Security Audit Report

dYdX: Rewards, Consensus and Other Changes Q3 2023

Authors: Manuel Bravo, Andrija Mitrovic, Mirel Dalcekovic

Last revised 28 September, 2023

Table of Contents

Audit Overview

The Project

In August 2023, dYdX has engaged Informal Systems to work on partnership and conduct security audit of the new v4 open-source software that dYdX has been developing. For this phase of our partnership the following components and changes were agreed to be the scope:

- x/bridge and related modules, which is a generic module that facilitates the migration of tokens from the Ethereum chain to v4 (if deployed).
- Thoroughly examining the default reward mechanisms within the v4 software.
- Analyzing consensus adjustments made in the dYdX fork of CometBFT, aimed at ensuring the chain's uninterrupted operation and liveness (if deployed).

Scope of this report

The agreed-upon work plan consisted of the following tasks:

- Evaluating and analyzing the code and specifications of the x/reward and x/vest modules.
- Code inspection, spec and protocol analysis for x/bridge and related modules – implemented on the Cosmos dYdX v4 software (black-boxing the Ethereum side in the analysis).
- Protocol analysis of the consensus changes.

Audit plan:

Auditing was organized between August 22nd and September 15th, 2023

Conducted work

During the kick-off meeting, representatives from Informal Systems and dYdX agreed on the audit process and identified the need for several onboarding sessions to ensure a smooth and efficient transition for the auditing team to the project.

The team began by reviewing the shared documentation (Notion pages shared with Informal Systems) and examining the code base to gain a preliminary understanding of the system before conducting code walkthrough sessions with the dYdX development team. The team prepared questions for the upcoming meetings with the dYdX team to gain a high-level overview and walkthrough of the code base.

After establishing a clear understanding of the existing implementation, the auditing team performed a manual code review with a primary focus on code correctness and a critical points analysis of the agreed-upon scope.

Timeline

- 22.08.2023: Audit partnership Q3 2023 start - estimation of shared scope.
- 23.08.2023: Kick-off meeting with the dYdX team.
- 28.08.2023: Code walkthrough for x/rewards and x/vest modules.
- 31.08.2023: Weekly sync - questions regarding the x/rewards and x/vest modules, and consensus changes - review of readiness and discussion about the expectations of the PR needed to be review. The dYdX team and auditing team agreed that this scope's analysis will postponed until the fix for the existing issue is merged. It was agreed to prioritize this in the dYdX team, so that auditing team has everything ready on September 5th.
- 31.08.2023: x/bridge and related modules walkthrough.

- 07.09.2023: Meeting with x/bridge and related modules team developers, questions and presentation of current findings. The auditing team asked for the review of the findings and remediation plans.
- 11.09.2023: Weekly sync meeting. Auditing team has shared the status of the tasks until the closure meeting.
- 14.09.2023: Closure meeting. Overview of findings, sharing the report with dYdX team. Agreements about potential additional polishing of the report (if needed).
- 19.09.2023: The draft version of the report has been shared with the dYdX team.
- 28.09.2023: Final version of the report has been shared with the dYdX team with updated remediation plans for findings.

Conclusions

In general, we found the codebase to be of high quality. The documentation should be updated or provide additional information for external readers about the actual business usage of the features developed - specially when it comes to the x/bridge and related modules.
The code is well-structured and easy to follow and is covered with tests, which could be richer with additional edge test cases.

We found couple of higher severity issues in the codebase, that we believe should be addressed right away.
Most of the remaining issues we encountered were related to optimizations, validations, aesthetics and suggestions for improving the current design.

Protocol analysis of latest dYdX CometBFT fork changes has resulted in a high level analysis shared with the clients in this document. We have concluded that the changes to consensus mitigate the liveness problem but do not solve it completely. We have made some recommendations to further mitigate the problem and provided some ideas with the potential of solving the issue completely.

Overall, we are impressed with the quality of the codebase and by the innovative and creative utilization of ABCI++ and various other solutions suggested to the CometBFT team and implemented in dYdX CometBFT fork.

Further Increasing Confidence

The scope of this audit was limited to manual code review and manual analysis and reconstruction of the protocols. To further increase confidence in the protocol and the implementation, we recommend following up with more rigorous formal measures, including automated model checking and model-based adversarial testing. Our experience shows that incorporating test suites driven by Quint / TLA+ models that can lead the implementation into suspected edge cases and error scenarios enables discovery of issues that are unlikely to be identified through manual review. This is especially relevant to the analysis of locks in the CometBFT fork.

As a follow-up to the protocol analysis of the dYdX CometBFT changes, an appropriate code inspection analysis should be done to guarantee the correctness of the dYdX CometBFT fork. This analysis will also verify that all the required changes from the CometBFT fork have been successfully integrated.

Disclaimer

report should not be considered or utilized as a complete assessment of the overall utility, security or bug free status of the code.

This audit report contains confidential information and is only intended for use by the client. Reuse or republication of the audit report other than as authorized by the client is prohibited.

This report is not, nor should it be considered, an “endorsement”, “approval” or “disapproval” of any particular project or team. This report is not, nor should it be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts with Informal to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor does it provide any indication of the client’s business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should it be leveraged as investment advice of any sort.

Blockchain technology and cryptographic assets in general and by definition present a high level of ongoing risk. Client is responsible for its own due diligence and continuing security in this regard.

