



# **Security Audit Report**

Stride 2024 Q2: Trade Route, Batching  
(Un)Delegations

Authors: Marko Juric, Mirel Dalcekovic

Last revised 24 June, 2024

# Table of Contents

<b>Audit Overview .....</b>	<b>1</b>
The Project	1
Conclusions	1
<b>Audit Dashboard .....</b>	<b>2</b>
Target Summary	2
Engagement Summary	2
Severity Summary	2
<b>System overview .....</b>	<b>3</b>
dYdX Trade Routes	3
Swap Flow	5
Batching Delegations and Undelegations	8
<b>Threat inspection .....</b>	<b>9</b>
Threat #1: Current design could not support more Trade Routes (currently there is only one Trade Route for dYdX)	9
Threat #2: Stride liquid staking flows could be impacted or blocked by the inability of Trade Route(s) performing swaps to native (DYDX) tokens	10
Threat #3: Rewards amount swap speed should be carefully designed and analyzed because of its potential impact on the final amount of DYDX rewards reinvested.	12
Threat #3.1: Monitoring of the Trade Route	12
Threat #3.2: Impacting on max trade size and the total amount of swapped and reinvested DYDX tokens	12
Threat #4: Impact on the Redemption Rate (RR)	14
Threat: #5: Possibility of front-running with off chain agent solution	15
Threat #6: Consequences if the exec account is compromised	15
Threat #7: Economic considerations of swaps performed	16
Batching of (Un)Delegations - Code Correctness and Specification Alignment Review	17
<b>Findings .....</b>	<b>18</b>
Exec Account on Trade Zone Capable of Performing Any Token Pair Swaps if Compromised	20
Custom PFM ACK Processing Instead of or in Addition to Monitoring	23
Exec Account Could Be Drained of OSMO Tokens Needed for Swap Fees	24
Possible Accumulation of USDC rewards on Trade ICA account due to off chains script logic	25
Possibility of Front-Running	27

Trade Route Economic Considerations	29
Incorrect Status Update for Referenced Epoch EXIT_TRANSFER_QUEUE HZU Records	31
Iterative Update of HZU Records Despite totalNativeTokensUnbonded Reaching Zero	33
<b>Appendix: Vulnerability Classification .....</b>	<b>36</b>
Impact Score	36
Exploitability Score	36
Severity Score	37

# Audit Overview

## The Project

In May 2024, Stride continued collaboration with [Informal Systems](#) for a partnership and security audit of the following scope:

- dYdX Trade Route - standard security code review;
- Batching of (un)delegations - code review for decided and implemented design approach.

Given that the final scope was determined during the audit project, the auditing and the Stride teams agreed on the priorities and the services to be provided, taking into account the constraints of time and available personnel.

As a result, we decided to allocate some of our time to consulting on potential design options for:

- stTIA multisig to ICA migration
- Batching of undelegations

## Conclusions

dYdX Trade Route was audited with standard Informal Security methodology and resulted in several informational and couple of higher severity issues as presented in the [Findings](#). It's crucial to emphasize the **utilization of PFM (Packet Forward Middleware) as a vital component in the Trade Route design**. To our knowledge, **PFM has never undergone an audit**.

Batching delegations and undelegations code inspection resulted in one critical and informational severity issues.

There were changes made to the `x/stakeibc` module since the previous audit of main flows: Delegation and Redemption, and considering there were changes introduced in the meantime and during this audit - it would be beneficial to cover main flows with threat models and systematic code review approach in the next partnership phase.

# Audit Dashboard

## Target Summary

- **Type:** Protocol and Implementation
- **Platform:** Go
- **Artifacts:**
  - dYdX Trade Route over commit hash: [6c53680](#)
  - Batching Delegations over commit hash: [a1a242b](#)
  - Batching Undelegations over commit hash: [df49379](#)

## Engagement Summary

- **Dates:** 09.05.2024. - 10.06.2024.
- **Method:** code review

## Severity Summary

Finding Severity	#
Critical	1
High	0
Medium	1
Low	3
Informational	3
<b>Total</b>	8

## System overview

### dYdX Trade Routes

dYdX host zone staking rewards are accrued in USDC instead of DYDX denomination, so Stride needs to convert - swap USDC to DYDX first, in order to reinvest collected rewards amount and use it in the regular reinvestment flow.

Initial solution was running entirely on-chain - conversion mechanism was using epochs and `x/authz` grantee account controlled by the ICA's account.

The current solution utilizing epochs and an off-chain swaps controlling script, was introduced due to:

- the increased TVL from a large dYdX community pool and
- possibility of front - running with keeping everything transparent and visible during consensus.

