Audit Report

Dymension Point 0D

v1.0

March 8, 2024

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

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

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

This audit has been performed by

Oak Security GmbH

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

Introduction

Purpose of This Report

Oak Security has been engaged by Dymension Technologies Ltd to perform a security audit of parts of the Dymension codebase.

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 targets:

Repository	https://github.com/dymensionxyz/dymension
Commit	`32ab176c1452f8739936701d1679667933fcab2a`
Scope	The application in the `app` and `cmd` directories, the `streamer` module in the `x/streamer` directory, and verifying that the `x/delayedack`, `x/denommetadata`, `x/rollapp`, and `x/sequencer` modules are correctly using the `RollapsEnabled` parameter to disable the RollApps functionality temporarily.
Identifier	In this report, all paths pointing to this repository are prefixed with `dymension:`

Fixes verified at commit	`001fe5c4158e658005fa99e056697325cf8873e3` Note that changes to the codebase beyond fixes after the initial audit have not been in scope of our fixes review.

Repository	https://github.com/dymensionxyz/osmosis
Commit	`ea4377e8fe195d416793e4f9e42a0cc8505888d3`
Scope	All changes to the Osmosis fork from `v15.2.0` at commit `4a4a94585872e3196b9c83286979cda11d4889e3` are in scope.
Identifier	In this report, all paths pointing to this repository are prefixed with `dymension-osmosis:`
Fixes verified at commit	`a2ce7ad032ce30b760c9b071aeac79b579142d60` Note that changes to the codebase beyond fixes after the initial audit have not been in scope of our fixes review.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Dymension is a Cosmos SDK chain acting as a network of easily deployable and lightning-fast modular blockchains called RollApps. By integrating a fork of Osmosis, users are able to pay transaction fees with any token that has a liquidity pool paired with the native DYM coin.

Additionally, a taker fee is imposed on all swaps on the Dymension Hub AMM, subsequently burning any such collected fees.

The `streamer` module allows the Dymension Hub to use the Osmosis model of automatically creating incentivized gauges with immediate reward distributions.

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	-
Code readability and clarity	Medium-High	The codebase generally followed readability and clarity standards, but there were some instances of ambiguous and misleading variable and function names.
Level of documentation	Medium-High	-
Test coverage	Medium-High	`go test` outputs a test code coverage for the `streamer` module of 77.6% statements.

Summary of Findings

No	Description	Severity	Status
1	The invariant checking the last stream ID will break over time, potentially causing the chain to halt	Critical	Resolved
2	Unbounded loop in `PowApprox` can be exploited to halt the chain	Critical	Resolved
3	`MsgEthereumTx` EVM messages can be nested within a `MsgExec` message to bypass Ante handlers and steal transaction fees or perform a Denial-of-Service attack	Critical	Resolved
4	Directly sending funds to the `txfees` module can result in a denial-of-service	Critical	Resolved
5	EVM contract addresses squatting via the vesting module resulting in non-functioning contracts and inaccessible funds	Major	Resolved
6	`CreateStream` does not prevent `startTime` in the past	Minor	Resolved
7	Non-sequential stream creation during `InitGenesis` will cause invariant failure	Minor	Resolved
8	Queries to disabled modules are not prevented	Minor	Acknowledged
9	Inadequate validation of weights in `DistrInfo` creation	Minor	Resolved
10	CLI commands for updating and replacing stream distributions are not registered	Minor	Resolved
11	Errors during stream distribution prevent the processing of other streams	Minor	Resolved
12	A stream is incorrectly considered active after it has finished	Minor	Resolved
13	Rewards can be added to a finished gauge resulting in inaccessible funds	Minor	Resolved
14	Missing message registration for Amino codec	Minor	Resolved
15	Errors during fee swaps may result in an erroneous state	Minor	Resolved

16	CLI long descriptions are lacking detail	Informational	Resolved
17	Gauge sorting validation may lead to unnecessary errors	Informational	Acknowledged
18	Potential key collision due to `KeyIndexSeparator`	Informational	Acknowledged
19	Add a new invariant to validate stream distribution records	Informational	Acknowledged
20	The correct content type is not set	Informational	Resolved
21	Incentives module genesis state validation does not validate `Params` field	Informational	Resolved
22	General code improvements	Informational	Acknowledged

Detailed Findings

1. The invariant checking the last stream ID will break over time, potentially causing the chain to halt

Severity: Critical

The `LastStreamIdInvariant` function in `dymension:x/streamer/keeper/invariants.go` enforces the invariant that the ID of the last stream in the `streams` array must match the last used stream ID retrieved from the `GetLastStreamID` function.

However, the `streams` array retrieved via the `GetStreams` function is sorted by the storage key composed of either the upcoming, active, or finished prefix and the stream ID, and not strictly by the stream ID itself.

Specifically, the order of streams in the KV storage is as follows:

1. Upcoming streams
2. Active streams
3. Finished streams

Over time, as streams will be moved from their initial upcoming state into other states (i.e., active or finished), possibly out of order, depending on their start time, the last stream in the `streams` array will not be equal to the last added stream.

Consequently, this invariant will error and cause the chain to halt.

A test case reproducing the issue is provided in Appendix 1.

We classify this issue as critical since it could cause a chain halt if validators have enabled invariant checks in production.

Recommendation

We recommend sorting the streams in ascending order by their `Id`.

Status: Resolved

2. Unbounded loop in `PowApprox` can be exploited to halt the chain

Severity: Critical

The `PowApprox` function in `dymension-osmosis:osmomath/math.go` computes the approximate power of a base to an exponent using Taylor series approximation.

However, the internal `for` loop does not have a bound on the maximum number of iterations, possibly resulting in a long runtime that is not accounted for in the gas costs of transactions. An attacker can purposefully exploit this with a specially crafted transaction exploiting this expensive computation and spam Osmosis nodes without having to pay very much gas in exchange. By doing this repeatedly, the whole chain can be brought to a halt.

Please note that this issue has been identified and disclosed to Osmosis by Trail of Bits and fixed in PR-6627. For more information on the issue, please refer to the article Numbers turned weapons: DoS in Osmosis' math library by Trail of Bits.

Recommendation

We recommend adding a bound on the maximum number of iterations in the PowApprox function.

Status: Resolved

3. `MsgEthereumTx` EVM messages can be nested within a `MsgExec` message to bypass Ante handlers and steal transaction fees or perform a Denial-of-Service attack

Severity: Critical

The `newCosmosAnteHandler` function in `dymension:app/ante/handlers.go` returns the decorated Ante handler for regular Cosmos SDK transactions. However, while the `RejectMessagesDecorator` decorator is used to reject `MsgEthereumTx` messages, this handler can be bypassed by including a `MsgEthereumTx` message as a nested message using the `MsgExec` message type from the authz module. Consequently, the message is not validated by Ethermint's Ante handlers to ensure that the gas fee is deducted and the gas limit is enforced.

This allows an attacker to steal transaction fees, denominated in DYM, through the refund of unused EVM gas as well as bypassing the block gas limit and gas payment to perform a Denial-of-Service (DoS) attack against the chain. For more information on this issue, please refer to the article Stealing Gas: Bypassing Ethermint Ante Handlers by Jump Crypto.

This issue has been independently reported by a third party during the audit.

Recommendation

We recommend using Ethermint's `NewAuthzLimiterDecorator` decorator to restrict the `MsgEthereumTx` message type from being nested in a `MsgExec` message.

Status: Resolved

4. Directly sending funds to the `txfees` module can result in a denial-of-service

Severity: Critical

The `ModuleAccountAddrs` function in `dymension:app/app.go:936-946` returns a map of all module account addresses and a boolean indicating if the module account is allowed to receive funds through direct and explicit actions, such as `MsgSend` or `SendCoinsFromModuleToAccount` bank messages.

However, in line `944`, the `txfees` Cosmos SDK module is explicitly allowed to receive funds. This opens up a griefing possibility in the `txfees` module's `AfterEpochEnd` hook function in `dymension-osmosis:x/txfees/keeper/hooks.go:34-82` that is executed after an epoch ends.

Specifically, the `AfterEpochEnd` function retrieves all its module account's coin balances in line `43` via the `GetAllBalances` function and subsequently iterates over them. By sending a lot of different coin denoms via ICS-20 transfers to the `txfees` module, the `AfterEpochEnd` function will consume a lot of computational resources, potentially resulting in a denial-of-service (DoS).

As the Dymension chain does not have a `tokenfactory` module to create new tokens easily, the attacker would need to create numerous tokens on other IBC-connected chains and transfer them to the Dymension chain, which makes the attack more difficult to execute. Thus we classify this issue as major instead of critical.

Recommendation

We recommend adding the `txfees` module to the bank module's blocklist in the `ModuleAccountAddrs` function.

Status: Resolved

5. EVM contract addresses squatting via the vesting module resulting in non-functioning contracts and inaccessible funds

Severity: Major

During the Dymension app's initialization in `dymension:app/app.go:707`, the vesting module is enabled. By using the `MsgCreateVestingAccount`, `MsgCreatePermanentLockedAccount`, or `MsgCreatePeriodicVestingAccount` messages, anyone can create a new vesting account for a specific account.

However, as the address of an EVM contract deployed via the `CREATE` or `CREATE2` opcodes is deterministic and predictable, an attacker can front-run the contract deployment and create a vesting account for this address. As a result, the account is internally not stored as an `EthAccountI` type but as a `VestingAccount` type, which prevents storing the contract's

code, thus rendering the contract non-functioning and any funds sent to the contract address inaccessible.

This issue has been independently reported by a third party during the audit.

Recommendation

We recommend either disabling the vesting module or restricting the vesting messages via the Ante handler.

Status: Resolved

6. `CreateStream` does not prevent `startTime` in the past

Severity: Minor

The `CreateStream` function in `dymension:x/streamer/keeper/keeper.go:76` does not validate that the `startTime` of the new stream being created is not in the past. Allowing for the creation of a stream that has a start time in the past will have unintended consequences on the stream's distribution.

While it is likely that this validation was not added since stream creation proposals may make it difficult estimating an appropriate stream start time, it is best practice to have the code enforce this validation to ensure that the streams function properly.

Recommendation

We recommend enforcing that the stream's `startTime` is not in the past. If the time has passed, we recommend setting the `startTime` to the block time that the stream proposal is being passed.

Status: Resolved

7. Non-sequential stream creation during `InitGenesis` will cause invariant failure

Severity: Minor

During `InitGenesis` in `dymension:x/streamer/keeper/genesis.go:19`, `genstate.Streams` is not validated to be sequential. A non-sequential list will cause the `LastStreamIdInvariant` invariant to fail.

Recommendation

We recommend enforcing the streams in `genstate.Streams` are sequential during `InitGenesis`.

Status: Resolved

8. Queries to disabled modules are not prevented

Severity: Minor

The initial version of Dymension is intended to be released with the RollApp-associated modules and functionality disabled. While the modules prevent direct calls and message handling, they expose their querying endpoints. This may provide query responses that are misleading and impact the user experience.

Recommendation

We recommend adding logic to the disabled module's query handlers to prevent them from being directly queried until the RollApp functionality is enabled.

Status: Acknowledged

9. Inadequate validation of weights in `DistrInfo` creation

Severity: Minor

In `dymension:x/streamer/types/distr_info.go:24`, the `Validate` function does not check if weights are positive or the resulting total weight is zero. As a result, it may be the case that there are negative weights in `DistrRecords` or that no distribution records are specified, preventing reward distribution to gauges in the `DistributeByWeights` function in `dymension:x/streamer/keeper/distribute.go:21`.

Recommendation

We recommend validating that distribution records are specified and that all `DistrRecord` weights are positive.

Status: Resolved

10. CLI commands for updating and replacing stream distributions are not registered

Severity: Minor

The `getGovProposalHandlers` function in `dymension:app/app.go:181-197` returns a list of governance proposal handlers for registering as CLI child commands that are mounted under the governance CLI.

However, the handlers for the `UpdateStreamDistributionProposal` and `ReplaceStreamDistributionProposal` proposals are missing and thus not registered as CLI child commands, preventing users from submitting these proposals via the CLI.

Recommendation

We recommend adding the missing handlers for the `UpdateStreamDistributionProposal` and `ReplaceStreamDistributionProposal` proposals to the `getGovProposalHandlers` function.

Status: Resolved

11. Errors during stream distribution prevent the processing of other streams

Severity: Minor

The `DistributeByWeights` function in `dymension:x/streamer/keeper/distribute.go` allocates and distributes funds to gauges based on the specified weights. In lines `16-18`, it is checked that the coins to allocate are not empty. Otherwise, an error is returned. Similarly, the total weight of the distribution is checked in lines `20-22` to be non-zero.

However, such an error is propagated all the way up to the `AfterEpochEnd` hook in `dymension:x/streamer/keeper/hooks.go:62` and thus prevents processing other streams.

Recommendation

We recommend gracefully handling an error while processing a stream distribution and skipping such a faulty stream distribution to ensure that other streams are processed.

Status: Resolved

12. A stream is incorrectly considered active after it has finished

Severity: Minor

The `IsActiveStream` function in `dymension:x/streamer/types/stream.go:29-34` returns `true` if the stream is in an active state during the provided `curTime`. However, the condition in line `30` is incorrect and will always determine the stream to be active if `curTime` is after the stream's `StartTime` and not equal to the current block time.

This is due to the `&&` operator having higher precedence than `||` and thus resulting in the condition that checks if the current time is equal to the stream `StartTime` and if all epochs are processed, evaluating to `false` in case the current block time does not match the start time of the stream.

Consequently, a finished stream can be updated and replaced via a governance proposal which results in an incorrect state in `InitGenesis`.

Similarly, this issue is also observed in the `IsActiveGauge` function in `dymension-osmosis:x/incentives/types/gauge.go:43`. Please note that this issue has been identified and fixed upstream by the Osmosis team in PR-4306.

Recommendation

We recommend adjusting the condition in the `IsActiveStream` function to `(curTime.After(stream.StartTime) || curTime.Equal(stream.StartTime)) && (stream.FilledEpochs < stream.NumEpochsPaidOver)` as well as in the `IsActiveGauge` function.

Status: Resolved

13. Rewards can be added to a finished gauge resulting in inaccessible funds

Severity: Minor

The `AddToGaugeRewards` function in `dymension-osmosis:x/incentives/keeper/gauge.go:146-162` does not determine if the gauge is already finished and allows adding rewards without erroring.

As this function is used within the `DistributeByWeights` function in `x/streamer/keeper/distribute.go:47` to distribute the stream's funds to gauges, funds may be added to a gauge that is already finished, rendering the funds inaccessible.

Due to this issue being mainly caused by misconfiguration of the stream distribution, we classify it as minor.

Please note that this issue has been identified and fixed upstream by the Osmosis team in PR-6445.

Recommendation

We recommend checking in the `AddToGaugeRewards` function if the gauge is finished, and if so, prevent adding rewards to it by returning an error.

Status: Resolved

14. Missing message registration for Amino codec

Severity: Minor

The `RegisterCodec` and `RegisterInterfaces` functions in `dymension-osmosis:x/lockup/types/codec.go` are missing to register the `MsgExtendLockup` and `MsgForceUnlock` messages for the Amino codec.

Recommendation

We recommend adding the missing messages to the `RegisterCodec` and `RegisterInterfaces` functions.

Status: Resolved

15. Errors during fee swaps may result in an erroneous state

Severity: Minor

After an epoch ends, the `AfterEpochEnd` function of the `txfees` module in `dymension-osmosis:x/txfees/keeper/hooks.go` swaps all received non-DYM transaction fees to DYM and burns them.

However, if the `RouteExactAmountIn` function called in line `67` internally errors, and due to using `continue` in this error case, the state machine is not reset, and the next fee coin is processed, resulting in a potentially erroneous state.

Recommendation

We recommend wrapping the `RouteExactAmountIn` function with the `ApplyFuncIfNoError` utility function to create a new cache context and reset the state machine in case of an error.

Status: Resolved

16. CLI long descriptions are lacking detail

Severity: Informational

The stream queries in `dymension:x/streamer/client/cli/query.go` do not provide detailed long descriptions. Long descriptions improve the user experience of the CLI and should contain all relevant details of the query command and its usage. Currently, most of the queries In the file only provide a short description.

Recommendation

We recommend providing long descriptions of the queries in `dymension:x/streamer/client/cli/query.go`.

Status: Resolved

17. Gauge sorting validation may lead to unnecessary errors

Severity: Informational

The `validateGauges` function in `dymension:x/streamer/keeper/distr_info.go:41` performs a validation to ensure the gauges are supplied in sorted order. This validation may be problematic as it may cause valid proposals to be rejected unnecessarily, as gauges could be sorted by the function.

Recommendation

We recommend updating the `validateGauges` function to sort the gauges before saving them.

Status: Acknowledged

18. Potential key collision due to `KeyIndexSeparator`

Severity: Informational

The `streamer` module utilizes a specific encoding scheme for processing keys, as defined in `dymension:x/streamer/types/keys.go`. This scheme includes the use of `KeyIndexSeparator`, `KeyPrefixStreams`, and various other key prefixes.

There is a potential for key collision due to the current implementation of `KeyIndexSeparator` set to `0x07`. This byte is added after each prefix key. The defined key prefixes include `KeyPrefixStreams = 0x04`, `KeyPrefixUpcomingStreams = 0x0400`, `KeyPrefixActiveStreams = 0x0401`, and `KeyPrefixFinishedStreams = 0x0402`.

While this does not cause any issues in the current implementation, the existence of a new key prefix `KeyPrefixNewStreams` set to `0x0407` will lead to a key collision because `combineKeys(KeyPrefixStreams, ID1)` and `combineKeys(KeyPrefixNewStream, ID2)` equal to `0x0407` and `0x040707` start with the same bytes.

Recommendation

We recommend reconsidering the encoding scheme.

Status: Acknowledged

19. Add a new invariant to validate stream distribution records

Severity: Informational

In `dymension:x/streamer/keeper/invariants`, no invariant currently checks that the sum of records weights is equal to the total weight of distribution and all the weights are greater than 0.

Recommendation

We recommend adding the invariant described above.

Status: Acknowledged

20. The correct content type is not set

Severity: Informational

In `dymension:app/healthcheck.go:47`, `application/json` type is not set, so HTTP resources for error and normal responses will have different content types.

Recommendation

We recommend setting `application/json` for all HTTP responses.

Status: Resolved

21. Incentives module genesis state validation does not validate `Params` field

Severity: Informational

The `Validate` function in `dymension-osmosis:x/incentives/types/genesis.go:42-47`, called during the

genesis state validation, misses validation of the genesis state's `Params`, ensuring that the `DistrEpochIdentifier` is valid.

Please note that this issue has been identified and fixed upstream by the Osmosis team in PR-6529.

Recommendation

We recommend calling the `Validate` function of the `Params` type during the genesis state validation.

Status: Resolved

22. General code improvements

Severity: Informational

In several instances of the codebase, the code quality and readability can be improved:

- In `dymension:x/streamer/keeper/genesis_test.go:30`, there is an ineffectual assignment to `err` that leads to ignoring the error.
- In `dymension:cmd/dymd/main.go:16-18`, type switch on errors fails on wrapped errors.
- Remove unused functions (e.g., `dymension:cmd/dymd/cmd/inspect_tendermint.go:19`, `dymension:x/streamer/types/keys.go:44`).
- Remove unused errors (e.g., `dymension:x/streamer/types/errors.go:9,11,15,16`).
- Remove unused constants and variables (e.g., `dymension:x/streamer/types/genesis.go:6`, `dymension:x/streamer/types/events.go:9`, `dymension:x/streamer/client/proposal_handler.go:11`).
- Mark unused parameters using `_` or remove them (e.g., `dymension:cmd/dymd/cmd/inspect.go:32`, `x/streamer/types/codec.go:18`).
- In `dymension:app/healthcheck.go:66`, the error is not checked and ignored.
- `dymension:proto/dymension/streamer/gov_distribution.proto` contains useful comments, but they are hard to find. Moving the comments to another place close to implementation would be helpful.
- The name of the `StreamsInvariant` invariant in `dymension:x/streamer/keeper/invariants.go:65` can be improved.
- In `dymension:app/ante/ante_options.go:30-47`, there is no validation that `IBCKeeper`, `FeegrantKeeper`, and `ExtensionOptionChecker` are not `nil`.
- `dymension:x/streamer/types/distr_info.go` uses a deprecated `types/math.go` package.

- `sdkerrors.Register` used in `x/streamer/types/errors.go:9-19` is deprecated, functionality of that package has been moved to `cosmossdk.io/errors`.
- In `x/streamer/client/cli/tx_create_stream.go:86-87` `govcli.FlagTitle` and `govcli.FlagDescriptionis` are deprecated. They are only used for v1beta1 legacy proposals.
- Remove todos and create issues instead (e.g., `dymension:x/streamer/client/cli/utils.go:12`, `dymension:app/params/config.go:62`, `dymension:x/streamer/keeper/keeper.go:60`).

Recommendation

We recommend applying the recommendations mentioned above.

Status: Acknowledged

Appendix: Test Cases

1. **Test case for "The invariant checking the last stream ID will break over time, causing the chain to halt"**

The following test case reproduces the described issue.

```
// Tested in x/streamer/keeper/genesis_test.go
func TestStreamerOrder(t *testing.T) {
	app := app.Setup(t, false)
	ctx := app.BaseApp.NewContext(false,
tmproto.Header{}).WithBlockTime(time.Now())

	// checks that the default genesis parameters pass validation
	validateGenesis := types.DefaultGenesis().Params.Validate()
	require.NoError(t, validateGenesis)

	// create coins, lp tokens with lockup durations, and a stream for
this lockup
	coins := sdk.Coins{sdk.NewInt64Coin("stake", 10000)}
	startTime := time.Now()

	distr := types.DistrInfo{
		TotalWeight: math.NewInt(100),
		Records: []types.DistrRecord{{
			GaugeId: 1,
			Weight:  math.NewInt(50),
		},
			{
				GaugeId: 2,
				Weight:  math.NewInt(50),
			},
		},
	}

	stream := types.Stream{
		Id:                1,
		DistributeTo:      &distr,
		Coins:             coins,
		NumEpochsPaidOver: 2,
		DistrEpochIdentifier: "day",
		FilledEpochs:      2,
		DistributedCoins:  sdk.Coins(nil),
		StartTime:         startTime.Add(time.Second * 10).UTC(),
```

```
// stream starts in 10 seconds
	}

	stream2 := types.Stream{
		Id:                  2,
		DistributeTo:        &distr,
		Coins:               coins,
		NumEpochsPaidOver:   2,
		DistrEpochIdentifier: "day",
		FilledEpochs:        0,
		DistributedCoins:    sdk.Coins(nil),
		StartTime:           startTime.Add(time.Second * 60 * 60 *
24 * 1).UTC(), // stream starts in 1 day
	}

	stream3 := types.Stream{
		Id:                  3,
		DistributeTo:        &distr,
		Coins:               coins,
		NumEpochsPaidOver:   2,
		DistrEpochIdentifier: "day",
		FilledEpochs:        0,
		DistributedCoins:    sdk.Coins(nil),
		StartTime:           startTime.Add(time.Second * 60 * 60 *
24 * 2).UTC(), // stream starts in 2 days
	}

	// initialize genesis with specified parameter, the stream created
earlier, and lockable durations
	app.StreamerKeeper.InitGenesis(ctx, types.GenesisState{
		Params:       types.Params{},
		Streams:      []types.Stream{stream, stream2, stream3},
		LastStreamId: 3,
	})

	// check that the stream created earlier was initialized through
initGenesis and still exists on chain
	streams := app.StreamerKeeper.GetStreams(ctx)
	lastStreamID := app.StreamerKeeper.GetLastStreamID(ctx)
	require.Len(t, streams, 3)
	require.Equal(t, stream, streams[0])
	require.Equal(t, stream2, streams[1])
	require.Equal(t, stream3, streams[2])
	require.Equal(t, lastStreamID, uint64(3))
```

```
	// Forward block time by 1 minute to start the stream with id 1
	ctx = ctx.WithBlockTime(startTime.Add(time.Second * 60))

	// Move stream with id 1 from upcoming to active
	err := app.StreamerKeeper.MoveUpcomingStreamToActiveStream(ctx,
stream)
	require.NoError(t, err)

	streams = app.StreamerKeeper.GetStreams(ctx)
	lastStreamID = app.StreamerKeeper.GetLastStreamID(ctx)
	require.Len(t, streams, 3)
	// The order of streams has changed
	require.Equal(t, stream2, streams[0])
	require.Equal(t, stream3, streams[1])
	require.Equal(t, stream, streams[2])

	var broken bool

	if len(streams) == 0 {
		if lastStreamID != 0 {
			broken = true
		}
	} else if streams[len(streams)-1].Id != lastStreamID {
		broken = true
	}

	require.False(t, broken, "last stream id invariant broken")
}
```