Audit Dashboard

Target Summary

- **Type**: Specification and Implementation
- **Platform**: Go
- **Artifacts**
  - **Rewards**: x/rewards and x/vest modules - over 0e84b4f
  - **Bridge** - x/bridge, x/delaymsg and bridge daemon - over e2ea4e4
  - **Consensus changes** - Pull request initial scope / Commits: Commit_1, Commit_2, Commit_3 - newly agreed scope due to existing issue impacting to the initially communicated PR.
    - Agreed to do the high level analysis over changes made and provide the protocol semantic analysis results.

Engagement Summary

- **Dates**: 22.08.2023 until 15.09.2023.
- **Method**: Manual code review & protocol analysis
- **Employees Engaged**: 3

Severity Summary

Finding Severity	**#**
Critical	1
High	0
Medium	3
Low	5
Informational	4
Total	13

Resolution Status Summary

Resolution Status	**#**
Resolved	2
Risk-accepted	9
Functioning as Designed	2
Total	13

Findings

Title	Type	Severity	Impact	Exploitability	Issue
Dispatching of delayed messages should be done over cached context	IMPLEMENTATION	4 CRITICAL	3 HIGH	3 HIGH	https://github.com/dydxprotocol/v4-chain/pull/214
Events from events map in bridge manager are not deleted after processing	IMPLEMENTATION	2 MEDIUM	2 MEDIUM	2 MEDIUM	
Consensus changes does not make v4 live	PROTOCOL	2 MEDIUM	3 HIGH	1 LOW	
Struct alignment issues	IMPLEMENTATION	2 MEDIUM	2 MEDIUM	2 MEDIUM	
Missing validation of x/rewards parameters in MsgUpdateParams	IMPLEMENTATION	1 LOW	2 MEDIUM	1 LOW	
Optimization of x/rewards transient store RewardShare entries	IMPLEMENTATION	1 LOW	1 LOW	1 LOW	
Delayed messages dispatching optimization and code simplicity suggestions	IMPLEMENTATION	1 LOW	1 LOW	1 LOW	
CmdQueryBlockMessageIds validations are missing for block id value	IMPLEMENTATION	1 LOW	1 LOW	1 LOW	https://github.com/dydxprotocol/v4-chain/pull/224

Title	Type	Severity	Impact	Exploitability	Issue
Ethereum dYdX users could initiate multiple bridge transfers for a very small amount and cause bridging DDoS	PROTOCOL	1 LOW	2 MEDIUM	1 LOW	
Minor code changes in x/rewards and x/vest modules	IMPLEMENTATION	0 INFORMATIONAL	0 NONE	0 NONE	
Documentation formulas for calculations in vesting and rewards modules contain errors	DOCUMENTATION	0 INFORMATIONAL	0 NONE	0 NONE	
Improve dispatching of delayed messages design	PROTOCOL	0 INFORMATIONAL	0 NONE	0 NONE	
Minor code changes in bridge components	IMPLEMENTATION	0 INFORMATIONAL	0 NONE	0 NONE	

Minor code changes in x/rewards and x/vest modules

Title	Minor code changes in x/rewards and x/vest modules
Project	dYdX: Rewards, Consensus and Other Changes Q3 2023
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	

Involved artifacts

- x/rewards/keeper/keeper.go
- x/feetiers/keeper/keeper.go
- x/vest/keeper/keeper_test.go

Description

Here is the list of aesthetic and minor code improvements found during the code inspection of the dYdX audit of x/rewards and x/vest modules. They do not pose a security threat nor do they introduce an issue, but the following suggestions are shared to improve the code readability, keep consistency, optimize, and improve logging.

- for clarity purposes rename: lowestMakerFee to lowestMakerFeePpm here and rename function name from `GetLowestMakerFee` to `GetLowestMakerFeePpm` and variables name here.
- instead of creating two BigInt0 objects with `lib.BigInt0()`, create one and use in this two places for comparing: here and here.
- Wrong comment in test. It is not aligned with input values. The vester balance is 2 000 000 not 1 000 000.

Problem Scenarios

Findings listed above could not introduce any issues, they are suggestions for code improvements.

Recommendation

As explained in the Description section.

Remediation Plan

Risk accepted: The dYdX team accepted the risk of issues listed in this finding. They plan on resolving most of them in 2024.

Missing validation of x/rewards parameters in MsgUpdateParams

Title	Missing validation of x/rewards parameters in MsgUpdateParams
Project	dYdX: Rewards, Consensus and Other Changes Q3 2023
Type	IMPLEMENTATION
Severity	1 LOW
Impact	2 MEDIUM
Exploitability	1 LOW
Issue	

Involved artifacts

- x/rewards/types/params.go

Description

1. ValidateBasic() for `MsgUpdateParams` could contain additional check for expected and valid values of `feemultiplierPpm` in params.Validate() function.
    - `feeMultiplierPpm` should not be 0. The direct consequence of this is that reward sharing will be "disabled", since the following calculation of expected total reward tokens `bigRatRewardTokenAmount`

      ```
      bigRatRewardTokenAmount = lib.BigRatMulPpm(
          bigRatRewardTokenAmount,
          params.FeeMultiplierPpm,
      )
      ```

      would be equal to 0 - here.

      Afterwards, when determining the `tokensToDistribute` amount, we would get 0, once again and return from the `ProcessRewardsForBlock` - here.

- ```
  // Get tokenToDistribute as the min(F, T).
  tokensToDistribute := lib.BigMin(rewardTokenBalance.Amount.BigInt(),
  bigIntRewardTokenAmount)
  ....
  if tokensToDistribute.Sign() == 0 {
  ```
  ```

```
    // Nothing to distribute. This can happen either when there is no reward
token in the treasury account,
    // or if no reward shares were recorded for this block.
    return nil
}
```

If there is a need to have a special config parameter enabling rewards sharing it should be introduced.

2. Also, an additional stateful check could be added for `MsgUpdateParams`, since for a valid existing market, marketId value.
    - if rewards parameters would contain invalid `marketId` - we would find out that at this point in the EndBlocker(). Which means, we would reject each block due to inability of processing the rewards in the EndBlock().

Problem Scenarios

As explained above, in the description paragraph.

Recommendation

Introduce additional validations of `feeMultiplierPpm` and `marketId` parameters. Even though this `MsgUpdateParams` will be executed upon the governance proposal passed - the consequences of the invalid values are not negligible.

Remediation Plan

Risk-accepted: dYdX team plans on fixing this issue in 2023 Q4.

Documentation formulas for calculations in vesting and rewards modules contain errors

Title	Documentation formulas for calculations in vesting and rewards modules contain errors
Project	dYdX: Rewards, Consensus and Other Changes Q3 2023
Type	DOCUMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	

Involved artifacts

- x/rewards spec
- x/vest spec

Description

Specification documents shared with Informal audit team as onboarding material contains errors or is missing information for calculations executed for x/vest and x/rewards modules.

- x/rewards spec should contain information that rewards parameter `feeMultiplierPpm` is crucial for determining the total number of the expected reward tokens calculated, since it defines:

> *The amount (in ppm) that fees are multiplied by to get the maximum rewards amount*

as explained in the spec document and in the code here. The parameter is used in calculations in the code here.

- x/vest module's formula in the docs does not correspond to the code (comments and code).

Recommendation

Revise the specification documents to align them with the code implementation, as suggested in the description paragraph.

Remediation Plan

Risk-accepted: dYdX team plans on fixing this issue in 2023 Q4.

Optimization of x/rewards transient store RewardShare entries

Title	Optimization of x/rewards transient store RewardShare entries
Project	dYdX: Rewards, Consensus and Other Changes Q3 2023
Type	IMPLEMENTATION
Severity	1 LOW
Impact	1 LOW
Exploitability	1 LOW
Issue	

Involved artifacts

- x/rewards/keeper/keeper.go
- x/rewards/types/reward_share.pb.go

Description

RewardShare struct contains info about the address and the share weight:

```
// RewardShare stores the relative weight of rewards that each address is
// entitled to.
type RewardShare struct {
    Address string
`protobuf:"bytes,1,opt,name=address,proto3" json:"address,omitempty"`
    Weight  github_com_dydxprotocol_v4_chain_protocol_dtypes.SerializableInt
`protobuf:"bytes,2,opt,name=weight,proto3,customtype=github.com/dydxprotocol/v4-
chain/protocol/dtypes.SerializableInt" json:"weight"`
}
```

Entire struct is serialized and saved as a value, for a data entry where a key is equal to an address. So we are duplicating addressees information with SetRewardShare(), while we can save only weights as a value in a store for addresses keys.

```
// SetRewardShare set a reward share object under rewardShare.Address.
func (k Keeper) SetRewardShare(
    ctx sdk.Context,
    rewardShare types.RewardShare,
) {
    store := prefix.NewStore(ctx.KVStore(k.transientStoreKey),
types.KeyPrefix(types.RewardShareKeyPrefix))
    b := k.cdc.MustMarshal(&rewardShare)
```

```
    store.Set(types.RewardShareKey(
        rewardShare.Address,
    ), b)
}
```

Problem Scenarios

The described issue does not impose any security threat, but optimizing memory issues if they are not impacting performance are always considered an improvement.

Recommendation

Refactor the way reward share entries are saved in the transient KV store. Store only `RewardShare` data structure part (weight values) not employed as a key in the Cosmos SDK transient store, and then recreate data structures on the fly when reading them from the store by combining the key and the value parts.

Remediation Plan

Risk-accepted: dYdX team plans on fixing this issue in 2023 Q4.

Dispatching of delayed messages should be done over cached context

Title	Dispatching of delayed messages should be done over cached context
Project	dYdX: Rewards, Consensus and Other Changes Q3 2023
Type	IMPLEMENTATION
Severity	4 CRITICAL
Impact	3 HIGH
Exploitability	3 HIGH
Issue	https://github.com/dydxprotocol/v4-chain/pull/214

Involved artifacts

- x/delaymsg/keeper/dispatch.go

Description

When calling the execution of the wrapped message contained in the delay message, MsgService function execution is triggered:

```
handler := k.Router().Handler(msg)
if _, err := handler(ctx, msg); err != nil {
     k.Logger(ctx).Error("failed to execute delayed message with id %v: %v", id, err)
}
```

This is executed in `DispatchMessagesForBlock` function in the `EndBlock` logic of `x/delaymsg` module.

```
func EndBlocker(ctx sdk.Context, k types.DelayMsgKeeper) {
    defer telemetry.ModuleMeasureSince(types.ModuleName, time.Now(),
telemetry.MetricKeyEndBlocker)
    keeper.DispatchMessagesForBlock(k, ctx)
}
```

To draw a parallel, lets take a look at `DeliverTx` processing in Cosmos SDK's:

- For each `tx` being delivered, there is a new context based of the existing context - cached context created prior to executing all the messages in `RunMsg`.

- In case that one of the messages fails to execute (for any reason) the changes made to the cached `msCache` context of `runMsgCtx` can not be persisted.

In case of complete success, the changes made in the cached `runMsgCtx` will be written to the underlying store.

In the `EndBlocker` phase, there is no branching the `MultiStore` implemented. Branching should be done per delayed message in a similar way like in the `DeliverTx` phase.
In the current implementation, all the changes made during the delayed message dispatching are persisted in the `deliverTx` state, propagated from the `baseapp` when triggering the EndBlocker execution of logic in modules.

Problem Scenarios

In case of any message execution failure, the state will be left in the inconsistent state, since the `deliverState` context is not cached prior to execution of each delayed message.

Recommendation

Implement caching mechanism over context prior to triggering the message handler execution.
Write cached state to the underlying store after each successful message execution.

In case of failure, log the error and drop the changes made in the state.
As a good example of how it is done, we suggest taking a look at the public open source Axelar Bridge solution where the `utils` defined RunCached function is used for the `MultiStore` manipulation.

Remediation Plan

Resolved: The pull request containing the suggested fix for executing messages over cached context was merged into the main branch and reviewed at the moment of final report generation process.

Additionally, an issue related to the suggested implementation was identified internally by the dYdX team (regarding the propagation of emitting events from cached context), but the team is working on fixing it.

Delayed messages dispatching optimization and code simplicity suggestions

Title	Delayed messages dispatching optimization and code simplicity suggestions
Project	dYdX: Rewards, Consensus and Other Changes Q3 2023
Type	IMPLEMENTATION
Severity	1 LOW
Impact	1 LOW
Exploitability	1 LOW
Issue	

Involved artifacts

- x/delaymsg/keeper/delayed_message.go
- x/delaymsg/keeper/dispatch.go
- x/delaymsg/keeper/block_message_ids.go

Description

We noticed several possible places for improvement, depending on the needs of potential future development.

1. When dispatching messages in the `EndBlock` function, we need to read ids of delayed messages saved to be executed in current block. Once we retrieved the ids, there is no need to access the store once again and read the same ids at the point when after executing them, the delayed messages need to be removed from the store in `deleteMessageIdFromBlock()`
2. It is not clear why are we performing one by one dispatching of all messages and then at the end, once again, deleting of all dispatched messages one by one, from the store.
   Suggestion: dispatching messages one by one and deleting them in the same loop.
   Our understanding of the current implementation is that:
   - all the delayed messages for the current block will try to be dispatched
   - if the execution of the delayed message fails for any reason, it will be ignored - error will be logged, but no panicking and halting of the node/chain will be possible
   - after all the messages are processed with dispatching - we are deleting all the messages, regardless of their success or any other reason.
3. Finally, it is unclear why we are performing deletion of the messages one by one in case of `DispatchMessagesForBlock` at all.

   Suggestion: is to implement `DeleteMessages` function in delayed_message.go instead or besides the existing `DeleteMessage` function, in case that there is a need for existence of this function due to some further development.
   Our understanding of the current implementation:

- If the delay messages were processed with the `EndBlock`, the entire record for the current block height in `BlockMessageIdsPrefix` store, should be deleted. There is no need for deleting one by one entry, keeping the order of the delay message ids and storing the changes, for each delayed message.

Based on our current understanding of the x/delaymsg module and implementation behind, we suggest applying the improvement 3.
The only possible situation when we could have errors returned in the following functions is when there is no message found in the store:
`DeleteMessage` - here
`deleteMessageIdFromBlock` - here.

The last possible place for returning an error is actually unreachable code, since:
we are calling the `deleteMessageIdFromBlock` function for delay message id looping through `blockMessageIds` retrieved from the store with `GetBlockMessageIds.` Thus, there is no possible way for getting an error even if we have duplicates in the `blockMessageIds` entry for a block height, since we would return here in that case leaving the duplicate id value in the `GetBlockMessageIds` and record existing for block height (with current implementation).

Problem Scenarios

Listed improvements and ambiguity above, do not impose a security threat.
However, the code is unclear and the design ideas behind it, as well. Current implementation seems like there is more future development planned, that will provide explanations for the selected implementation approach.

Recommendation

Reconsider suggestions provided above and aim to optimize the implementation for simplicity, minimizing memory usage as much as possible.

Remediation Plan

Risk-accepted: The dYdX team agreed to implement suggestion number three: to remove DeleteMessage, add DeleteMessages, in the next 2.5 weeks (as communicated on September 13th).
At the moment of generating the report the PR was not merged and reviewed by the auditing team.

Improve dispatching of delayed messages design

Title	Improve dispatching of delayed messages design
Project	dYdX: Rewards, Consensus and Other Changes Q3 2023
Type	PROTOCOL
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	

Involved artifacts

- x/delaymsg/keeper/dispatch.go
- x/delay-msg spec

Description

According to the:

- analysis and remarks presented in the finding: *Delayed messages dispatching optimization and code simplicity suggestions* and
- notes we have found in the `x/delaymsg` spec,

we could conclude that there could be a possibility of implementing the **retry mechanism** of delay messages that failed to be executed.

Retry mechanism would probably make sense only in specific cases. The protocol would first analyze the error retrieved with routed execution of delayed message (e.g. if there are not enough funds on the bridge module account to perform bridging) and try to place the new delayed message for the wrapped message, once again. New delay message should be scheduled, for another block height.

The information about the failed message should be potentially persisted for the processed old block height and the failed message should not be deleted, since it could be used for:

- future retry mechanism development
- feedback loop that would provide information to the Ethereum side if the transfer has succeeded

With the current implementation, all the messages (both successful and failed during the execution) will be deleted from the store, at the end of the current block processing.

Problem Scenarios

The analysis provided in the Description paragraph contains suggestions for possible future development ideas and highlights the possible changes in the current design if some future features are in plan.

Recommendation

If retrying mechanism is something that will be implemented and there is potential use when providing info about failed messages in old block heights, we suggest you delete only successfully routed and executed delay messages.

Leave the information about the failed messages in the store.

For the retry mechanism: potentially, during migration to newer dYdX version old block height delayed messages remaining in the store could be rescheduled during the upgrade and placed as new delayed messages for a new block height.

Implement governance message for rescheduling the delay messages to some other block height.
The existing failed messages in blocks from the past could be queried and in case they are successfully validated, they could be rescheduled for another block height.
This feature could be used in cases:

- of issues with bridge and
- retrying failed delay messages execution.

Remediation Plan

Functioning as Designed: The issue above contains suggestions on how to improve delayed dispatching messages design, depending on intended usage.

CmdQueryBlockMessageIds validations are missing for block id value

Title	CmdQueryBlockMessageIds validations are missing for block id value
Project	dYdX: Rewards, Consensus and Other Changes Q3 2023
Type	IMPLEMENTATION
Severity	1 LOW
Impact	1 LOW
Exploitability	1 LOW
Issue	https://github.com/dydxprotocol/v4-chain/pull/224

Involved artifacts

- x/delaymsg/types/query.pb.gw.go
- x/delaymsg/client/cli/query.go
- lib/bytes.go

Description

For `CmdQueryBlockMessageIds` query it is possible that a query request is created with a negative block height value. The following call stack shows that the check is missing:

- It is possible that a query request is created with a negative block height value - here.
- Function can retrieve the `BlockHeight` argument with negative value.
- There are no checks when calling the execution of the `GetBlockMessageIds` function on the keeper.
- When generating the key within the `Int64ToBytesForState` function to retrieve a value from the store, a problem arises when converting to uint64, resulting in an incorrect block height value.

Problem Scenarios

Query will return invalid results for negative block id, and depending on the use of the query results the impact can be of varying severity.

Recommendation

Add validation of the input argument `BlockHeight` and change the library function `Int64ToBytesForState`, to contain the check for negative id values prior to casting them to uint64.

Remediation Plan

Resolved: This issue is fixed by adding validations here and here (with specific error messaging), but left `Int64ToBytesForState` untouched, due to confirming with the development and auditing team that casting to uint64 will not in this case impact on returning the wrong block id. The underlying bits presentation of the block id number is used, so the key will be unique.

The auditing team lowered the severity, since the only issue was that validations were missing.

Ethereum dYdX users could initiate multiple bridge transfers for a very small amount and cause bridging DDoS

Title	Ethereum dYdX users could initiate multiple bridge transfers for a very small amount and cause bridging DDoS
Project	dYdX: Rewards, Consensus and Other Changes Q3 2023
Type	PROTOCOL
Severity	1 LOW
Impact	2 MEDIUM
Exploitability	1 LOW
Issue	

Involved artifacts

- daemons/bridge/client/client.go
- Ethereum smart contract (out of scope, black boxed)

Description

Ethereum smart contract communicating with dYdX v4 Cosmos chain (if deployed) over the bridge implemented on the Cosmos side is not in the scope of this audit, so we will black box its implementation and analyze possible negative impact to the bridge implemented on the dYdX v4 software.

dYdX v4 bridge daemon would query a smart contract on the Ethereum “source chain” for the logs emitted - representing the initiated transfers of tokens to the “destination chain”.

In case there is no limit or predefined amount of tokens to be bridged over - there is a possibility of creating “user specific” DDoS of with respect to the bridge.

Problem Scenarios

If the bridge daemon queries a Ethereum smart contract for logs and retrieves large number of logs for multiple bridge transfers, coming from one or multiple users (malicious users acting together), it will create the same large number of recognized events on the dYdX Cosmos v4 side in the `bridgeEventManager`.

The consequence would not be the complete DDoS, but the bridge would be occupied for some time with transferring only bridged tokens for specific users.
This could have various impacts depending on the expectations of this bridge behavior.

Recommendation

Protection on both source and destination chain should be introduced, so that described scenario is not possible.

Remediation Plan

Functioning as Designed: The dYdX team believes that this could be protected on the Ethereum side with gas and no additional level of protection is needed due to intended one-way transfers usage of dYdX bridge.

Minor code changes in bridge components

Title	Minor code changes in bridge components
Project	dYdX: Rewards, Consensus and Other Changes Q3 2023
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	

Involved artifacts

- x/bridge/types/genesis.go
- x/delaymsg/keeper/dispatch.go
- x/delaymsg/keeper/delayed_message.go
- x/delaymsg/keeper/block_message_ids.go
- protocol/daemons/bridge/client/client.go

Description

Here is the list of aesthetic and minor code improvements found during the code inspection of the dYdX audit of x/bridge and x/delaymsg modules. They do not pose a security threat nor do they introduce an issue, but the following suggestions are shared to improve the code readability, keep consistency, optimize, and improve logging.

- Consider introducing the maximum valid value for `DelayBlocks` Safety parameter.
- DefaultGenesis bridge state holds either invalid comment or really big value for `DelayBlocks` safety parameter 86 400 blocks.
- There is several places in the `x/delaymsg` module where the *continue and log error* approach is implemented or *multiple checks in the same flow of execution* but we could not see the reasonable explanation:
  - when executing delay message ids - if during delete message () we can not retrieve the message with executed id - panic, do not just log error and continue: here and here.
  - we are looping through `blockMessagesId` retrieved with `GetBlockMessageIds` and for each id we are dispatching the message and deleting the message with `DeleteMessage`, going once again through the same `blockMessageIds`. From `DeleteMessage` → `deleteMessageIdFromBlock` once again the `GetBlockMessageIds` is called and in case if the block height is not found it returns the error where we should probably panic or do not even perform this redundant check.

Suggestion is to accept panicking when the second check is failing in the execution flow. Panicking in that situations is a must to point out that there is a severe issue in the system, even if this means halting the chain to fix the issue. In case of redundant checks, do not perform double check in straightforward execution calls.

- This check could be moved to the top of Validate function?
- In `protocol/daemons/bridge/client/client.go` function there is no need for execution of this code, if no logs are retrieved from the Ethereum smart contract. Add `if len(logs) == 0` condition here and return from function.

Problem Scenarios

Findings listed above could not introduce any issues, they are suggestions for code improvements.

Recommendation

As explained in the Description section.

Remediation Plan

Risk accepted: The dYdX team accepted the risk of issues listed in this finding. They plan on resolving most of them in 2024.

Events from events map in bridge manager are not deleted after processing

Title	Events from events map in bridge manager are not deleted after processing
Project	dYdX: Rewards, Consensus and Other Changes Q3 2023
Type	IMPLEMENTATION
Severity	2 MEDIUM
Impact	2 MEDIUM
Exploitability	2 MEDIUM
Issue	

Involved artifacts

- server/types/bridge/bridge_event_manager.go

Description

Map

```
    // Bridge events by ID
    events map[EventId]BridgeEventWithTime
```

in `BridgeEventManager` struct is only filled with events, but those are never deleted from the map after being processed on v4 (**ACKNOWLEDGED**) .

This lead to continual growth of this map.

Problem Scenarios

This leads to constant accumulation of events in the map, which leads to longer prepare proposal after each arrival of new events from the Ethereum side because iterating and searching through the `bridgeManager.events` map here. This issue could make issue https://informalsystems.atlassian.net/l/cp/1EbpRMw0 more severe if users because of the lack of any limit on the amount to be sent, sent an enormous number of transactions leading to large number of events being stored in the bridge manager thus slowing down the creation of blocks.

Recommendation

Newly acknowledged events in the current block should be removed in the end blocker of the bridge module. This should speed up iteration and accessing speed to the map but not reduce memory consumption due to the specific map behavior in Go.

For a substantial reduction in memory consumption upon deleting processed events from `bridgeEventManager`, we recommend exploring alternative data structures. Removing elements from maps does not guarantee immediate memory reclamation by the garbage collector. When an element is deleted from a map, the memory it occupied is marked as available for future map operations, allowing Go to reuse it when adding new elements. One potential solution is to use pointers with a map of `BridgeEventTime` events, like `map[EventId]*BridgeEventWithTime`.

Remediation Plan

Risk-accepted: The dYdX team is actively working to remediate this issue within the next few weeks and is currently engaged in an ongoing discussion about the solution approach.

Consensus changes does not make v4 live

Title	Consensus changes does not make v4 live
Project	dYdX: Rewards, Consensus and Other Changes Q3 2023
Type	PROTOCOL
Severity	2 MEDIUM
Impact	3 HIGH
Exploitability	1 LOW
Issue	

Involved artifacts

- consensus/state.go

Description

The consensus changes introduced in dYdX v4 do not fix the liveness problem: the application still does not guarantee coherence (Requirement 3 in abci++ spec). In some scenarios (as the one described below), this may cause consensus to never terminate.

The changes mitigate the liveness problem though by minimizing the calls to `processProposal`. Thus, when a correct validator is able to form a proof-of-lock, it is likely that consensus will terminate. Intuitively, this happens because eventually the value for which the proof-of-lock was formed will be proposed and validators will accept it without calling `processProposal`. Note that this was not the case before. Without the changes, validators would call `processProposal` in the above situation, and given that the prices were computed in a previous round, these are likely to have changed significantly, such that validators won't accept the proposal and vote `nil`.

Problem Scenarios

A simple scenario is that in which prices forever fluctuate and we are never able to have +2/3 of the validators voting for any proposal. In more detail:

- Assume round 0 and that its leader is correct.
- Since it is the first round, the leader of the round proposes a new value.
- By the time validators receive the proposal the prices have changed significantly and all correct validators vote `nil`.
- We move to the next round in a situation similar to the original one. This may repeat forever.

Recommendation

We recommend rethinking the price acceptance design to guarantee coherence (Requirement 3 in abci++ spec) or the weaker eventual coherence: there is a point in time after which coherence is guaranteed but it may not be guaranteed from the beginning. We believe that eventual coherence should be enough for liveness. Following we propose some ideas to explore.

Leverage timestamps

1. Assume that marketplaces allow querying prices at a given timestamp: the price daemon can ask for the price at a given time.
2. All validators may be configured to query prices at marketplaces periodically at the same pace:
    a. This means that all validators get the prices for the same timestamps.
    b. Note that this does not guarantee that all validators get the prices for a given timestamp at the same time: price daemons still query based on the validator's local clock and the message delay may be different for each validator.
3. The proposer could propose the prices and the timestamp from which these were computed:
    a. The timestamp could be picked slightly in the past according to the proposer's local clock.
    b. How much in the past could be a function of the round: the higher the round, the more in the past to increase the chances that validators have already received the prices for the proposed timestamp.
    c. Note that this does not mean that proposed timestamps at higher rounds are smaller than timestamps at lower rounds, as timestamps are picked wrt. the proposer's local clock, which advances with rounds.
4. `processProposal` would do the following:
    a. A validator can still reject proposals if the timestamp is too stale in comparison to its local clock. Eventually, assuming there is an upper-bound in clock drifts during synchrony, correct processes won't reject correct proposals due to this (similar to `timely` in PBTS)
    b. If the validator has prices from all marketplaces for the proposed timestamp, it computes the prices and marks the proposal as valid if it matches the proposed prices. Note that this would be guaranteed if the leader is correct.
    c. If the validator is missing some prices for the proposed timestamp, it votes `nil`. Next round there will be a better chance of having the prices for the proposed timestamp.
    d. Note that validators would have to keep prices for multiple timestamps as opposed to keeping only the latest. We believe this can be garbage collected based on the upper-bound in clock drifts and the validator's local clock.

We think that by (3), one could come up with a correctness argument that proves eventual coherence and hence liveness. Of course, this solution depends on the assumption that marketplaces allow querying prices at a given timestamp.

Weaken the price acceptance requirements as rounds increase

1. This has the potential to further mitigate the liveness issue.
2. Assuming that there is an upper bound on how much prices may change over time, this would guarantee eventual coherence. Note that even if this upper bound does not theoretically exist, it will exist in practice with a high probability. The protocol will adapt autonomously until the upper bound is met.

Adopt Vote Extensions

Alternatively, the team should also consider adopting vote extensions as soon as its performance meets the project requirements.

Remediation Plan

Risk-accepted: The dYdX team is looking to accept vote-extensions as soon as possible (when the performance is acceptable).

Struct alignment issues

Title	Struct alignment issues
Project	dYdX: Rewards, Consensus and Other Changes Q3 2023
Type	IMPLEMENTATION
Severity	2 MEDIUM
Impact	2 MEDIUM
Exploitability	2 MEDIUM
Issue	

Involved artifacts

- structs defined in v4-chain/protocol/x/ files

Description

During the audit, it was observed that dYdX v4 code extensively utilizes structs. Subsequently, we employed linter tools, specifically the fieldalignment and betteralign which revealed that, in the majority of cases, the defined structs were not properly aligned.

The findings from the tool's analysis are partially documented in the Appendix: fieldalignment and betteralignment tool analysis.

With only a single object of each struct type and a single pass executed using the tools, applying all alignment suggestions can potentially reduce memory consumption from 11,528 pointer bytes to 9,312 pointer bytes, representing an optimization of approximately 19%. Considering that a single order could be reduced from 80 pointer bytes to 24, and given the likelihood of a large number of orders in the CLOB, there is a potential for a more significant severity issue to arise in the product.

Problem Scenarios

Depending on the number of objects created for each struct type in dYdX v4, memory consumption may significantly increase due to misaligned struct fields. This has the potential to impact performance by increasing the frequency of peaks within the Go garbage collector, and it can lead to higher memory resource costs. To accurately assess the impact, comprehensive benchmarking is necessary.

Recommendation

We recommend considering to refactor the structs defined in the dYdX code using a '*fieldalignment process*,' which involves reordering and aligning the fields within the structs to minimize their memory footprint.

The memory consumption depends on the system's architecture (32-bit, 64-bit) and struct padding.
System architecture defines the size (in bits) of each *word* and how the memory of the system is aligned. This defines the sizes of the basic types in Golang. Struct padding purpose is to improve the performance of memory

usage by padding - adding extra bytes to struct so it’s size is a multiple of *word* size.

Alignment will speed up memory access by creating code that requires one instruction to read and write to a memory location.
Benchmarking can be conducted with more realistic data volumes based on statistical information from dYdX v3, where applicable. It may also be necessary to perform multiple passes of the 'alignment' tool analysis to ensure maximum memory optimization. Furthermore, after aligning the structs, performance analysis should be undertaken, as in rare instances, the most compact order may not always be the most efficient. In such cases, it could lead to two variables, each updated by its own goroutine, occupying the same CPU cache line, resulting in memory contention known as 'false sharing' that can slow down both goroutines.

Remediation Plan

Risk-accepted: The dYdX team currently has no plans to implement the optimization suggested with this finding, since they are focusing on other performance improvements that have potentially higher gains.

Appendix: Vulnerability Classification

For classifying vulnerabilities identified in the findings of this report, we employ the simplified version of Common Vulnerability Scoring System (CVSS) v3.1, which is an industry standard vulnerability metric. For each identified vulnerability we assess the scores from the *Base Metric Group*, the Impact score, and the Exploitability score. The *Exploitability score* reflects the ease and technical means by which the vulnerability can be exploited. That is, it represents characteristics of the *thing that is vulnerable*, which we refer to formally as the *vulnerable component*. The *Impact score* reflects the direct consequence of a successful exploit, and represents the consequence to the *thing that suffers the impact*, which we refer to formally as the *impacted component*. In order to ease score understanding, we employ CVSS Qualitative Severity Rating Scale, and abstract numerical scores into the textual representation; we construct the final *Severity score* based on the combination of the Impact and Exploitability sub-scores.

As blockchains are a fast evolving field, we evaluate the scores not only for the present state of the system, but also for the state that deems achievable within 1 year of projected system evolution. E.g., if at present the system interacts with 1-2 other blockchains, but plans to expand interaction to 10-20 within the next year, we evaluate the impact, exploitability, and severity scores wrt. the latter state, in order to give the system designers better understanding of the vulnerabilities that need to be addressed in the near future.

Impact Score

The Impact score captures the effects of a successfully exploited vulnerability on the component that suffers the worst outcome that is most directly and predictably associated with the attack.

Impact Score	Examples
High	Halting of the chain; loss, locking, or unauthorized withdrawal of funds of many users; arbitrary transaction execution; forging of user messages / circumvention of authorization logic
Medium	Temporary denial of service / substantial unexpected delays in processing user requests (e.g. many hours/days); loss, locking, or unauthorized withdrawal of funds of a single user / few users; failures during transaction execution (e.g. out of gas errors); substantial increase in node computational requirements (e.g. 10x)
Low	Transient unexpected delays in processing user requests (e.g. minutes/a few hours); Medium increase in node computational requirements (e.g. 2x); any kind of problem that affects end users, but can be repaired by manual intervention (e.g. a special transaction)
None	Small increase in node computational requirements (e.g. 20%); code inefficiencies; bad code practices; lack/incompleteness of tests; lack/incompleteness of documentation

Exploitability Score

The Exploitability score reflects the ease and technical means by which the vulnerability can be exploited; it represents the characteristics of the vulnerable component. In the below table we list, for each category, examples of actions by actors that are enough to trigger the exploit. In the examples below:

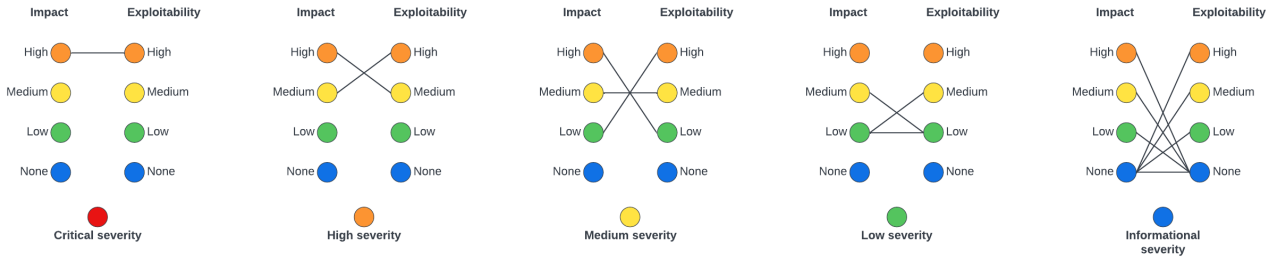
- *Actors* can be any entity that interacts with the system: other blockchains, system users, validators, relayers, but also uncontrollable phenomena (e.g. network delays or partitions).
- *Actions* can be

    - *legitimate*, e.g. submission of a transaction that follows protocol rules by a user; delegation/redelegation/bonding/unbonding; validator downtime; validator voting on a single, but alternative block; delays in relaying certain messages, or speeding up relaying other messages;
    - *illegitimate*, e.g. submission of a specially crafted transaction (not following the protocol, or e.g. with large/incorrect values); voting on two different alternative blocks; alteration of relayed messages.
- We employ also a *qualitative measure* representing the amount of certain class of power (e.g. possessed tokens, validator power, relayed messages): *small* for < 3%; *medium* for 3-10%; *large* for 10-33%, *all* for >33%. We further quantify this qualitative measure as relative to the largest of the system components. (e.g. when two blockchains are interacting, one with a large capitalization, and another with a small capitalization, we employ *small* wrt. the number of tokens held, if it is small wrt. the large blockchain, even if it is large wrt. the small blockchain)

Exploitability Score	Examples
High	illegitimate actions taken by a small group of actors; possibly coordinated with legitimate actions taken by a medium group of actors
Medium	illegitimate actions taken by a medium group of actors; possibly coordinated with legitimate actions taken by a large group of actors
Low	illegitimate actions taken by a large group of actors; possibly coordinated with legitimate actions taken by all actors
None	illegitimate actions taken in a coordinated fashion by all actors

Severity Score

The severity score combines the above two sub-scores into a single value, and roughly represents the probability of the system suffering a severe impact with time; thus it also represents the measure of the urgency or order in which vulnerabilities need to be addressed. We assess the severity according to the combination scheme represented graphically below.



As can be seen from the image above, only a combination of high impact with high exploitability results in a Critical severity score; such vulnerabilities need to be addressed ASAP. Accordingly, High severity score receive vulnerabilities with the combination of high impact and medium exploitability, or medium impact, but high exploitability.

Severity Score	Examples
Critical	Halting of chain via a submission of a specially crafted transaction

Severity Score	Examples
High	Permanent loss of user funds via a combination of submitting a specially crafted transaction with delaying of certain messages by a large portion of relayers
Medium	Substantial unexpected delays in processing user requests via a combination of delaying of certain messages by a large group of relayers with coordinated withdrawal of funds by a large group of users
Low	2x increase in node computational requirements via coordinated withdrawal of all user tokens
Informational	Code inefficiencies; bad code practices; lack/incompleteness of tests; lack/incompleteness of documentation; any exploit for which a coordinated illegitimate action of all actors is necessary