Setup:

1. **Host zone must be registered** in `x/stakeibc` - (in a context of dYdX trade route, this is dYdX host zone) with:
  - a. WITHDRAWAL ICA
2. **Register trade route** in `x/stakeibc` with gov proposals - `CreateTradeRoute` (code [ref](#)). Trade route is defined with all the details about the round trip through all the zones for performing the swap:
  - a. creating two new ICA accounts:
    - i. REWARD/UNWIND ICA (on Noble)
    - ii. TRADE ICA (on Osmosis)
  - b. transfer channels between the non-Stride chains are being tracked (dydx → noble, noble → osmosis, osmosis → dydx)

```
// TradeRoute represents a round trip including info on transfer and how
// to do
// the swap. It makes the assumption that the reward token is always
// foreign to
// the host so therefore the first two hops are to unwind the ibc denom
// enroute
// to the trade chain and the last hop is the return so funds start/end in
// the
// withdrawal ICA on hostZone
// The structure is keyed on reward denom and host denom in their native
// forms
// (i.e. reward_denom_on_reward_zone and host_denom_on_host_zone)
```

TradeRoute:

```
// ibc denom for the reward on the host zone
RewardDenomOnHostZone: String

// should be the native denom for the reward chain
RewardDenomOnRewardZone: String

// ibc denom of the reward on the trade chain, input to the swap
RewardDenomOnTradeZone: String

// ibc of the host denom on the trade chain, output from the swap
HostDenomOnTradeZone: String
```

```

// should be the same as the native host denom on the host chain
HostDenomOnHostZone: String

// ICAAccount on the host zone with the reward tokens
// This is the same as the host zone withdrawal ICA account
HostAccount: ICAAccount

// ICAAccount on the reward zone that acts as the intermediate
// receiver of the transfer from host zone to trade zone
RewardAccount: ICAAccount

// ICAAccount responsible for executing the swap of reward
// tokens for host tokens
TradeAccount: ICAAccount

// Channel responsible for the transfer of reward tokens from the host
// zone to the reward zone. This is the channel ID on the host zone
side
HostToRewardChannelId: String

// Channel responsible for the transfer of reward tokens from the
reward
// zone to the trade zone. This is the channel ID on the reward zone
side
RewardToTradeChannelId: String

// Channel responsible for the transfer of host tokens from the trade
// zone back to the host zone. This is the channel ID on the trade
zone side
TradeToHostChannelId: String

// Minimum amount of reward token that must be accumulated before
// the tokens are transferred to the trade ICA
MinTransferAmount: Int

// Deprecated, the trades are now executed off-chain via authz
// so the trade configuration is no longer needed
//
// specifies the configuration needed to execute the swap
// such as pool_id, slippage, min trade amount, etc.
TradeConfig: TradeConfig // Deprecated: Do not use

```

- ICA TRADE account will grant the exec account with appropriate trade executor permissions through the `authz` module to perform swaps on the trade zone (on Osmosis) with `ToggleTradeController` (code [ref](#))  
Creator of the `MsgToggleTradeController` should be a system admin.

```

// Grants or revokes trade permissions to a given address via authz
type MsgToggleTradeController struct {
    // Message signer (admin only)
    Creator: String
    // Chain ID of the trade account

```

```
ChainId: String
// Permission change (either grant or revoke)
PermissionChange AuthzPermissionChange
// Address of trade operator
Address: String
}
```

Trade routes can also be updated `UpdateTradeRoute` (code [ref](#)) and deleted `DeleteTradeRoute` (code [ref](#)) with governance proposals.

Besides the listed messages, there is also a `RestoreInterchainAccount` (code [ref](#)), used for restoring the ICA accounts, reverting all the actions in progress with changing the records in the state.

## Swap Flow

### Business Logic

All the USDC rewards collected during one epoch should be swapped until next epoch starts - ideally. In order to send the USDC rewards to Trade Zone (Osmosis chain) for swaps, **PFM - Packet Forward Middleware** is utilized as a crucial component for propagating the packet through Reward Zone (Noble chain).

The amount of rewards is split to smaller amounts and processed during the duration of one epoch. The processing (swapping) is triggered by an off chain script, that swaps these smaller amounts on randomized times. Safeguards are implemented by:

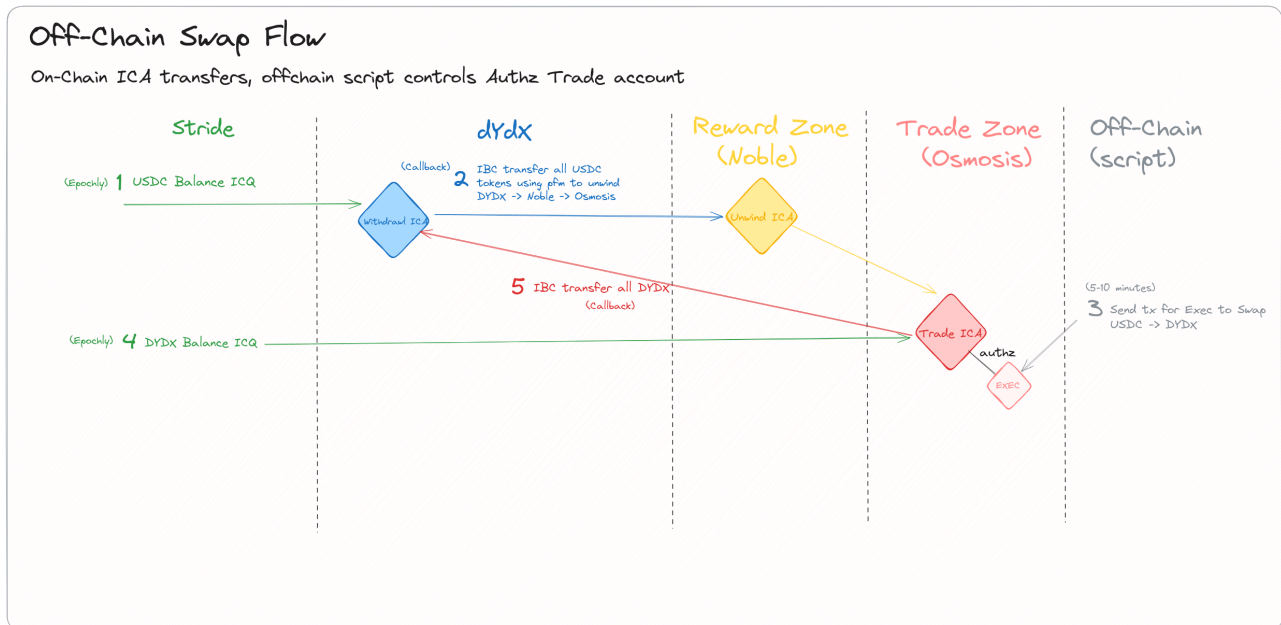
- limiting the slippage of any trade to 1% and
- limiting the maximum size of any trade to be less than 500 USDC.

### On chain logic:

1. USDC rewards are accrued in the `WITHDRAWAL` account (the account used for all `x/stakeibc` host zone rewards)
2. At the top of the epoch, an ICQ is submitted to query the balance of USDC rewards in the `WITHDRAWAL` account - `TransferAllRewardTokens: WithdrawalRewardBalanceQuery` (code [ref1](#), [ref2](#))
3. In the callback of that ICQ - `WithdrawalRewardBalanceCallback` (code [ref](#)), the rewards are transferred through Noble (the Reward zone and through `UNWIND` account) to Osmosis (the Trade zone, and to the `TRADE` account) using PFM (to make this multihop transfer atomic) - `TransferRewardTokensHostToTrade` (code [ref](#)). Swapping must go through Noble to unwind the ibc denom trace. PFM transfer message is built with `BuildHostToTradeTransferMsg` (code [ref](#)).
4. Once the USDC is in the `TRADE` account, the **off chain agent** will swap it to DYDX in batches every 5-10 minutes, as explained with the section below.
5. The next epoch, an ICQ is submitted for the balance of DYDX in the `TRADE` account - `TransferAllRewardTokens: TradeConvertedBalanceQuery` (code [ref](#))
6. In the callback of that ICQ, the DYDX is transferred back to the `WITHDRAWAL` account (code [ref](#)) - `TradeConvertedBalanceCallback` (code [ref](#)).

Since there are now native DYDX tokens in the `WITHDRAWAL` account, those tokens will be naturally reinvested via the normal `x/stakeibc` reinvestment flow, as if they had accrued directly.





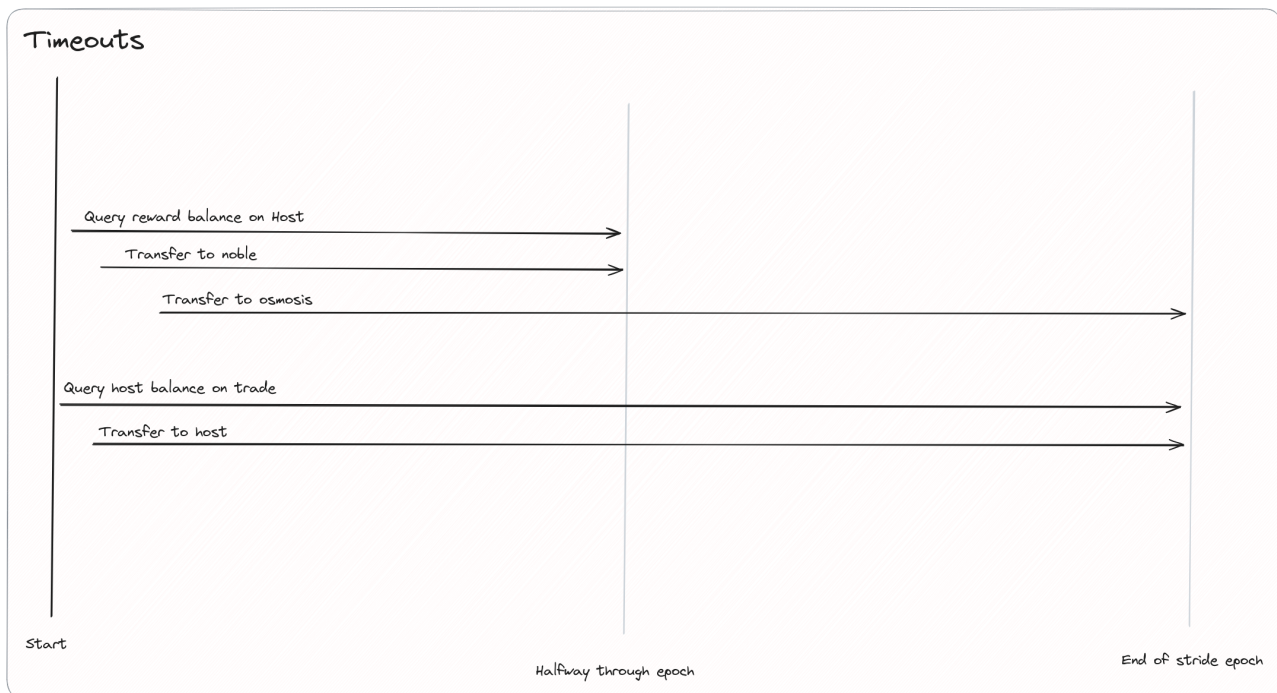
## Off-chain agent logic

(copied from the hand-off doc, since no access to the code):

Step by Step (line numbers reference `dydx_rewards.py`)

1. The trade handler is executed every 5 minutes
2. [L275] A random delay of 0-45 seconds is observed (to avoid being too predictable and being front run).
3. [L288] The start of the current epoch is calculated. Based on the current epoch and the amount of time that has passed since the start of the epoch, a conservative (lower bound) number of remaining trades for the rest of the epoch is determined.
  - a. [L292] The current USDC balance in the trade is divided by the number of remaining trades to determine a target trade size for this execution
  - b. Some safety checks follow
4. [L296-307] If the target trade size is greater than the hardcoded max trade size of 500 USDC, the target trade size is reduced to that value. This increases the likelihood of better swaps being executed at the tradeoff that funds might accumulate across epochs. If funds in the account accumulate, monitoring would alert us and this hardcoded threshold can be increased based on the rewards inflows and available liquidity.
5. [L309-328] If the simulated slippage on the target trade size is too high, an error is thrown and trade execution is halted. Manual intervention is needed to assess the liquidity in the pool
6. [L330] If the target trade size is smaller than a hard coded trade size of 50 USDC, the target trade size is increased to this value. This prevents excessive fees from impacting the overall execution due to many trivial swaps occurring.
7. [L338;L177] Once a target trade size is determined and passes all the sanity checks, the swap is executed through osmosis
  - a. [L192] A max slippage of 1% is enforced by passing a min out parameter to the swap command
  - b. For all osmosis commands executed, a handler ``exec_tx.execute_tx`` will query for the transaction hash and raise an alert if it isn't successful

## Timeouts



When creating the PFM memo, timeout is [defined](#) as:

```
// Timeout the first transfer halfway through the epoch, and the second transfer at
// the end of the epoch
// The pfm transfer requires a duration instead of a timestamp for the timeout, so we
// just use half the epoch length

halfEpochDuration := strideEpochTracker.Duration / 2
transfer1TimeoutTimestamp := uint64(strideEpochTracker.NextEpochStartTime -
halfEpochDuration) // unix nano
transfer2TimeoutDuration := fmt.Sprintf("%ds", halfEpochDuration/
1e9) // string in seconds
```

## Monitoring in place

The Stride team has set up monitoring system in order to detect any system errors and edge cases not processed as expected during reward swaps performed over Trade Routes. The monitored categories are as listed:

- **Summary monitoring** - showing trades that happened every 5-10 mins, what was the average price achieved compared to the CoinGecko price and daily sum of traded amount.
- **dYdX WITHDRAWAL ICA balance - showing the amount of USDC** in the Withdrawal account on the Host zone - if there is more than a certain amount, this means that entire flow has broken (maybe some of the channels are broken, or transfers are not happening for some reason)
- **dYdX WITHDRAWAL ICA balance - showing the amount of DYDX**. If there is more than certain amount of DYDX tokens, that means that there is an issue with the normal staking flow. For a specific DYDX reinvestment flow, there is a rebate calculated, stride fee is reduced from the total amount and then the remaining amount is staked.
- **USDC/DYDX TRADE ICA balance:**
  - to much USDC → swaps are not happening
  - to much DYDX → swaps are happening, but final transferring to the WITHDRAWAL ICA is not working
- **DYDX swap price execution:** comparison of prices on CEX (CoinGecko) and DEX osmosis pool prices.

## Batching Delegations and Undelegations

Due to gas limits, there is a fixed number of delegation and undelegation messages that can be submitted in a single transaction for each blockchain. To support a validator set of over 100 validators, Stride needs to batch these messages. Each batch includes the validator address and the amount delegated or undelegated, with appropriate callbacks defined. Transactions for delegations and undelegations are sent to each host zone using Inter-Chain Accounts (ICAs).

### Undelegation Batching Context

For undelegations, an additional constraint is that each validator can only process up to seven undelegations simultaneously. If an undelegation fails, it cannot be immediately reattempted to unbond from the same validator. Specifically, each unbonding record can only initiate one undelegation per validator at a time.

The redemption flow is more complex, so the cascading approach used will be explained at a high level in relation to the undelegation process:

Cascading callbacks and ICA transaction acknowledgements and timeouts update:

- the remaining undelegation amounts - in case of ACK success and
- the status - in case of ACK failures, timeouts and ACK success - once all the undelegation amount is triggered for undelegation

for each HZU records processed during each epoch. Each callback indicating the amount of undelegated funds reduces the total undelegation amount, starting with the oldest record. Once the amount in the oldest record is reduced to zero, the process continues to the next record. When a record's amount is fully reduced to zero, its status is changed to `EXIT_TRANSFER_QUEUE`, and the unbonding time is updated to 21 days from that point. Additionally, all stTokens in that HZU record are burned.

If the ICA transactions fail or timeout, the record is marked for retry *the next day*.

### Batching Process and Retry Mechanism

The batching process ensures that failed delegation or undelegation attempts can be retried in the next epoch. Each record processed for delegation or undelegation is marked with an appropriate status to track its progress. Additionally, a counter tracks the number of ICA transactions in progress, and this counter is updated based on number of ICA txs sent out and acknowledgments and timeouts received from the host zones.

### Handling Failures and Timeouts

In case of transaction timeouts, there is a mechanism to restore the ICA channel. Failed delegation or undelegation batches are handled by a rebalancing process, which ensures that the overall delegation remains balanced despite occasional retry failures.

### Key Points

1. **Gas Limits:** Only a fixed number of messages can be submitted per ICA transaction.
2. **Batching:** Messages are batched to support large validator sets (100+).
3. **Constraints:** Undelegations are limited to 7 per validator and cannot be retried immediately on failure (Cosmos SDK constraint).
4. **Status Tracking:** Records are marked to track delegation and undelegation progress.
5. **Retry Mechanism:** Failed batches are retried in the next epoch.
6. **Rebalancing:** Ensures overall delegation remains balanced despite retry failures.
7. **ICA Transactions:** A counter tracks ICA transactions in progress and is updated based on outcomes.

This system ensures efficient and balanced management of delegations and undelegations, accommodating large validator sets while addressing constraints and handling failures through retries and rebalancing.

## Threat inspection

### Threat #1: Current design could not support more Trade Routes (currently there is only one Trade Route for dYdX)

- **Invariant:** Stride liquid staking supports more trading routes for different rewards collected in non native tokens.

Currently there is only one Trade Route defined for the DYDX rewards, collected in the USDC. Design and code should support more Trade Routes for different host zones, if rewards are accrued in non native tokens and need swapping prior to the reinvestment.

### Threat inspection to perform

- Are more trade routes supported with the implementation and design?
- Is there a possibility to support more pools for swapping the rewards to native token, for the same trade route?

## Conclusions

Code inspection proved that:

- **More Trade Routes are supported:**
  - Trade Routes are created with governance proposals.
  - At the end of each Stride epoch reward conversion is **triggered**; where all trade routes are **processed**.
  - Errors are only logged in cases that ICQ was **not submitted** for querying the reward balance for the host withdrawal account and query for the amount of converted balance was **not submitted**.
  - if queries were not submitted - callbacks will not be processed, meaning that with:
    - ICQ `WithdrawalRewardBalanceQuery` : no swap through Noble -> Osmosis will be triggered
    - ICQ `TradeConvertedBalanceQuery` : transfer of swapped rewards back to the host zone's Withdrawal ICA will not be triggered
  - Could there be a non deterministic reason for these ICQs to error?
    - (un)marshaling errors; validation of query should pass;
  - Why would the callbacks error out?
    - unmarshaling errors, validations should pass;
- **More trade pools for performing swaps per Trade Route could be supported:**
  - Currently, swaps through multiple pools can be implemented and driven by an off-chain agent. Previously, swaps were triggered by Trade ICA accounts ([diff PR](#)). Different pools can be used to reduce the number of swaps through a single pool, especially since protocol is constantly selling USDC and buying DYDX through just one pool, which depends greatly on the liquidity provided and could create price movement pressure in the traded pool. However, the complexity of the off-chain script and monitoring would increase with the careful balancing and decision-making required to determine which pool is used to perform the swap.

## Findings



## Threat #2: Stride liquid staking flows could be impacted or blocked by the inability of Trade Route(s) performing swaps to native (DYDX) tokens

- **Invariant:** Trade routes do not impact existing liquid staking: delegation, redemption and and reinvestment flows.

### Threat inspection to perform

Confirm that ICQ & PFM are correctly utilized:

- Memo field is populated correctly for forwarding the messages to other chains through PFM.
- Flow is not blocked with waiting the ACK from the trade route chains.
- Check if ICA is needed on Noble (middle hop) anymore since newer versions of PFM don't require it?
- Is/should stride handle the ACK after PFM transfer? If ACK fail is not handled, allow flow to continue?
- Timeouts and retries:
  - Are the timeouts properly set related to the stride epoch (code [ref](#)).
  - Why is number of retries set to 0?
- If there is an issue with blocked cross-chain flows, are all the mechanisms in place to detect and bypass them?

### Assumptions:

- *There will always be one honest and fully functional relayer.*
- *Relayer should try to send out packets before they timeout.*
- *Exec account triggering swaps on Trade zone chain contains sufficient amount of native tokens to perform swaps.*

### Conclusions

- PFM is integrated (used) correctly, potential changes and improvements:
  - use invalid recipient address instead of ICA. [Ref](#) to the PFM documentation explanation.
- PFM ACK processing: Generally without `memo` to handle, all handling by this module is delegated to ICS-020. ICS-020 ACK are written and parsed in any case (ACK are backwarded). [Ref](#) to the PFM documentation. Informal team concludes that potentially, Stride could benefit from custom ACK processing in the informational finding reported: [Custom PFM ACK Processing Instead of or in Addition to Monitoring](#)
- **Timeouts and number of PFM retries analysis:**
  - The retry parameter is set to 0 in order for protocol to know that by the end of the epoch the package has either succeed, failed or timeout.
- If the system has some issue - how could the protocol overcome it? Reasons that could lead to failures:
  - **Fees needed to be paid throughout the trading flow:**
    - Expensive relayers: Relayers are not being paid through the protocol. Regardless of this, it was discussed with the Stride team on how big are the trading route expenses and it was shared that this is not a big amount, but the dYdX relayers are having unpredictable fees, depending on the traffic.
    - Osmosis trading fees: Are being paid in Osmosis tokens. There was a funding of this account, and the Stride team should only make sure there is enough amount of OSMO tokens on the exec account triggering the swaps. This could be added as a monitoring category as explained in the [Exec Account Could Be Drained of OSMO Tokens Needed for Swap Fees](#).
  - **closed channels:** controller chain or relayers can open a closed channel. For ordered channels, one would be closed in case of a packet timeouts. There is a function implemented in Stride `x/stakeibc` API for restoring the ICA account (code [ref](#)).
- **Potential failure points:**
  - Funds are stuck on the Reward zone (Noble chain)?
    - This should not be possible with the PFM utilized for the middle hop, since it is performed atomically. Usage of PFM could be an issue for future chains used as middle hops in case they

are not integrating PFM and valid receiver address is used. In this case funds could be stuck on this address. However:

- with the Trade Route design the ICA account owned by the Stride is used as a receiver address. In case of not supporting PFM, funds would land on the ICA account and additional tx could be triggered in order to move those funds vs transfer simply failing and amount being reverted to the Host Withdrawal account in case of invalid bech32 address used as a receiver.
- trade routes are defined with governance proposals, so no chains should be used as middle hops if not integrating PFM.
- It was concluded that there is no downside with this approach, since it can support older versions of PFM, where the explicit existing address is required as the receiver, as well as the newer versions.
- Funds are stuck on the Trade zone (Osmosis chain)?
  - In both cases current monitoring Stride has in place will catch these issues.
    - prior to swap: meaning off-chain script did not trigger the swap
    - after the swap: meaning ICA transfer of swapped rewards is not initiated from the Stride chain.
- **PFM versions used across the flow?**
  - Middle hop is performed on the Noble chain integrating older version ( [github.com/cosmos/ibc-apps/middleware/packet-forward-middleware/v4\\_v4.1.2](https://github.com/cosmos/ibc-apps/middleware/packet-forward-middleware/v4_v4.1.2) ). Two important notes related to this is that:
    - PFM was never audited (to our knowledge).
    - Auditing team did not perform diff analysis of PFM versions used or PFM audit with this project.

## Findings

- [Custom PFM ACK Processing Instead of or in Addition to Monitoring](#)
- [Exec Account Could Be Drained of OSMO Tokens Needed for Swap Fees](#)

## Threat #3: Rewards amount swap speed should be carefully designed and analyzed because of its potential impact on the final amount of DYDX rewards reinvested.

**Ideally, all the rewards collected during one epoch are swapped until the start of the next reinvestment epoch.**

However, this is not strictly an invariant because the swapping process relies on various factors including off-chain script randomness, interactions with other blockchain networks, and IBC transfers managed by relayers.

Informal auditing team however thinks that Stride components should ensure that epochly collected USDC rewards are split and swaps are initiated in a way that all tx and IBC transfers could reach the dYdX Host zone, in order to be reinvested with the next epoch.

The Stride protocol should not be the reason for failing to achieve this ideal reinvestment flow **unless** there are some **economic considerations involved**.

### Threat #3.1: Monitoring of the Trade Route

Threat inspection to perform:

- Check if monitoring will catch all the problematic cases:
  - What if funds are stuck in the trade or reward zone? Does monitoring cover all the possible trade route failure points? Confirm there is monitoring of this situation and that non converted USDCs will be noticed.
  - What will happen if ICQ fails? What happens if forwarding fails?

### Conclusions

- Monitoring in place seems sufficient for detecting all the potential flow issues. One additional information, Informal team considers important, is monitoring of the time elapsed from the initialization of transfer on Stride until USDC funds land on Trade ICA on osmosis in order to detect faster if there are any issues.
  - However, we are not sure we this could be easily be tracked! We need to query (time of provided answer is not the time of transfer succeeded) or log once the funds land on this account (this means that osmosis code needs to be changed and osmosis logs tracked)
  - Additional possible improvements to the monitoring are shared within:
    - [Custom PFM ACK Processing Instead of or in Addition to Monitoring](#)
    - [Exec Account Could Be Drained of OSMO Tokens Needed for Swap Fees](#)

### Findings



### Threat #3.2: Impacting on max trade size and the total amount of swapped and reinvested DYDX tokens

- What is the average amount of USDC rewards per epoch?
- How will the Stride protocol distribute them during the swap?
- Is it possible that not all rewards are swapped within one epoch but due to off-chain script calculating number of swaps possible and since trade size is limited to 50 min - 500 max?
  - Will the frequency of swaps be higher if more swaps need to perform? Will this fit to swap being triggered every 5-10 minutes?



1. [L288] The start of the current epoch is calculated. Based on the current epoch and the amount of time that has passed since the start of the epoch, a conservative (lower bound) number of remaining trades for the rest of the epoch is determined.
  - a. [L292] The current USDC balance in the trade is divided by the number of remaining trades to determine a target trade size for this execution
  - b. Some safety checks follow
2. [L296-307] If the target trade size is greater than the hardcoded max trade size of 500 USDC, the target trade size is reduced to that value. This increases the likelihood of better swaps being executed at the tradeoff that funds might accumulate across epochs. If funds in the account accumulate, monitoring would alert us and this hardcoded threshold can be increased based on the rewards inflows and available liquidity.

## Conclusions

During the analysis of the off-chain agent logic (taking into account information provided to the auditing team) we have concluded:

- script could make sure that entire amount of the rewards contained by the Trade ICA account is split to the trade sizes and triggered with appropriate frequency, so that all the swaps could be finalized in a timely manner.
- Also, it was concluded that these parameters could be affected with the economic considerations taken into account prior the swaps triggered. Analysis and suggestions related to this are provided with the [Threat #7 conclusions](#).
- Currently, it is shared by the Stride team (quoted off-chain agent logic explained):

*This increases the likelihood of better swaps being executed at the tradeoff that funds might accumulate across epochs. If funds in the account accumulate, monitoring would alert us and this hardcoded threshold can be increased based on the rewards inflows and available liquidity.*

We have shared potential idea for off-chain agent/script improvement within the finding listed as result of this threat inspection.

## Findings

- [Prevent Accumulation of USDC rewards with an automatic Off-Chain Script adjustment](#)



## Threat #4: Impact on the Redemption Rate (RR)

$$\text{Redemption Rate} = \frac{(\text{Deposit Account Balance} + \text{Undelegated Balance} + \text{Tokenized Delegation} + \text{Native Delegation})}{(\text{stToken Supply})}$$

### Threat inspection to perform

- if PFM transfer keeps failing, USDC will not be converted to DYDX, how can this affect RR?
  - Can users make advantage of stuck swaps ? Can users make advantage of lower RR at the moments prior to swaps being triggered?

### Conclusions

- Low RR (due to stuck swaps) can impact on users to liquid stake at that moment, and undelegating once the swap is enabled.
- If more undelegations happen during stuck swaps, stToken supply will decrease as it will be getting burned. At the same time, no rewards will be transferred to the Deposit account balance - seems like RR which could make RR higher

During the discussion with Stride team it is concluded that even though the impact on the RR technically exists, it is not to be seen how profitably it would be for users to take advantage of possible stuck swaps. We also learned that anything less than a day that changes the RR is so small that makes the change negligible and is not providing a significant impact to RR. Even after a day, it doesn't seem likely to have a worst case scenario where a lot of users would immediately liquid stake (in case of discovering the swaps are stuck) and redeem when RR pops up since they would still need to wait 21+ days. On the other hand, this potential problem could also exist in the normal flow, where disruptions could be introduced by issues on relayer's side.

The best way to mitigate the risk is to monitor RR which Stride already does and is alerted if RR is having unusual changes.

### Findings



## Threat: #5: Possibility of front-running with off chain agent solution

One of the auditing team's tasks during this audit was to provide feedback on the design change introduced to the Trade Routes, where actual swaps would be triggered by an off-chain agent. The primary motivation for this change was to maintain the unpredictability of the swap transactions.

If the swaps were triggered at regular intervals by the on-chain defined epochs, it would be easier to predict them, potentially enabling front-running attacks.

There are few ways of front-running:

- validators could inject transactions once they detect the Stride swap tx on trade zone
- external users could observe the exact periods of time when swaps would be performed and try to make advantage on trade zone
- On dYdX Host zone, users could detect that there were trades held back, resulting in a lower RR (redemption rate) and conclude that it would be beneficial to liquid stake since the unblocked swapped rewards would impact the RR.

### Threat inspection to perform

- Think of possible ways to detect when will the swaps be performed. List the worst case scenarios and mitigation ideas under Stride team's control.

### Conclusion

It is concluded that there is a possibility of front running, but it is highly unlikely for an attacker to succeed or, in case he does, to make a higher impact on the Stride protocol. The analysis is presented in the form of a finding listed below.

#### Findings

- [Possibility of Front-Running](#)

## Threat #6: Consequences if the exec account is compromised

Is it possible to perform any trade other than the ones ICA Trade account granted the exec `x/authz` account?

### Threat inspection to perform

- Think of how the revenue from TradeICA could be extracted?
- How fast could this be noticed with monitoring?
- How bad would be the consequences?

### Conclusions

Currently there is no limitation to swap only USDC to DYDX. The exec account could actually swap anything. This poses a risk and we described potential worst case scenario and possible solution ideas in the finding listed below.

#### Findings

- [Exec Account on Trade Zone Capable of Performing Any Token Pair Swaps if Compromised](#)

## Threat #7: Economic considerations of swaps performed

### Economic considerations to provide feedback to

- Will the Stride protocol lose significant amount of rewards with swaps/rewards after reinvestment due to slower reinvestment?
- Should swaps be halted if the difference in the prices on CEX and osmosis pool differ too much?
- Could front-running impact the prices in the pool or only impact on user making profit?
- Could the pool potentially become illiquid? Will this be noticed by the Stride monitoring? How to solve this issue?
- Is it possible for Stride to move the value of DYDX token on CEX?

### Points and questions shared by the Stride team

#### *Economic security considerations:*

- We try to swap all transferred USDC in equally spaced trades by the end of the epoch when more USDC is likely to arrive in the TRADE ICA.
- We are doing all of these swaps through the single Osmosis pool 1426 which has about \$140k of trade volume a day, we are doing \$20k-\$30k all in the same sell-USDC-buy-DYDX direction.
- There have been times when the **DEX (osmosis1426) and CEX (CoinGecko) prices for DYDX have temporarily depegged. Our thought is unless it is extreme, we should continue trading.**
- Over time it is like dollar cost averaging the swings, and it is more important that we do not allow the pipeline to become backed-up without rewards accruing.
- **At some level though, we would need to pause trading because we would be wasting USDC revenue without getting appropriate DYDX back. How should we weigh this tradeoff?**
- The timing and amounts for the trades are semi-predictable. Before when trades were done on-chain, it was one large trade per epoch which had a chance to swing the price and incur slippage by itself. This offchain design allows us to split the epochly amount into many small trades with minute level timing randomness.

## Conclusions

This analysis was conducted to provide feedback on the economic considerations and strategies adopted by the Stride team and presented in the finding listed below.

## