Audit Report

Cosmos SDK

v1.0

January 23, 2024

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 Informal Systems Inc to perform a security audit of selected changes to Cosmos SDK that will be used within Gaia between v0.45.16-ics-lsm and v0.47.4.

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/cosmos/cosmos-sdk		
Commit	`c9144f02dda85d2bbf09115a134ba7f81c9a5052`		
Scope	**Component**	**Paths in scope**	**Type of audit**
	baseapp	`baseapp`	Full audit
	post handlers and post decorators	`type/handler.go`, `baseapp`	Full audit
	store	`store`	Full audit
	gov module	`x/gov`	Full audit
	auth module	`x/auth`	Full audit
	authz module	`x/authz`	Full audit
	consensus module	`x/consensus`	Full audit
	upgrade module	`x/upgrade`	Full audit
	math	`math`	Full audit

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

Cosmos SDK is an open-source framework for building multi-asset public Proof-of-Stake (PoS) blockchains, like the Cosmos Hub, as well as permissioned Proof-of-Authority (PoA) blockchains. Blockchains built with Cosmos SDK are generally referred to as application-specific blockchains.

The goal of Cosmos SDK is to allow developers to easily create custom blockchains from scratch that can natively interoperate with other blockchains. Cosmos SDK is envisioned to be a npm-like framework to build secure blockchain applications on top of CometBFT. SDK-based blockchains are built out of composable modules, most of which are open-source and readily available for any developer to use. Anyone can create a module for Cosmos SDK, and integrating already-built modules is as simple as importing them. What's more, Cosmos SDK is a capabilities-based system that allows developers to better reason about the security of interactions between modules.

This audit covers selected components of Cosmos SDK on a release candidate for v0.47.x, see above for details.

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	-
Code readability and clarity	Medium-High	-
Level of documentation	High	The included documentation is comprehensive and detailed.
Test coverage	Medium-High	The test coverage for the audited components is: • `baseapp`: 70.2% • `math`: 81.7% • `store`: 76.9% • `x/auth`: 62.1% • `x/authz`: 75.2% • `x/consensus`: 85.2% • `x/gov`: 60.1% • `x/upgrade`: 68.2%

Summary of Findings

No	Description	Severity	Status
1	Attackers can perform an inexpensive DoS attack by spamming `Deposit` transactions	Critical	Resolved
2	Attackers can steal funds using the `tip postHandler`	Critical	Resolved
3	Attackers can perform an inexpensive DoS attack by spamming `Grant` transactions	Critical	Resolved
4	The `postHandler` is called with a context that does not embed transaction updates and the current `mode`	Major	Resolved
5	The `PrepareProposal` function could silently fail due to deferred panic handling	Major	Acknowledged
6	`PrepareProposalHandler` does not adhere to ABCI++ interface specification	Major	Resolved
7	Inefficiencies in block capacity validation within the `PrepareProposal` function	Major	Acknowledged
8	Failing `AfterProposalFailedMinDeposit` and `AfterProposalVotingPeriodEnded GovHooks` could lead to inconsistent state	Major	Resolved
9	Attackers depositing extra coins for a proposal could prevent other deposits	Major	Resolved
10	Attackers can perform an inexpensive DoS attack by spamming `Vote` transactions	Major	Acknowledged
11	The pointer to `big.Int` is copied to a new instance of `Int` and `Uint` types	Major	Resolved
12	Missing validation for keys and values in the `dbadapter`	Minor	Resolved
13	Unhandled errors and variables	Minor	Resolved
14	Lack of `MinInitialDepositRatio` parameter validation in gov module's `ValidateGenesis` function	Minor	Resolved
15	Incomplete validation in gov module's `ValidateGenesis` function	Minor	Resolved

16	Proposals executing a large number of messages could be used to perform a DoS attack on the chain	Minor	Acknowledged
17	Gov module's invariant function could be used to perform a DoS attack on nodes using invariant checks	Minor	Resolved
18	Usage of vulnerable library in the `DownloadUpgrade` function	Minor	Partially Resolved
19	Files are created with non-restrictive permissions	Minor	Partially Resolved
20	`UpgradeInfo` data read from disk lacks validation	Minor	Resolved
21	Missing implementation of `ValidateGenesis` method for `authz` module	Minor	Resolved
22	Missing implementation of the `ValidateBasic` method for `GenericAuthorization`	Minor	Resolved
23	`RelativePow` returns an incorrect value when it raises 0 to 0	Minor	Resolved
24	`MsgExec` messages bypass message level decorators	Minor	Acknowledged
25	Resolve stale `TODO` comments in the codebase	Informational	Acknowledged
26	Use of deprecated errors package	Informational	Acknowledged
27	No default clause in the switch statement of the gov module's `InitGenesis`	Informational	Acknowledged
28	The `Id` value is returned instead of `AccountId` in case of an error in the auth module's query	Informational	Resolved
29	Unnecessary input validation in `NewMsgCreatePeriodicVestingAccountCmd`	Informational	Acknowledged
30	Potentially incorrect telemetry vote counter data	Informational	Resolved
31	Tip decorator `postHandler` incorrectly defined as an `anteHandler`	Informational	Acknowledged
32	Redundant declaration of `TipTx`	Informational	Resolved
33	Deprecated fields in the gov module's `GenesisState`	Informational	Acknowledged
34	Misleading comments	Informational	Resolved

35	The `AfterProposalFailedMinDeposit` and `AfterProposalVotingPeriodEnded` `GovHooks` should be not resource intensive	Informational	Acknowledged
36	`UpdateParams` messages with incomplete `ConsensusParams` lead the to a panic	Informational	Resolved
37	`CancelUpgrade` transactions silently fail	Informational	Resolved

Detailed Findings

1. Attackers can perform an inexpensive DoS attack by spamming `Deposit` transactions

Severity: Critical

In `x/gov/keeper/deposit.go:166-180`, the functions `RefundAndDeleteDeposits` and `DeleteAndBurnDeposits` are invoked within the `EndBlocker` function of the `gov` module.

However, since these functions perform unbounded loops through all `Deposits` associated with proposals that have expired, an attacker could create a large number of accounts and send from each of them a `Deposit` transaction to attack the mentioned unbounded iteration and perform a DoS attack on the chain.

Even worse, an attacker could send `Deposit` transactions containing more than one `Coin`. In such cases, the `SendCoinsFromModuleToAccount` function would be triggered to iterate through all the `Coins` for each `Deposit` item retrieved from storage.

Note that the gas cost associated with a `Deposit` transaction can be as low as the minimum gas price a validator configures multiplied by the provided gas. In the case of the Cosmos Hub the recommended gas price is 0.0025 uATOM and the associated transaction gas cost is 0.00039705 ATOM. Attackers might opt for depositing `0uatom` or utilize worthless coins generated through a `tokenfactory` module, while paying only 397.05 ATOM on the Cosmos Hub for gas to execute 1,000,000 transactions and generate an equivalent number of `Deposit` records. This is exacerbated by the fact that the deposit functionality does not prevent deposit amounts of `0uatom`. This attack might be even cheaper on other Cosmos SDK chains, depending on the configured minimum gas price and the value of the native coin.

Consequently, the execution of the `EndBlocker` will take more time and resources than expected causing the chain to slow down or, in the worst case, even halt.

A test case showcasing this issue is provided in the Appendix 1.

Recommendation

We recommend allowing deposits only from coins whitelisted in `MinDeposit` and defining a minimum deposit for deposits.

Status: Resolved

The issue has been fixed in v0.50 but it is still present in v0.47.

2. Attackers can steal funds using the `tip postHandler`

Severity: Critical

In `x/auth/posthandler/tips.go:39-58`, the `tip Posthandler` allows users to specify a `tipper` account that pays a tip to the `feePayer`.

However, since there is no validation on who can charge the `tipper`, an attacker could specify any `tipper` account in order to steal funds from them that are sent to the `feePayer`.

Recommendation

We recommend implementing functionality to whitelist addresses that can request the `tipper` to pay the tip.

Status: Resolved

The issue has been fixed in v0.50 at commit `6715b5afb59f12627ec5267fd44143444a884dc6`. The issue is still present in v0.47 and the client has communicated to not use the feature.

3. Attackers can perform an inexpensive DoS attack by spamming `Grant` transactions

Severity: Critical

In `x/authz/keeper/keeper.go:378-403`, the `DequeueAndDeleteExpiredGrants` function is executed during the `authz` module's `EndBlocker` function to remove all expired grants from the store.

However, to do so, an iteration occurs over all the expired grants in the `GrantQueue`, and for each of the retrieved `GrantQueueItems`, another iteration takes place over all the items in the associated `MsgTypeUrls` slice.

Note that the gas cost associated with a `Grant` transaction can be as low as the minimum gas price a validator configures multiplied by the provided gas. In the case of the Cosmos Hub the recommended gas price is 0.0025 uATOM and the associated transaction gas cost is 0.00017513 ATOM. Attackers might pay only 175.13 ATOM on the Cosmos Hub for gas to execute 1,000,000 transactions and generate an equivalent number of `Grant` records. This is exacerbated by the fact that attackers can target a specific block to execute all the iterations by specifying the `expiration` parameter in the `Grant`.

This attack might be even cheaper on other Cosmos SDK chains, depending on the configured minimum gas price and the value of the native coin.

If such an attack is executed, the `EndBlocker` function can consume a significant time and resources causing the chain to slow down or, in the worst case, even halt.

Recommendation

We recommend introducing a throttling mechanism for removing expired grants in smaller, manageable batches. This would not lead to expired and not yet removed grants becoming executable, since the `Exec` transaction handler is not executing messages if the `Grant` has expired.

Status: Resolved

The issue has been fixed in v0.51 but it is still present in v0.47.

4. The `postHandler` is called with a context that does not embed transaction updates and the current `mode`

Severity: Major

In `baseapp/baseapp.go:735-760`, within the `runTx` function, the `postHandler` is invoked following the execution of messages performed by the `runMsgCtx`.

However, if the provided `mode` does not align with either `runTxModeDeliver` or `runTxModeSimulate`, the `postHandler` is executed solely with the updates from the `anteHandler`. This scenario introduces a potential error risk, especially if the `postHandler` contains logic that relies on the updated state, which is its intended purpose.

For instance, the `tip postHandler` may encounter difficulties in verifying whether the tipper possesses sufficient coins to facilitate the transfer in `x/auth/posthandler/tips.go:39-58` if the `mode` is different from `runTxModeDeliver`.

Recommendation

We recommend implementing a method on the `context` that empowers developers to retrieve the current `mode` in the `postHandler` as well as documenting the necessity to manage different `modes` within the `postHandler`.

Additionally, we recommend skipping the execution of the `tip postHandler` if the mode is not `runTxModeDeliver` or `runTxModeSimulate`.

Status: Resolved

The issue has been fixed in v0.50 with the implementation of the `execMode` context's method that allows developers to retrieve the current `mode` in the `postHandler`. The issue is still present in v0.47.

5. The `PrepareProposal` function could silently fail due to deferred panic handling

Severity: Major

In `baseapp/abci.go:275-286`, the `PrepareProposal` function includes a deferred function designed to handle panics and returning `abci.ResponsePrepareProposal{Txs: req.Txs}`.

This mechanism leads to the execution of transactions even if the `PrepareProposal` function panics.

If a panic occurs within the `prepareProposal` handler, it typically indicates a broken invariant, and allowing the proposer to proceed with sending transactions in the order specified by the CometBFT's mempool could be problematic.

Consider, for instance, a blockchain where security relies on the execution order of transactions or on Blockbuster's lanes to prioritize specific types of messages. In such a scenario, the presence of this deferred function could potentially compromise the security and integrity of the protocol.

Recommendation

We recommend returning an empty slice instead of `req.Txs` to prevent an unintended execution of transactions.

Status: Acknowledged

The client states that this approach is adopted to maintain the continuity of the blockchain. The `PrepareProposal` function issues an error in the event a panic happens and leverages the input transactions provided by CometBFT to generate a block, bypassing the application's mempool and block proposal mechanism. The intended workflow is that node operators or appchain developers will detect the error messages in the logs and promptly initiate corrective actions without causing a complete cessation of the chain. If the stakeholders of the blockchain perceive it as necessary, possibly due to the critical nature of block construction in their operations, they have the option to trigger an emergency halt.

An alternative approach the team has considered involves allowing the blockchain to panic during the execution of `PrepareProposal`. This strategy prioritizes correctness in block proposal over continuous operation, which may align with the preferences of certain developers. Therefore, the client remains open to revisiting this issue, acknowledging that the severity of the situation is contingent upon its significance to specific appchains.

6. `PrepareProposalHandler` does not adhere to ABCI++ interface specification

Severity: Major

In the `PrepareProposalHandler` function in `baseapp/baseapp.go:957`, if there is no mempool set for the `DefaultProposalHandler`, then the function will simply return the `req.Txs` specified in the request. This is problematic as it may potentially return a list of transactions that exceeds `RequestPrepareProposal.max_tx_bytes`. Additionally, the function will also return an unchecked list of transactions if the defer function is triggered by a panic in the `PrepareProposal` function. In both cases, it will directly return the `req.Txs` without checking that the list does not exceed the maximum bytes limit. This does not adhere to the CometBFT specification:

"*CometBFT MAY include a list of transactions in RequestPrepareProposal.txs whose total size in bytes exceeds RequestPrepareProposal.max_tx_bytes. Therefore, if the size of RequestPrepareProposal.txs is greater than RequestPrepareProposal.max_tx_bytes, the Application MUST remove transactions to ensure that the RequestPrepareProposal.max_tx_bytes limit is respected by those transactions returned in ResponsePrepareProposal.txs .*"

Recommendation

We recommend ensuring that any list of transactions returned in the `PrepareProposalHandler` is checked to ensure that it does not exceed `RequestPrepareProposal.max_tx_bytes`.

Status: Resolved

7. Inefficiencies in block capacity validation within the `PrepareProposal` function

Severity: Major

In the `PrepareProposalHandler` function in `baseapp/baseapp.go:952`, there is a loop that iterates over transactions in the mempool and adds them to the response transaction slice until the total size of the transactions in the proposal reaches or exceeds the maximum size defined as `req.MaxTxBytes`.

In line `997`, a transaction is selected only if its addition does not exceed `req.MaxTxBytes`. If the sum of `txSize` and `totalTxBytes` is greater than `req.MaxTxBytes`, the iteration will continue without adding the transaction to `selectedTxs`. Finally, in line `1014` there is a condition to break the loop if the `totalTxBytes >= req.MaxTxBytes` condition is met.

This creates a situation where the iteration will not break as expected. Instead, it will loop through the entire mempool even if there is no remaining transaction that can satisfy the

condition in line 997. Under normal circumstances, this condition tends to be evaluated as false, except for rare instances where these values happen to be exactly equal.

Furthermore, under certain conditions, this can potentially be exploited by an attacker to force the block proposer to timeout. When there are only a few bytes remaining before reaching the capacity limit, the code continues to iterate through all transactions and execute them within a branched context. This behavior introduces unnecessary computational overhead and significantly impacts performance. Depending on the maximum number of transactions allowed in the mempool, this could result in an excessive amount of iteration. Note that during each iteration, the ante decorators including validate basic will be executed which in combination could result in a large computational resource demand.

Recommendation

We recommend modifying the loop in the `PrepareProposalHandler` function to break iteration early once it is determined that no more transactions can be added without exceeding `req.MaxTxBytes`.

For example, a configurable parameter could be created for a minimum transaction size. When this threshold is reached, the iteration should be broken as it is not expected that any transactions remaining in the mempool will satisfy the remaining byte size limit.

A similar approach should also be applied to gas calculations.

Status: Acknowledged

The client states that this function has been modified to use `TxSelector` in newer versions.

8. Failing `AfterProposalFailedMinDeposit` and `AfterProposalVotingPeriodEnded` `GovHooks` could lead to inconsistent state

Severity: Major

In `x/gov/abci.go:33` and `x/gov/abci.go:121`, in the `EndBlocker` function, the `AfterProposalFailedMinDeposit` and `AfterProposalVotingPeriodEnded` `GovHooks` are triggered.

However, since those functions run on the actual `context` rather than a `cacheContext`, if the hook encounters an error, any modifications made to the store will not be discarded, potentially leading to an inconsistent state.

Recommendation

We recommend using a `cacheContext` for `AfterProposalFailedMinDeposit` and `AfterProposalVotingPeriodEnded` hooks.

Status: Resolved

The issue has been fixed in v0.50 but it is still present in v0.47.

9. Attackers depositing extra coins for a proposal could prevent other deposits

Severity: Major

In `x/gov/keeper/deposit.go:185-202`, the `Deposit` function does not automatically refund any extra coins that a user sends when depositing into a proposal. Instead, these extra coins are added to the `TotalDeposit` coin slice.

An attacker could leverage this behavior and mint a large number of coins and use them to grow the `TotalDeposit` coin slice. For instance, the attacker could mint coins with a `tokenfactory` module on another chain and then move them to the Cosmos Hub through IBC.

Consequently, since the `AddDeposit` function in `x/gov/keeper/deposit.go:127-147` iterates deposited coins multiple times during calls of the `Sort`, `Add`, and `IsAllGTE` functions, if the `TotalDeposit` slice is sufficiently large, the execution of `Deposit` messages will get more expensive for all other depositors, and in the worst case even run out of gas, which inhibits legit depositors from depositing coins.

A test case showcasing the issue is provided in Appendix 2.

Recommendation

We recommend limiting deposits to coins that are defined in `params.MinDeposit`.

Status: Resolved

The issue has been fixed in v0.50 but it is still present in v0.47.

10. Attackers can perform an inexpensive DoS attack by spamming `Vote` transactions

Severity: Major

In `x/gov/keeper/tally.go:14-127`, the `Tally` function is executed in the `EndBlocker` in order to compute the result of expired proposals.

However, since this function performs an unbonded iteration through all the submitted votes and for each of them iterates through all the voter's delegations, as well as through all bonded validators, a significant number of votes will slow down the execution of the `EndBlocker` causing the chain to slow down or in the worst case even halt.

An attacker could create a large number of accounts and send a `Vote` transaction from each of them in order to attack the mentioned unbounded iteration and hence perform a DoS attack on the chain.

It's noteworthy that a single vote cost is as low as the minimum gas price a validator configures multiplied by the provided gas.In the case of the Cosmos Hub the recommended gas price is 0.0025 uATOM and the associated transaction gas cost is 0.000206105 ATOM. This implies that sending 1,000,000 votes would require only 206.105 ATOM.

Recommendation

We recommend not allowing accounts that do not possess a minimum amount of staked coins to vote.

Status: Acknowledged

The client states this issue will be fixed in the next `gov` module refactoring. Chains affected by this issue in v0.47 can add a custom ante handler. The issue is tracked in pull request https://github.com/cosmos/cosmos-sdk/pull/18186.

11. The pointer to `big.Int` is copied to a new instance of `Int` and `Uint` types

Severity: Major

In `math/int.go:115` and `math/uint.go:238`, to generate a new type from a `big.Int` instance, the underlying pointer is just copied to a created `math.Int` or `math.Uint` type.

As a result, if the value on that `big.Int` points to is changed, then the value of the new type will be also changed.

Recommendation

We recommend using value semantics instead of pointer semantics.

Status: Resolved

The issue has been fixed in v0.50 through pull requests https://github.com/cosmos/cosmos-sdk/pull/18214 and https://github.com/cosmos/cosmos-sdk/pull/17352. The issue is still present in v0.47.

12. Missing validation for keys and values in the `dbadapter`

Severity: Minor

The documentation provided for the DB type of the `github.com/cometbft/cometbft-db` package states:

"Keys cannot be nil or empty, while values cannot be nil".

However, upon inspecting the implementation in `store/dbadapter/store.go`, it becomes evident that this validation is not performed, with the exception of the `Set` function that checks for `nil` keys.

It is noteworthy that the `memdb` package within `cometbft-db`, on the other hand, consistently enforces validation for both keys and values across all of its functions, as exemplified in https://github.com/cometbft/cometbft-db/blob/a21532dafc74c4752f2a3960474ef6bd1b13f6a0/memdb.go#L93.

Recommendation

We recommend enhancing the validation within the implementation of the wrapper in `store/dbadapter/store.go` to ensure compliance with the documented requirements for keys and values.

Status: Resolved

The issue has been fixed in v0.50 through pull request https://github.com/cosmos/cosmos-sdk/pull/17900. The issue is still present in v0.47.

13. Unhandled errors and variables

Severity: Minor

In the instances specified below, there is a lack of error handling of returned errors:

- `store/iavl/store.go:226`
- `store/iavl/store.go:373`
- `store/listenkv/store.go:147`
- `store/rootmulti/store.go:315`
- `store/rootmulti/store.go:331`
- `store/rootmulti/store.go:1071`
- `store/rootmulti/store.go:1171`
- `store/rootmulti/store.go:1180`
- `store/streaming/constructor.go:166`
- `store/streaming/constructor.go:179`
- `store/streaming/constructor.go:189`
- `store/tracekv/store.go:198`

- `x/auth/tx/builder.go:445`
- `x/auth/tx/service.go:357`
- `x/gov/genesis.go:15`

Although we could not identify direct security implications of this issue, we classify it as minor since it could have unintended consequences in future extensions of the codebase.

Recommendation

We recommend implementing proper error checking and handling procedures for the returned errors in these instances or providing clear comments explaining why the errors are intentionally ignored.

Status: Resolved

The client states that in v0.50, the `errcheck` linter has been added and all the relevant errors have been fixed. The issue is still present in v0.47.

14. Lack of `MinInitialDepositRatio` parameter validation in gov module's `ValidateGenesis` function

Severity: Minor

In `x/gov/types/v1/params.go:89`, during the execution of the gov module's `ValidateGenesis` function, the `MinInitialDepositRatio` parameter is not validated.

Since this parameter represents a percentage it should be in the `[0, 1]` range.

Recommendation

We recommend validating the `MinInitialDepositRatio` during the execution of the `ValidateGenesis` function.

Status: Resolved

15. Incomplete validation in gov module's `ValidateGenesis` function

Severity: Minor

In `x/gov/module.go:79-86`, the `ValidateGenesis` function within the gov module validates the `Params` structure, but it does not perform validation or deduplication for all of its fields.

Specifically, the `Deposits`, `Votes`, and `Proposals` slices are not validated and deduplicated.

Recommendation

We recommend validating and deduplicating the `Deposits`, `Votes`, and `Proposals` slices.

Status: Resolved

16. Proposals executing a large number of messages could be used to perform a DoS attack on the chain

Severity: Minor

In `x/gov/abci.go:76-90`, the gov module's `EndBlocker` function executes the associated messages for each approved proposal.

Since there is no gas limit for that message execution, a proposal containing numerous or computationally intensive messages could lead to a slowdown of block production, eventually halting the chain.

We classify this issue as minor since this attack is only possible if a malicious proposal passes.

Recommendation

We recommend defining a gas limit for the message execution of a proposal.

Status: Acknowledged

The client states that since the proposal's execution requires governance approval, they do not consider this as a viable DoS attack.

17. Gov module's invariant function could be used to perform a DoS attack on nodes using invariant checks

Severity: Minor

In `x/gov/keeper/invariants.go:27-48`, the `ModuleAccountInvariant` function defines the invariant check for the gov module. When registered, this invariant is executed during the `EndBlocker` of the `crisis` module.

However, since this function performs several unbounded iterations, it could be leveraged to perform a DoS attack on nodes using invariant checks.

Specifically, in line `31`, the function iterates through all the `Deposits` in the `IterateAllDeposits` function. Furthermore, in line `42`, the `IsAllGTE` function iterates through all the `Coins` in the balance of the gov module.

Both iterations could be targeted by attackers to grow the cardinality of `Deposits` or `Coins`.

We classify this issue as minor since invariant checks are optional.

Recommendation

We recommend limiting the number of `Deposit` items stored and not iterating through the module balance by inverting the inequality.

Status: Resolved

The issue has been fixed in v0.50 but it is still present in v0.47.

18. Usage of vulnerable library in the `DownloadUpgrade` function

Severity: Minor

In `x/upgrade/plan/downloader.go:26-44`, the `go-getter` package is used to download files.

This library provides rich functionality and supports many protocols (e.g., http, git, s3, file, gcp), hash functions, including insecure ones (e.g., md5, sha1), and detectors transforming URLs.

Although there are six published CVEs for this library described here and it is documented how to apply proper configurations to enable mitigation mechanisms here, those recommendations are not implemented in the codebase.

The `ValidateIsURLWithChecksum` function in `x/upgrade/plan/downloader.g:134` checks that the input URL contains a `checksum` parameter.

However, the following potentially dangerous URLs are accepted by this function:

- URL containing several `checksum` parameters
- Checksum parameter starting with `sha1`
- URL starting with `file://`
- URL containing two different `checksum` parameters
- URL containing arbitrary port numbers

We classify this issue as minor since the downloading process cannot affect the core functionality of the Cosmos SDK.

Recommendation

We recommend implementing the following security mitigations for the `go-getter` package:

- Explicitly set `insecure: false`
- Disable all detectors
- Enforce timeouts
- Disable symlinks
- Disable `X-Terraform-Get`

We recommend implementing the following verification for input URL:

- Verify that the URL starts with `https`
- Verify that the URL contains `sha256` in the checksum parameter
- Verify that the URL contains exactly one checksum parameter
- If possible, verify that the URL does not contain the explicit port number

Status: Partially Resolved

19. Files are created with non-restrictive permissions

Severity: Minor

In:

- `x/upgrade/keeper/keeper.go:409`,
- `server/util.go:504`, and
- `store/streaming/constructor.go:110`,

directories are created with `os.ModePerm` option.

However, this allows non-privileged users (`others`) to remove files that could cause unintended behaviors.

Recommendation

We recommend creating directories with the read access level only for `others`.

Status: Partially Resolved

20. `UpgradeInfo` data read from disk lacks validation

Severity: Minor

In `x/upgrade/keeper/keeper.go:443`, the upgrade information read from disk is not validated.

However, since the `UpgradeInfo` can be modified by any user of the system due to the `777` privilege mask on the parent directory for this file, it should be validated before being handled.

Recommendation

We recommend validating the data using the existing `ValidateBasic` function.

Status: Resolved

The issue has been fixed in v0.50 through pull request https://github.com/cosmos/cosmos-sdk/pull/18210. The issue is still present in v0.47.

21. Missing implementation of `ValidateGenesis` method for `authz` module

Severity: Minor

In `x/authz/genesis.go:14-17`, the `ValidateGenesis` method for the `authz` module is not implemented and always returns `nil`.

However, since the `GenesisState` contains a slice of `GrantAuthorization` instances, it should be validated.

This validation should encompass the verification of `Granter` and `Grantee` as valid addresses, ensuring that `Authorization` belongs to a recognized type, and confirming that `Expiration` does not reference to a past date.

Recommendation

We recommend implementing `ValidateGenesis` for the `authz` module.

Status: Resolved

The issue has been fixed in v0.50 through pull request https://github.com/cosmos/cosmos-sdk/pull/18042. The issue is still present in v0.47.

22. Missing implementation of the `ValidateBasic` method for `GenericAuthorization`

Severity: Minor

In `x/authz/generic_authorization.go:26-29`, the `ValidateBasic` method for `GenericAuthorization` is not implemented and always returns `nil`.

However, since the `GenericAuthorization` contains a `Msg` field, it should be validated.

This validation should encompass the verification of `Msg` to be a valid message type, ensuring at least that it is a non-empty string.

Recommendation

We recommend implementing the `ValidateBasic` function for `GenericAuthorization`.

Status: Resolved

The issue has been fixed in v0.50 through pull request https://github.com/cosmos/cosmos-sdk/pull/18209. The issue is still present in v0.47.

23. `RelativePow` returns an incorrect value when it raises 0 to 0

Severity: Minor

In `math/uint.go:246`, the `RelativePow(x, n, b)` function raises `x` to the power of `n`, where `x` (and the result, `z`) are scaled by factor `b`.

However, the function always returns `b` if `x` and n are equal to 0, but it should return 1 according to the comments in the function.

Recommendation

We recommend returning 1 instead of returning `b`.

Status: Resolved

The issue has been fixed in v0.50 through pull request https://github.com/cosmos/cosmos-sdk/pull/18211. The issue is still present in v0.47.

24. `MsgExec` messages bypass message level decorators

Severity: Minor

The `DispatchActions` function in `x/authz/keeper/keeper.go:99` is called when an `authz MsgExec` is executed to route each inner message to its corresponding message handler. This method of message routing will bypass message level ante and post decorators though.

While the `MsgExec`'s `ValidateBasic` function loops over the inner messages and calls the `ValidateBasic` function on each of them, it is important to consider that these messages will not have the same pre and post processing as normal Cosmos SDK messages.

Additionally, the `MsgExec` message in `x/authz/keeper/msg_server.go:72` can be called directly by any account, even if they do not have an active grant. This is possible if the user calls `Exec` by themselves.

In the `DispatchActions` function in `x/authz/keeper/keeper.go:99`, the conditional block is bypassed if `granter = grantee`. This will cause the `MsgExec`'s message list to

be directly routed to its message handler. This message route could be used to bypass message level decorators.

While on Cosmos Hub this alternative message routing currently does not have any direct adverse implications, it could be used as an alternative route for an account to perform message execution. If additional decorators are implemented in the future on chains that use the `authz` module, this could be problematic.

For example, in the baseapp level, when `tx.GetMsgs` is called for a transaction module, authors should be aware that this will only get the top level of messages, and if a transaction contains a `MsgExec` message, that message contains inner messages that will not be returned.

Recommendation

We recommend creating an ante decorator in the `authz` module that extracts the inner messages contained in `MsgExec` and then passes them through the other module's chained ante handlers.

Additionally, we recommend returning an error if `MsgExec` is called with `granter = grantee`.

Status: Acknowledged

The client states that this is a current drawback of the system and that it will be resolved in future versions with the server/runtime working group.

In the meantime, the client added a warning to the documentation through pull request https://github.com/cosmos/cosmos-sdk/pull/18709.

25. Resolve stale `TODO` comments in the codebase

Severity: Informational

In the codebase, there are multiple `TODO` comments that are stale and need to be removed to enhance the overall code readability:

- `store/cachekv/store.go:118`
- `store/cachekv/internal/mergeiterator.go:16`
- `store/cachemulti/store.go:148`
- `store/gaskv/store.go:39 and 51`
- `store/prefix/store.go:154`
- `store/rootmulti/store.go:562 and 669`
- `store/trackv/store.go:25`
- `store/types/gas.go:184`

Recommendation

We recommend resolving all `TODO` comments in the codebase.

Status: Acknowledged

26. Use of deprecated errors package

Severity: Informational

The codebase currently relies on the deprecated `sdkerrors.wrap` package, which has migrated to the `cosmossdk.io/errors` module.

Deprecated code not only poses the risk of unfixed bugs but also creates potential compatibility and maintenance challenges once the deprecated code is removed in a future update.

Recommendation

We recommend migrating from the `sdkerrors.wrap` package to the recommended `cosmossdk.io/errors` module to ensure compatibility with future versions.

Status: Acknowledged

27. No default clause in the switch statement of the gov module's `InitGenesis` function

Severity: Informational

The `ProposalStatus` type defined in `proto/cosmos/gov/v1/gov.proto:102` has six possible values.

However, in `x/gov/genesis.go:34-39`, in the `InitGenesis` function, the switch statement is implemented only for two of the six possible `ProposalStatus` values, specifically `StatusDepositPeriod` and `StatusVotingPeriod`.

Additionally, the switch statement does not implement a `default` clause.

Recommendation

We recommend implementing a `default` switch clause.

Status: Acknowledged

The client states that the `default` clause is not strictly needed since the other possible `ProposalStatus` types are not used in that function.

28. The `Id` value is returned instead of `AccountId` in case of an error in the auth module's query

Severity: Informational

In `x/auth/keeper/grpc_query.go:36`, the `AccountAddressByID` function returns an `Id` field equal to zero if an error occurs instead of returning the actual `AccountId`.

Recommendation

We recommend returning `AccountId` in the error message.

Status: Resolved

29. Unnecessary input validation in `NewMsgCreatePeriodicVestingAccountCmd`

Severity: Informational

The `NewMsgCreatePeriodicVestingAccountCmd` function validates the newly created message in `x/auth/vesting/client/cli/tx.go:198`.

However, since all messages are also validated in the `GenerateOrBroadcastTxWithFactory` function in line `43`, this check is redundant.

Recommendation

We recommend removing unnecessary validation from the `NewMsgCreatePeriodicVestingAccountCmd` function.

Status: Acknowledged

30. Potentially incorrect telemetry vote counter data

Severity: Informational

The vote telemetry counter in `x/gov/keeper/msg_server.go:125` does not properly handle the situation where a voter makes multiple votes. Voters can call the message server method `Vote` multiple times to update their vote, but this will incorrectly increment the telemetry counter.

Consequently, the vote counter may increase even though the actual number of votes does not increase. The counter could end up larger than the real vote count.

Recommendation

We recommend modifying the function to ensure that the vote telemetry counter is only incremented if the voter has not previously voted.

Status: Resolved

The issue has been fixed in v0.50 through pull request https://github.com/cosmos/cosmos-sdk/pull/17910. The issue is still present in v0.47.

31. Tip decorator `postHandler` incorrectly defined as an `anteHandler`

Severity: Informational

The `tipDecorator` in `x/auth/posthandler/tips.go:29` is intended to be used as a `postHandler`, but it is incorrectly defined with the `AnteHandler` method.

Recommendation

We recommend updating the `AnteHandler` method in `x/auth/posthandler/tips.go:29` to reflect the fact that the method is intended to be a `PostHandler`.

Status: Acknowledged

The client has removed the tip decorator in v0.50 and communicated to not use this feature in older versions.

32. Redundant declaration of `TipTx`

Severity: Informational

In `x/auth/tx/builder.go:42`, the `wrapper` struct implements the expected set of methods declared in the `TipTx` interface. However, it is already declared in line `39`.

Recommendation

We recommend removing the redundant declaration.

Status: Resolved

33. Deprecated fields in the gov module's `GenesisState`

Severity: Informational

In `x/gov/genesis.go:13-53`, the gov module's `InitGenesis` function does not handle the `DepositParams`, `VotingParams`, and `TallyParams` fields of the `GenesisState` struct since they are deprecated.

Recommendation

We recommend removing deprecated fields from the `GenesisState`.

Status: Acknowledged

The client states that since the recommendation contains a breaking change, it will be not applied in the current version but in a future one.

34. Misleading comments

Severity: Informational

The following misleading comments have been found:

- The governance module's `EndBlocker` function in `x/gov/abci.go:14` contains a misleading comment that states *"EndBlocker called every block, process inflation, update validator set."* However, this function does not process inflation or update the validator set. The comment seems to have been copied from another module's `EndBlocker` implementation.
  The `EndBlocker` function in the governance module processes governance proposals at the end of each block, including tallying votes and executing passed proposals. Having an incorrect comment can lead to confusion for developers and maintainers.
- The `postHandler tipDecorator` struct in `x/auth/posthandler/tips.go:11` has a comment that refers to `anteHandler`, which is misleading.
- In `math/int.go:73`, the comment states "*Int wraps big.Int with a 257 range bound*". However, it is actually 256 bit range bound.
- In `x/authz/client/cli/tx.go:259`, an example for sending from `granter` to `recipient` is missing an `amount` argument.
- `x/upgrade/client/cli/tx.go:38`, the comment says that the `NewCmdSubmitUpgradeProposal` function should be used instead of the deprecated `NewCmdSubmitLegacyUpgradeProposal` function. At the same time, the latter function is missing in the codebase
- `x/upgrade/client/cli/tx.go:108`, the comment says that the `NewCmdSubmitCancelUpgradeProposal` function should be used instead of the deprecated `NewCmdSubmitLegacyCancelUpgradeProposal` function. At the same time, the latter function is missing in the codebase.

Recommendation

We recommend updating the comments to accurately describe the implementation.

Status: Resolved

The issue has been fixed in v0.50 however it is still present in v0.47.

35. The `AfterProposalFailedMinDeposit` and `AfterProposalVotingPeriodEnded` `GovHooks` should not be resource intensive

Severity: Informational

In `x/gov/abci.go:33` and `x/gov/abci.go:121`, in the `EndBlocker` function, the `AfterProposalFailedMinDeposit` and `AfterProposalVotingPeriodEnded` `GovHooks` are executed.

Since those functions are executed in the `EndBlocker` function and there is no control over their execution flow, developers should be aware that a resource-intensive implementation of those hooks, might be abused to execute DoS attacks on the chain. Special care should be taken if any computation is done on user-provided input, for example, iterations that can be influenced by users.

Recommendation

We recommend documenting that the `AfterProposalFailedMinDeposit` and `AfterProposalVotingPeriodEnded` `GovHooks` implementations should be lightweight to not overload the `EndBlocker`.

Status: Acknowledged

The client intends to document this behavior as expressed in issue https://github.com/cosmos/cosmos-sdk/issues/10453#issuecomment-1811992571.

36. `UpdateParams` messages with incomplete `ConsensusParams` lead to a panic

Severity: Informational

In `x/consensus/keeper/msg_server.go:26-41`, the `UpdateParams` transaction enables the governance to update `ConsensusParams`.

However, in case not all the fields of `ConsensusParams` are provided, a segmentation violation in the `ToProtoConsensusParams` is triggered.

This will result in a panic during execution without generating a meaningful error.

A test case showcasing this issue is provided in Appendix 3.

Recommendation

We recommend ensuring that all fields of `ConsensusParams` are properly provided.

Status: Resolved

The issue has been fixed in v0.50 however it is still present in v0.47.

37. `CancelUpgrade` transactions silently fail

Severity: Informational

In `x/upgrade/keeper/msg_server.go:41-51`, the `CancelUpgrade` transaction enables the governance to remove a scheduled `Plan`.

However, in case of the absence of a saved `Plan`, the transaction will not generate an error but will complete successfully, even if it has not performed any actions.

Recommendation

We recommend returning an error in case there is no `Plan` to remove.

Status: Resolved

The issue has been fixed in v0.50 however it is still present in v0.47.

Appendix: Test Cases

1. Test case for **“Attackers can perform an inexpensive DoS attack by spamming Deposit transactions”**

```
func TestGovABCIOverload(t *testing.T) {
	suite := createTestSuite(t)

	app := suite.App
	ctx := app.BaseApp.NewContext(false, tmproto.Header{})

	numDepositors := 1_000_000
	numCoins := 10

	addrs := simtestutil.AddTestAddrs(suite.BankKeeper,
suite.StakingKeeper, ctx, numDepositors, valTokens)

	generatedCoins := sdk.NewCoins(sdk.NewCoin("ufake",
sdk.NewInt(1)))

	for i := 0; i < numCoins; i++ {
		denom := fmt.Sprint("ufake", i)
		generatedCoins = generatedCoins.Add(sdk.NewCoin(denom,
sdk.NewInt(1)))
	}

	header := tmproto.Header{Height: app.LastBlockHeight() + 1}
	app.BeginBlock(abci.RequestBeginBlock{Header: header})

	if err := suite.BankKeeper.MintCoins(ctx, minttypes.ModuleName,
generatedCoins.MulInt(sdk.NewInt(100_000_000_000))); err != nil {
		panic(err)
	}

	for i := 0; i < numDepositors; i++ {

		if err := suite.BankKeeper.SendCoinsFromModuleToAccount(ctx,
minttypes.ModuleName, addrs[i], generatedCoins); err != nil {
			panic(err)
		}
	}

	govMsgSvr := keeper.NewMsgServerImpl(suite.GovKeeper)
```

```
	inactiveQueue :=
suite.GovKeeper.InactiveProposalQueueIterator(ctx,
ctx.BlockHeader().Time)
	require.False(t, inactiveQueue.Valid())
	inactiveQueue.Close()

	newProposalMsg, err := v1.NewMsgSubmitProposal(
		[]sdk.Msg{mkTestLegacyContent(t)},
		sdk.Coins{sdk.NewInt64Coin(sdk.DefaultBondDenom, 5)},
		addrs[0].String(),
		"",
		"Proposal",
		"description of proposal",
	)
	require.NoError(t, err)

	res, err := govMsgSvr.SubmitProposal(sdk.WrapSDKContext(ctx),
newProposalMsg)
	require.NoError(t, err)
	require.NotNil(t, res)

	propId := res.ProposalId

	for i := 1; i < numDepositors; i++ {
		newDepositMsg := v1.NewMsgDeposit(addrs[i], propId,
generatedCoins)

		res, err := govMsgSvr.Deposit(sdk.WrapSDKContext(ctx),
newDepositMsg)
		require.NoError(t, err)
		require.NotNil(t, res)

		deposit, found := suite.GovKeeper.GetDeposit(ctx, propId,
addrs[i])
		require.True(t, found)
		require.Equal(t, generatedCoins,
sdk.NewCoins(deposit.Amount...))
		require.Equal(t, addrs[i].String(), deposit.Depositor)
	}

	inactiveQueue = suite.GovKeeper.InactiveProposalQueueIterator(ctx,
ctx.BlockHeader().Time)
	require.False(t, inactiveQueue.Valid())
	inactiveQueue.Close()
```

```
	newHeader := ctx.BlockHeader()
	newHeader.Time = ctx.BlockHeader().Time.Add(time.Duration(1) *
time.Second)
	ctx = ctx.WithBlockHeader(newHeader)

	inactiveQueue = suite.GovKeeper.InactiveProposalQueueIterator(ctx,
ctx.BlockHeader().Time)
	require.False(t, inactiveQueue.Valid())
	inactiveQueue.Close()

	newHeader = ctx.BlockHeader()
	newHeader.Time =
ctx.BlockHeader().Time.Add(*suite.GovKeeper.GetParams(ctx).MaxDepositPer
iod)
	ctx = ctx.WithBlockHeader(newHeader)

	inactiveQueue = suite.GovKeeper.InactiveProposalQueueIterator(ctx,
ctx.BlockHeader().Time)
	require.True(t, inactiveQueue.Valid())
	inactiveQueue.Close()

	gov.EndBlocker(ctx, suite.GovKeeper)

	inactiveQueue = suite.GovKeeper.InactiveProposalQueueIterator(ctx,
ctx.BlockHeader().Time)
	require.False(t, inactiveQueue.Valid())
	inactiveQueue.Close()
}
```

2. Test case for [“Attackers depositing extra coins for a proposal could prevent other deposits”]

```
func TestDepositsCoinsOverload(t *testing.T) {
	govKeeper, _, bankKeeper, stakingKeeper, _, ctx := setupGovKeeper(t)
	trackMockBalances(bankKeeper)
	TestAddrs := simtestutil.AddTestAddrsIncremental(bankKeeper, stakingKeeper, ctx, 2, sdk.NewInt(10000000))

	tp := TestProposal
	proposal, err := govKeeper.SubmitProposal(ctx, tp, "", "title", "description", TestAddrs[0])
	require.NoError(t, err)
	proposalID := proposal.Id

	generatedCoins := sdk.Coins{}

	for i := 0; i < 10000; i++ {
		denom := fmt.Sprint("ua", i, "tom")
		generatedCoins = generatedCoins.Add(sdk.NewCoin(denom, sdk.NewInt(100)))
	}

	if err := bankKeeper.MintCoins(ctx, minttypes.ModuleName, generatedCoins); err != nil {
		panic(err)
	}

	if err := bankKeeper.SendCoinsFromModuleToAccount(ctx, minttypes.ModuleName, TestAddrs[0], generatedCoins); err != nil {
		panic(err)
	}

	fourStake := generatedCoins.Add(sdk.NewCoin(sdk.DefaultBondDenom, stakingKeeper.TokensFromConsensusPower(ctx, 4)))
	fiveStake := sdk.NewCoins(sdk.NewCoin(sdk.DefaultBondDenom, stakingKeeper.TokensFromConsensusPower(ctx, 5)))

	addr0Initial := bankKeeper.GetAllBalances(ctx, TestAddrs[0])
	addr1Initial := bankKeeper.GetAllBalances(ctx, TestAddrs[1])

	require.True(t, sdk.NewCoins(proposal.TotalDeposit...).IsEqual(sdk.NewCoins()))
```

```
	// Check no deposits at beginning
	deposit, found := govKeeper.GetDeposit(ctx, proposalID,
TestAddrs[1])
	require.False(t, found)
	proposal, ok := govKeeper.GetProposal(ctx, proposalID)
	require.True(t, ok)
	require.Nil(t, proposal.VotingStartTime)

	// Check first deposit
	votingStarted, err := govKeeper.AddDeposit(ctx, proposalID,
TestAddrs[0], fourStake)
	require.NoError(t, err)
	require.False(t, votingStarted)
	deposit, found = govKeeper.GetDeposit(ctx, proposalID,
TestAddrs[0])
	require.True(t, found)
	require.Equal(t, fourStake, sdk.NewCoins(deposit.Amount...))
	require.Equal(t, TestAddrs[0].String(), deposit.Depositor)
	proposal, ok = govKeeper.GetProposal(ctx, proposalID)
	require.True(t, ok)
	require.Equal(t, fourStake,
sdk.NewCoins(proposal.TotalDeposit...))
	require.Equal(t, addr0Initial.Sub(fourStake...),
bankKeeper.GetAllBalances(ctx, TestAddrs[0]))

	// Check a second deposit from same address
	votingStarted, err = govKeeper.AddDeposit(ctx, proposalID,
TestAddrs[1], fiveStake)
	require.NoError(t, err)
	require.False(t, votingStarted)
	deposit, found = govKeeper.GetDeposit(ctx, proposalID,
TestAddrs[1])
	require.True(t, found)
	require.Equal(t, fiveStake, sdk.NewCoins(deposit.Amount...))
	require.Equal(t, TestAddrs[1].String(), deposit.Depositor)

	// Test deposit iterator
	// NOTE order of deposits is determined by the addresses
	deposits := govKeeper.GetAllDeposits(ctx)
	require.Len(t, deposits, 2)
	require.Equal(t, deposits, govKeeper.GetDeposits(ctx, proposalID))
	require.Equal(t, TestAddrs[0].String(), deposits[0].Depositor)
	require.Equal(t, fourStake, sdk.NewCoins(deposits[0].Amount...))
	require.Equal(t, TestAddrs[1].String(), deposits[1].Depositor)
```

```
	require.Equal(t, fiveStake, sdk.NewCoins(deposits[1].Amount...))

	// Test refund Deposits
	deposit, found = govKeeper.GetDeposit(ctx, proposalID,
TestAddrs[1])
	require.True(t, found)
	require.Equal(t, fiveStake, sdk.NewCoins(deposit.Amount...))
	govKeeper.RefundAndDeleteDeposits(ctx, proposalID)
	deposit, found = govKeeper.GetDeposit(ctx, proposalID,
TestAddrs[1])
	require.False(t, found)
	require.Equal(t, addr0Initial, bankKeeper.GetAllBalances(ctx,
TestAddrs[0]))
	require.Equal(t, addr1Initial, bankKeeper.GetAllBalances(ctx,
TestAddrs[1]))

	// Test delete and burn deposits
	proposal, err = govKeeper.SubmitProposal(ctx, tp, "", "title",
"description", TestAddrs[0])
	require.NoError(t, err)
	proposalID = proposal.Id
	_, err = govKeeper.AddDeposit(ctx, proposalID, TestAddrs[0],
fourStake)
	require.NoError(t, err)
	govKeeper.DeleteAndBurnDeposits(ctx, proposalID)
	deposits = govKeeper.GetDeposits(ctx, proposalID)
	require.Len(t, deposits, 0)
	require.Equal(t, addr0Initial.Sub(fourStake...),
bankKeeper.GetAllBalances(ctx, TestAddrs[0]))
}
```

3. Test case for [“`UpdateParams` messages with incomplete `ConsensusParams` lead to a panic”]

```
func (s *KeeperTestSuite) TestUpdateParams() {
	defaultConsensusParams := tmtypes.DefaultConsensusParams().ToProto()
	testCases := []struct {
		name      string
		input     *types.MsgUpdateParams
		expErr    bool
		expErrMsg string
	}{
		{
			name: "invalid  params",
			input: &types.MsgUpdateParams{
				Authority: s.consensusParamsKeeper.GetAuthority(),
				Block:     &tmproto.BlockParams{MaxGas: -10, MaxBytes: -10},
				Validator: nil,
				Evidence:  defaultConsensusParams.Evidence,
			},
			expErr:    true,
			expErrMsg: "this will panic",
		},
	}

	for _, tc := range testCases {
		tc := tc
		s.Run(tc.name, func() {
			s.SetupTest()
			_, err := s.msgServer.UpdateParams(s.ctx, tc.input)
			if tc.expErr {
				s.Require().Error(err)
				s.Require().Contains(err.Error(), tc.expErrMsg)
			} else {
				s.Require().NoError(err)
			}
		})
	}
}
```