure default keyring backend

Severity: Minor

In `cmd/zigchaind/cmd/root.go:93-96`, the initial command is being processed using the `overwriteFlagDefaults` function, in order to provide a default value for the chain ID and keyring backend.

However, the default value for the keyring backend is declared to be `"test"`. This poses a potential risk of misuse in production. Ideally, when the user does not specify the keyring backend explicitly, the node start should be aborted.

Using the `"test"` keyring backend in production should be avoided at all costs.

Recommendation

We recommend either removing the `"test"` keyring backend from default values or replacing it with a safer alternative like `"os"` keyring.

Status: Resolved

22. Missing `MINIMAL_LIQUIDITY` lock when minting first depositor share tokens

Severity: Minor

In `x/dex/keeper/msg_server_create_pool.go`, when creating a new pool, it is mandatory to also add some funds as the first, initial deposit. In this way, initial shares are minted, thus preventing the "share inflation attack", whereby creating one share, the attacker can take over the funds of subsequent depositors.

Nevertheless, following the typical XYK pools implementation, it was found that there is no concept of a `MINIMAL_LIQUIDITY` lock, preventing the redemption of 100% shares by any actor, including the pool creator.

While our tests did not produce a working Proof of Concept for share inflation attacks, this represents a deviation from established XYK pool best practices. It is worth analyzing whether minimal hardcoded LP shares should be "frozen" similar to Uniswap v2 and other protocols.

Recommendation

We recommend that a certain amount of LP shares be minted to the module during pool creation to avoid edge case scenarios related to share inflation attacks.

Status: Resolved

23. Missing upper bound check on `decimalDifference`

Severity: Minor

In the `x/tokenwrapper/keeper/keeper.go:275`, the `SetDecimalDifference` function is implemented, allowing for `DecimalDifferenceKey` to be adjusted. However, the `decimalDifference` is not checked against any upper bound or reasonable value.

For example, if 255 is set, during calculations, the `math.int` type will be easily exceeded, leading to overflow scenarios.

Recommendation

We recommend implementing an upper bound for the `decimalDifference` parameter passed to the `SetDecimalDifference` function by the operator, ideally as 18.

Status: Resolved

24. Deviation from specifications in `BurnTokens` may result in unauthorized burning

Severity: Informational

Based on `x/factory/README.md:74, 90-91`, token burning should be restricted to denoms created through the factory module, where the `BurnTokens` message sender must be the denom admin.

However, in `x/factory/keeper/msg_server_burn_tokens.go:14-73`, the `BurnTokens` function permits burning of any existing factory or non-factory tokens by any user, provided they hold the token in their balance (line `43`).

This inconsistency indicates the intended `BurnTokens` functionality, as outlined in `README.md`, is not implemented, potentially enabling unauthorized token destruction.

Furthermore, `README.md:18-20` specifies that bank admins can burn their denom from any account, a capability that remains unimplemented.

We initially classify this issue as major due to its significant deviation from the documented specification. However, if this behavior is deliberate and the documentation is inaccurate, we will reduce the severity to informational.

Recommendation

We recommend implementing the conditions outlined in `README.md` to:

- Limit token burning to authorized admins of factory-created denoms only.
- Enable authorized admins to burn tokens from any account.

If the current behavior is intentional, we recommend updating the documentation to accurately reflect the existing implementation.

Status: Resolved

25. Over-protective validations reduce code clarity

Severity: Informational

In `x/dex/types/params.go:75-82`, the variable `poolFee` of type `uint32` is validated to have a non-negative value. Standard linter warning is suppressed using `lint:ignore SA4003` annotation. While such validations can be useful for rejecting invalid states in the event of an accidental change of type, they should not be implemented directly in the source code.

Redundant validations negatively affect the clarity of the source code and hinder code reviews.

There are six similar issues in the codebase, all of which can be discovered by the `lint:ignore SA4003` annotation.

A similar issue has been discovered in `x/dex/keeper/msg_server_swap_exact_in.go:220-229`, where the validation `!newBaseTokenBalance.IsPositive()` is redundant unless `pool.Coins[fromCoin].Amount` is negative, which would be a severely incorrect state.

Recommendation

We recommend converting over-protective validations into unit test cases or using the standard Cosmos SDK "Invariants" feature.

Status: Resolved

26. Unnecessary `scaleFactor` implementation reduces calculation precision

Severity: Informational

In `x/dex/keeper/msg_server_remove_liquidity:130`, the scaling factor is implemented to maintain precision when performing integer division operations.

However, in the context of the code, it does not bring any real benefit. On the contrary, it reduces precision, since the fractional part during division will be lost.

For example, if there is no `scaleFactor`, there would be no need to calculate `shareRatio` when withdrawing liquidity:

```
denom1 := pool.Coins[0].Amount.Mul(lptoken.Amount).Quo(pool.LpToken.Amount)
denom2 := pool.Coins[1].Amount.Mul(lptoken.Amount).Quo(pool.LpToken.Amount)
```

Recommendation

We recommend removing the entire `scaleFactor` logic, which is also used in the `shareRatio` calculations, to eliminate unnecessary code complexity and enhance the precision of operations.

Status: Resolved

27. Swapped parameters in denom validation error messages

Severity: Informational

In `zutils/validators/coins.go:155-161`, the error message parameters are swapped in the `errorsmod.Wrapf` call. The current implementation passes `DenomRegexString, denom` as parameters, but the error message template expects the invalid denomination first, followed by the allowed pattern.

This results in a confusing error message like "*invalid coin: '[a-zA-Z][a-zA-Z0-9./]+' only 'invalid_denom' are allowed*" instead of the intended "*invalid coin: 'invalid_denom' only '[a-zA-Z][a-zA-Z0-9./]+' are allowed*".

Recommendation

We recommend swapping the parameter order to `denom, DenomRegexString` to match the error message template and provide clearer feedback to users.

Status: Resolved

28. Commented security logic creates code confusion and maintenance issues

Severity: Informational

In `zutils/validators/coins.go:138-151` and `zutils/validators/coins.go:203-218`, there are identical blocks of commented-out validation logic that check `types.DeconstructDenom`.

The comments suggest this validation should be applied for factory tokens to prevent exploitation attempts, but the code remains commented out with unclear guidance on when to enable it.

This creates confusion for developers and makes the codebase harder to maintain. Additionally, both `CheckDenomString` and `CheckCoinDenom` functions implement identical validation logic, leading to unnecessary code duplication.

Recommendation

We recommend removing the commented-out validation blocks entirely to eliminate confusion, as they would incorrectly reject valid non-factory denoms used in contexts like IBC settings. Consider simplifying `CheckCoinDenom` to call `CheckDenomString(coin.Denom)` directly to reduce code duplication and improve maintainability.

Status: Resolved

29. Error when withdrawing validator commission during export is ignored

Severity: Informational

In `app/export.go:84`, potential errors returned by `app.DistrKeeper.WithdrawValidatorCommission` are ignored, which means that some validators can miss their rewards. Next, within lines `125-138` all outstanding rewards are donated to the Community Pool.

This code only executes during state export for zero-height genesis, essentially preparing to start a new chain from the current state. Since this issue occurs rarely under particular circumstances and can be simulated beforehand, we classify it as informational severity.

Recommendation

We recommend handling all errors that occur in critical functions.

Status: Resolved

30. Token wrapping rejects amounts that convert to zero

Severity: Informational

In `x/tokenwrapper/module/module_ibc.go:220-224`, when converting the incoming 18-decimal token amount to a 6-decimal native token amount, the division (`amount.Quo(conversionFactor)`) rounds down. If the original amount is less than the `conversionFactor` (10^12), the `convertedAmount` becomes zero. The function correctly identifies this and returns an error acknowledgement, effectively rejecting such small transfers.

Recommendation

We recommend documenting this behavior clearly as a minimum transfer threshold if it is intended.

Status: Resolved

31. Unused governance authority for module parameters

Severity: Informational

In `x/tokenwrapper/keeper/msg_update_params.go:12-24`, the `tokenwrapper` module implements a `MsgUpdateParams` message controlled by the governance authority (typically the `x/gov` module account).

However, the module's `Params` struct (defined in `x/tokenwrapper/types/params.pb.go` and `x/tokenwrapper/types/params.go`) is empty with no fields.

This creates a governance interface that has permission to update parameters, but there are no parameters defined for it to actually update. All effective configuration and operational control (IBC settings, operator address, module enable/disable, decimal differences, etc.) are managed through the separate "operator" role via dedicated message types, such as `MsgUpdateOperatorAddress`, `MsgEnableTokenWrapper`, and `MsgUpdateIbcSettings`.

Recommendation

We recommend clearly documenting this design decision, as the current implementation may cause confusion about the governance model.

Status: Resolved

32. Administrative operations lack event emission

Severity: Informational

In `x/tokenwrapper/keeper/msg_server_update_operator_address.go:27-31`, successful administrative operations such as operator address changes do not emit events. A similar pattern exists across other message server files for IBC settings updates, module enable/disable actions, and wallet funding operations. This reduces transparency and makes it difficult for external systems to monitor critical administrative changes on-chain.

Recommendation

We recommend adding appropriate event emissions for all successful administrative operations.

Status: Resolved

33. Error handling can be improved using the `%w` formatter

Severity: Informational

As of Go 1.13, it is possible to use the `%w` verb in error messages, only for error values, which wraps the error such that it can later be unwrapped using `errors.Unwrap`.

Throughout the codebase, there are only 7 usages of this approach. At the same time, there are at least 65 cases of outdated error handling without the use of the `%w` verb.

Recommendation

We recommend updating the error handling approach used in the codebase.

Status: Resolved

34. Documentation incorrectly lists allowed characters in subdenoms

Severity: Informational

In `x/factory/docs/step_1_create_denom.md:3-4`, the documentation states that subdenoms can contain `[a-zA-Z0-9./]`, including uppercase letters, dots, and slashes.

However, the implementation in `CheckSubDenomString` (`zutils/validators/coins.go:264-284`) only allows `[a-z0-9]` - lowercase letters and numbers only. Additionally, dots serve as structural separators in the denom format `coin.{creator}.{subdenom}`, so allowing dots in subdenoms would break the parsing logic in `DeconstructDenom`.

Users reading the documentation may attempt to create subdenoms with uppercase letters, dots, or slashes, leading to validation failures and confusion about supported functionality.

Recommendation

We recommend updating `x/factory/docs/step_1_create_denom.md` to accurately reflect the implementation by changing the allowed character set from `[a-zA-Z0-9./]` to `[a-z0-9]`.

Status: Resolved

35. One-step operator and admin transfer may lead to lost admin privileges

Severity: Informational

The `UpdateOperatorAddress` function allows the current operator to execute a one-step ownership transfer, as seen in `x/tokenwrapper/keeper/msg_server_update_operator_address.go:28`.

Similarly, the `UpdateDenomAuth` function in `x/factory/keeper/msg_server_update_ denom_auth.go:41` allows the current admin to execute a one-step admin privileges transfer.

While this is a common practice, it presents a risk for the operator/admin addresses to be lost if the current operator/admin transfers the role to an incorrect address.

Implementing a two-step operator/admin address transfer would mitigate this risk. In a two-step process, the current admin proposes a new admin address, and the proposed new admin must then claim the role, allowing the old admin to retain control until the transfer is confirmed.

Recommendation

We recommend implementing a two-step operator/admin address transfer with the following flow:

- The current operator proposes a new operator address.
- The new operator account claims the operator role, which then applies the configuration changes.

Since it is currently possible to disable the bank or metadata admin by assigning it to an empty string, we recommend adding a new message (such as `DisableAdmin`) to allow disabling the bank or metadata admin in one step.

Status: Resolved

36. Unnecessary metadata validation may cause confusion

Severity: Informational

In `x/factory/keeper/msg_server_set_denom_metadata.go:15-19`, the code is validating `msg.Metadata` with a comment indicating "protection for internal calls." However, `msgServer` messages are not intended to be called by internal functions.

The exception to this is the wasm bindings, which call `ValidateBasic` on the entire `msg *types.MsgSetDenomMetadata`, provides more comprehensive validation than just `msg.Metadata.Validate`.

Consequently, this code is unnecessary and may cause confusion, as it would seem that it would be safe to call this function internally without calling `ValidateBasic` on the entire message.

Recommendation

We recommend removing this specific metadata validation. If there is any intention of allowing internal calls to `SetDenomMetadata`, it should be handled similarly to `wasmbinding/message_plugin.go:220`, where `ValidateBasic` is called on the entire message.

Status: Resolved

37. Missing underflow protection in the token burned amount calculation

Severity: Informational

In `x/factory/keeper/query_denom_all.go:44` and `x/factory/keeper/query_denom_get.go:71`, the code calculates total burned tokens using `denom.Minted.Sub(totalSupply)` without underflow protection. The

`cosmosmath.Uint.Sub` method will panic if `denom.Minted < totalSupply`, rather than returning a graceful error.

While this condition should never occur under normal operations for factory-created tokens, query endpoints should handle unexpected state conditions robustly to avoid ungraceful failures.

Recommendation

We recommend implementing safe arithmetic with underflow protection, such as checking if `denom.Minted >= totalSupply` before performing the subtraction, or using a method that returns an error instead of panicking.

Status: Resolved

38. Use of magic string for factory denom delimiter reduces maintainability

Severity: Informational

In `x/factory/types/denoms.go:44`, `57`, and `81`, a "full point" or "period" character is used as the delimiter for factory denoms. This character is used in various parts of the code as a magic string, which reduces maintainability and can lead to errors if changes are needed in the future.

Recommendation

We recommend defining this delimiter as a constant, such as `FactoryDenomDelimiterChar`, to enhance maintainability and ensure consistency throughout the codebase.

Status: Resolved

39. Missing denom `MaxSupply` check may lead to incorrect states

Severity: Informational

In `x/factory/types/genesis.go:62-104`, there is no validation check on `MaxSupply`, which could allow setting `MaxSupply` to `0` for a Denom. This could lead to state inconsistencies as `0` is not an allowed value of `MaxSupply`, as seen in `x/factory/types/msg_create_denom.go:54`.

Recommendation

We recommend adding a validation check to ensure that `MaxSupply` is set to a positive value, thereby preventing the establishment of an invalid state.

Status: Resolved

40. Misleading error messages and comments

Severity: Informational

The following instances represent incorrect error messages or comments:

- In `app/ante.go:43-45`, the condition `options.TXCounterStoreService == nil` is rejected with the message "`wasm store service is required for ante builder`". This message appears to refer to the past name of the service and is no longer accurate.
- The comment of "SwapOutResponse is the response to the SwapIn query" in `wasmbinding/bindings/query.go:69` should be replaced with "SwapOutResponse is the response to the SwapOut query."
- In `x/tokenwrapper/keeper/msg_server_withdraw_from_module_wallet.go:24-27`, the error message states "only the current operator can fund the module wallet", while in fact it should be "only the current operator can withdraw from the module wallet", and the comment states "lock tokens", while in fact it should be "unlock tokens".
- In `x/factory/keeper/msg_server_mint_and_send_tokens.go:33`, the comment states "Checks if signer has permission to update the denom auth", while in fact it should be "Checks if signer has permission to mint tokens".
- The error message in `x/dex/keeper/msg_server_create_pool.go:133-142` is incorrect because the action is transferring the funds from the sender to `params.Beneficiary`, not to the module.
- In `zutils/validators/coins.go:225`, the message states that "only lowercase letters (a-z) followed by lowercase letters (a-z), and numbers (0-9) are allowed", but in fact the characters '/' and '.' can be accepted, too.
- In `x/dex/keeper/msg_server_swap_exact_out.go:75, 78, 96`, the comments refer to "*minimum amount out*" and "*above the minimum*," while they should instead refer to "*maximum amount in*" and "*below the maximum,*" respectively.
- In `x/factory/README.md:113`, the message is referred to as `MsgChangeAdmin`; it should read `MsgSetDenomMetadata`.
- In `x/factory/README.md:122`, the description states "Modify AuthorityMetadata state entry to change the admin of the denom", while it should read "Modify the metadata of the denom".

- In `wasmbinding/query_plugin.go:206-213`, generic `wasmvmtypes.Unknown` errors are returned instead of the actual error. This obscures actual error causes and reduces the maintainability of the codebase.

Recommendation

We recommend correcting and actualizing all error messages and comments.

Status: Resolved

41. Insufficient validation in `ValidateBasic` may lead to panic

Severity: Informational

In `x/dex/types/messages_create_pool.go:33-41`, the `ValidateBasic` function on `MsgCreatePool` only checks that both coins (`msg.Quote` and `msg.Base`) are valid.

However, it does not validate that they are not the same. If the same denom is provided in a `MsgCreatePool` message, the `CreatePool` function will panic in `x/dex/keeper/msg_server_create_pool.go:42` as `sdk.NewCoins` will panic, resulting in poor error handling.

Similarly, in `x/dex/types/message_add_liquidity.go:39-45`, the `ValidateBasic` function on `MsgAddLiquidity` does not validate that both coins are not the same. It could result in the `AddLiquidity` function to panic in `x/dex/keeper/msg_server_add_liquidity.go:45`.

While a panic in `ValidateBasic` would not crash the node due to the SDK's panic recovery mechanisms, relying on panic recovery is not a recommended practice due to the following reasons:

- Missed error handling: panics are not subject to type-checking, so it is easier to overlook error-handling mechanisms.
- Unrecoverable failures: if a panic signals a recoverable error and is not properly handled, the program may terminate unnecessarily.
- Reduced readability and debugging complexity: codebases that rely on panics are harder to follow and debug, increasing maintenance overhead.
- Performance overhead: unwinding the stack decreases program performance.

Consequently, relying on panics as the primary error-handling mechanism increases the difficulty of ensuring codebase safety and maintainability.

Recommendation

We recommend checking in `ValidateBasic` that both denoms are different and returning an error if they are the same, to ensure graceful error handling.

Status: Resolved

42. Possibility to bypass fees by swapping very low amounts

Severity: Informational

In `x/dex/keeper/msg_server_swap_exact_in.go:212`, the `fee` value is calculated by multiplying the amount of tokens by `feeRatePerHundredThousand`, followed by dividing by `scalingFactor`.

As a consequence, very low amounts of tokens do not incur fees. For instance, the 0.5% fee would be rounded down to 0 when the user swaps 199 tokens.

Under normal circumstances, this does not pose any risk. However, it could be exploited under either of the two very specific conditions:

- Hundreds of tokens can still have some value for high-price tokens, e.g., `nBTC` where 199 tokens (satoshis) are worth approximately $0.2. In this case, it is only a mild DoS factor, because a malicious party would still need to cover gas fees
- Factory tokens could have `decimals` set to as low as 0, which could render even 199 tokens as something having external value. It could be $199 if someone decided to implement their own stablecoin as a factory token. A malicious user could swap any amount of such tokens for free by splitting them into multiple deals of 199 tokens.

This vulnerability can be exploited only under specific circumstances and requires a mistake from the `pool.Creator` who must preview tokens, so it is reported only with informational severity.

Recommendation

We recommend introducing a minimum fee constant that would prevent the theoretical possibility of using liquidity services for free, as well as implementing validation of the token's decimals during pool creation.

Status: Resolved

43. Potential non-unique pool UIDs due to denom separator collision

Severity: Informational

In `x/dex/keeper/pool_uids.go:53`, a `poolUid` is derived from the denoms of the coins that make up the pool by concatenating both denoms with a `"/"` in between. Since the character `"/"` is allowed in denoms, it is theoretically possible for two different tuples of coins to result in the same UID. For example:

- `coins[0].Denom = "abc/def"` and `coins[1].Denom = "ghi"` results in `poolUidString = "abc/def/ghi"`

- `coins[0].Denom = "abc"` and `coins[1].Denom = "def/ghi"` also results in `poolUidString = "abc/def/ghi"`

Although this scenario is currently theoretical, as it is not possible to create new token denoms with a chosen full denom, it still poses a risk for potential collisions if such a feature is allowed in the future.

Recommendation

We recommend using a separator character that is not allowed in denoms, such as a space. If UIDs need to be made of characters that are all allowed in denoms, we recommend computing a cryptographic hash of the resulting string to prevent potential collisions and ensure unique pool UIDs.

Furthermore, we recommend that the separator be defined as a constant to avoid using a magic string, ensuring that any part of the code can be easily updated if the separator is changed.

Status: Resolved

44. Presence of debugging artifact in production code

Severity: Informational

In `x/dex/keeper/msg_server_create_pool.go:58`, a debugging artifact is still present in the code: `fmt.Println(err)`, which is not a proper logging practice.

Recommendation

We recommend either removing this line or using the appropriate logging mechanism if logging is required.

Status: Resolved

45. Magic numbers used for pool formula validation

Severity: Informational

In `x/dex/types/genesis.go:120`, the validation of the pool's `Formula` field checks that the string length is between `3` and `255`. Additionally, in `x/dex/keeper/msg_server_create_pool.go:294`, the `Formula` field is initialized as the "`constant_product`" string. Using magic numbers like `3` and `255` reduces code clarity and maintainability.

Recommendation

We recommend using an enum with a list of possible formulas. The validator should then check that the formula is one of the allowed values from this enum, improving code clarity and maintainability.

Status: Resolved

46. Unclear “invalid” state variable values in `DefaultGenesis`

Severity: Informational

It has been noticed that in `x/tokenwrapper/types/genesis.go:13`, in `DefaultGenesis`, hardcoded values of `"invalid"` are assigned to `GenesisState` for values such as `ClientId`, `SourcePort`, or `Denom`.

It is not clear what purpose they have, nor does the documentation allow us to identify the reason for such an assignment.

While there is no direct risk associated with this default approach, it may be misleading for developers and code readers.

Recommendation

We recommend that you adjust the documentation or comments describing `DefaultGenesis` to explain why the values are `"invalid"`, or adjust them accordingly to the valid content.

Status: Resolved

47. Potential division by zero during token balance calculation

Severity: Informational

In `x/dex/keeper/msg_server_swap_exact_out.go:204`, there is a potential risk of division by zero when calculating `newBaseTokenBalance := K.Quo(newQuoteTokenBalance)`. This occurs because `newQuoteTokenBalance` is derived from `pool.Coins[quoteCoin].Amount.Sub(outgoingQuote.Amount)` in line `204`, where `outgoingQuote` is obtained from `msg.Outgoing` in line `70`.

If `outgoingQuote.Amount` equals `pool.Coins[quoteCoin].Amount`, it results in a division by zero, leading to panic and improper error handling.

Recommendation

We recommend checking, before calculating `newQuoteTokenBalance` in line `204`, that `outgoingQuote.Amount` is less than `pool.Coins[quoteCoin].Amount` to prevent division by zero and ensure proper error handling.

Status: Resolved

48. Unused `ibcv2TransferStack` initialization may cause confusion

Severity: Informational

In `app/ibc.go:205`, the variable `ibcv2TransferStack` is initialized using `transferv2.NewIBCModule(app.TransferKeeper)`. However, in lines `206-212`, this variable is re-initialized without utilizing the initial `ibcv2TransferStack`. Instead, `transferv2.NewIBCModule(app.TransferKeeper)` is reused as the first parameter of the `NewIBCMiddleware` call.

This results in unnecessary computation and could be confusing due to redundant initialization.

Recommendation

We recommend updating the code in line `207` to use the previously initialized `ibcv2TransferStack`.

Status: Resolved

49. Hardcoded pool UID string construction reduces maintainability

Severity: Informational

In `x/dex/types/genesis.go:81`, the `poolUidString` is instantiated with `pool.Coins[0].Denom + "/" + pool.Coins[1].Denom + "/"`. Similarly, in `x/dex/types/genesis.go:89`, the index of the `poolIndexMap` is computed as `poolUids.PoolId+"/"`.

If the function to derive the pool UID string or the functions to derive the pool UID key and pool key for storage access are modified, this code may become invalid and cause errors, leading to maintainability issues.

Recommendation

We recommend using `poolUidString := PoolUidsKey(GetPoolUidString(pool))` and replacing `poolUids.PoolId+"/"` with `string(PoolKey(poolUids.PoolId))` to ensure consistency and maintainability.

Status: Resolved

50. Error events emitted for non-error cases may cause confusion

Severity: Informational

In `x/tokenwrapper/module/module_ibc.go:170`, `176`, `182`, `279`, `342`, `348`, and `354`, error events are being emitted for scenarios that do not constitute actual errors. This practice could potentially lead to confusion for those who rely on these emitted events, as it may not accurately reflect the result of the transaction.

Recommendation

We recommend updating the code to ensure that error events are emitted exclusively for genuine error cases.

Status: Resolved

51. Incorrect comments and unused types may cause confusion

Severity: Informational

In the file `proto/zigchain/factory/query.proto`, there are several comments and type definitions that are either incorrect or unused, which can cause confusion:

- Line `100`: The comment for `QueryDenomByAdminRequest` incorrectly states it is for the `Query/DenomsByAdmin` RPC method.
- Line `106`: The comment for `QueryAllDenomRequest` incorrectly states it is for the `Query/DenomAll` RPC method.
- Line `111`: The comment for `QueryAllDenomResponse` incorrectly states it is for the `Query/DenomAll` RPC method.
- Line `117`: The comment for `QueryDenomByAdminResponse` incorrectly states it is for the `Query/DenomsByAdmin` RPC method.
- Lines `123-133`: The types `QueryAllDenomsByAdminRequest` and `QueryAllDenomsByAdminResponse` are defined but not used and should be removed.
- Line `135`: The comment for `QueryGetDenomAuthRequest` incorrectly states it is for the `Query/GetDenomAuth` RPC method.
- Line `140`: The comment for `QueryGetDenomAuthResponse` incorrectly states it is for the `Query/GetDenomAuth` RPC method.

- Line 145: The comment for `QueryAllDenomAuthRequest` incorrectly states it is for the `Query/ListDenomAuth` RPC method.
- Line 150: The comment for `QueryAllDenomAuthResponse` incorrectly states it is for the `Query/ListDenomAuth` RPC method.

Recommendation

We recommend correcting the comments to accurately reflect the associated RPC methods for each request and response type.

Additionally, we recommend removing the unused types `QueryAllDenomsByAdminRequest` and `QueryAllDenomsByAdminResponse` to maintain clean and efficient code.

Status: Resolved

52. Unused functions and errors implemented in the codebase

Severity: Informational

There are several functions and errors in the module that are not used in their logic, as well as by any tests or other code fragments. Such code significantly reduces its quality, readability, and in the worst circumstances, may mean the lack of implementation of appropriate handlers that were foreseen during the conceptual creation of the solution.

These are:

- The `SendFromDexToPool` function in `x/dex/keeper/pool.go:309`
- The `SendFromPoolToDex` function in `x/dex/keeper/pool.go:437`
- The `GetPoolBalance` function in `x/dex/keeper/pool.go:491`
- The `ErrInvalidVersion` error in `x/dex/types/errors.go:13`

Recommendation

We recommend verifying whether the above functions and errors should not be used in the solution logic and removing them otherwise.

Status: Resolved

53. Incorrect use of `SignerCheck` may cause confusion

Severity: Informational

In `x/factory/types/messages_denom_update_denom_auth.go:46` and `57`, the `msg.BankAdmin` and `msg.MetadataAdmin` values of `MsgUpdateDenomAuth` are validated using the `validators.SignerCheck` function.

However, this function returns an error message indicating that the signer is invalid, which can be misleading since the `msg.BankAdmin` and `msg.MetadataAdmin` are not the signers in this context.

Recommendation

We recommend using `validators.AddressCheck` instead of `validators.SignerCheck`. This function performs the same validation but provides a more appropriate error message, enhancing clarity and accuracy in error reporting.

Status: Resolved

54. Incorrect base and quote values returned after liquidity removal

Severity: Informational

In `x/dex/keeper/msg_server_remove_liquidity.go:120`, after the successful liquidity removal, the message of `MsgRemoveLiquidityResponse` type is returned, containing two fields - `Base` and `Quote`. For both of them, pool reserves are assigned according to the coins used.

This is incorrect, as based on the liquidity removal logic, the `coinsOut` values should be returned to the user, informing them how many tokens were returned through the above-mentioned operation.

Recommendation

We recommend changing the `pool.Coins[0]` and `pool.Coins[1]` values to the `coinsOut[0]` and `coinsOut[1]` respectively.

Status: Resolved

55. Pool removal message is not implemented

Severity: Informational

In the `x/dex/keeper/pool.go:55`, the `RemovePool` function is implemented, with logic allowing for the pool's removal, if needed. However, this keeper function has no `msg_server` handler available, making pool removal impossible.

This is problematic because such behavior may have been planned, but during development, the functionality was forgotten.

Recommendation

We recommend that if the logic expects that pool removal should be possible - such functionality should be implemented. If not, the `RemovePool` function can be removed from the codebase.

Status: Resolved

56. Miscellaneous comments

Severity: Informational

Miscellaneous recommendations can be found below.

Recommendation

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

- In `x/dex/types/params.go:88`, the error message contains a typo: "less then" should be "less than".
- In `x/factory/keeper/msg_server_create_denom.go:126`, the event emitted does not contain any information about the fee charged for creating the denom.
- In `x/tokenwrapper/module/module_ibc.go:326`, the `data.Denom` becomes `moduleDenom`, if it is equal to the hardcoded `constants.BondDenom`. However, this change is performed before confirmation that the transaction is related to the Axelar transfer. In fact, it should be done after such a check.
- The documentation states that "*When tokens are sent back to Axelar: IBC vouchers are released and sent to the Axelar recipient address*", while in fact, vouchers are burned during the process.
- In the `x/dex/keeper/msg_server_create_pool.go:293`, there is a comment after the `Fee` field saying that the fee is set to 0.5%. That is not true, as the fee rate will be set during the genesis, and can be changed later by the governance.

Status: Resolved

Appendix A

1. Command to compute test coverage excluding proto-generated files

To compute the test coverage accurately without including code generated by protoc-gen-gogo and protoc-gen-grpc-gateway, file extensions that end with `.pb.go` or `.pb.gw.go` need to be excluded.

The following command is executed to compute the test coverage:

```
go test -coverprofile=coverage.out ./... && grep -vE ".pb.go|.pb.gw.go"
coverage.out | go tool cover -func=/dev/stdin | grep total | awk '{print $3}'
```

Appendix B: Test Cases

1. Test case for "Incorrect token amount can be added to the pool reserves"

```
func TestAddWrongAmount(t *testing.T) {
	// Test case: add liquidity to a pool

	// create a mock controller
	ctrl := gomock.NewController(t)

	// call this when the test is done, or exits prematurely
	defer ctrl.Finish() // assert that all expectations are met

	// create a sample signer address
	creator := sample.AccAddress()

	// Bech32 address for verifying method calls
	signer := sdk.MustAccAddressFromBech32(creator)

	// create all coins required for the pool creation
	createPoolBase := sample.Coin("abc", 100)
	createPoolQuote := sample.Coin("usdt", 50)
	createPoolCreationFee := sample.Coin("uzig", 100000000)
	// square root of a * b
	createPoolExpectedLPCoin := sample.Coin("zp1", 70)

	// how much we will add to the pool of abc and usdt coins
	addLiquidityBase := sample.Coin("abc", 20)
	addLiquidityQuote := sample.Coin("usdt", 10)
	// how much we will mint LP token as a result of adding liquidity
	addLiquidityExpectedLPCoin := sample.Coin("zp1", 14)

	// create dex keeper with pool mock
	server, dexKeeper, ctx, pool, poolAccount, bankKeeper, accountKeeper :=
common.ServerDexKeeperWithPoolMock(
	t,
	ctrl,
	signer,
	createPoolBase,
	createPoolQuote,
	createPoolCreationFee,
	createPoolExpectedLPCoin,
	)

	// extract the pool address from the pool account
	poolAddress := poolAccount.GetAddress()
```

```
        // code will check if the signer has the required balance of abc
        // HasBalance(context.Context, sdk.AccAddress, sdk.Coin) bool
        bankKeeper.
        EXPECT().
        HasBalance(gomock.Any(), signer, addLiquidityBase).
        Return(true).
        Times(2)

        // code will check if the signer has the required balance of usdt
        bankKeeper.
        EXPECT().
        HasBalance(gomock.Any(), signer, addLiquidityQuote).
        Return(true).
        Times(2)

        accountKeeper.
        EXPECT().
        GetAccount(ctx, poolAddress).
        Return(poolAccount).
        Times(1)

        bankKeeper.
        EXPECT().
        SendCoins(gomock.Any(), signer, poolAddress,
sdk.NewCoins(addLiquidityBase, addLiquidityQuote)).
        Return(nil).
        Times(1)

        // code will mint new lp tokens, so we can send them to the signer in the
next step
        bankKeeper.
        EXPECT().
        MintCoins(gomock.Any(), types.ModuleName,
sdk.NewCoins(addLiquidityExpectedLPCoin)).
        Return(nil).
        Times(1)

        // SendCoinsFromModuleToAccount(context.Context, string, sdk.AccAddress,
sdk.Coins) error
        // code will send the minted LP token from dex module to the signer
        bankKeeper.
        EXPECT().
        SendCoinsFromModuleToAccount(gomock.Any(), types.ModuleName, signer,
sdk.NewCoins(addLiquidityExpectedLPCoin)).
        Return(nil).
        Times(1)

        // create an "add liquidity" message
        txAddLiquidityMsg := &types.MsgAddLiquidity{
        Creator: creator,
```

```
	PoolId:  pool.PoolId,
	Base:  addLiquidityQuote,
	Quote:   addLiquidityBase,
	}

	// make rpc call to add liquidity
	resp, err := server.AddLiquidity(ctx, txAddLiquidityMsg)

	// make sure there is no error
	require.NoError(t, err)

	// check response
	require.Equal(t, addLiquidityExpectedLPCoin, resp.Lptoken)

	poolId := pool.PoolId
	// get the pool from the keeper, so we can compare new state to expected
values
	pool, found := dexKeeper.GetPool(ctx,
	poolId,
	)

	// make sure the pool was found
	require.True(t, found)

	// new abc is old abc + added abc
	newPoolBase := createPoolBase.Add(addLiquidityBase)
	// new usdt is old usdt + added usdt
	newPoolQuote := createPoolQuote.Add(addLiquidityQuote)
	// new LP token is old LP token + minted LP token
	newPoolLPToken :=
createPoolExpectedLPCoin.Add(addLiquidityExpectedLPCoin)

	// quick pool check
	common.PoolCheck(
	t,
	pool,
	poolId,
	creator,
	newPoolBase,
	newPoolQuote,
	newPoolLPToken,
	poolAddress,
	)

}
```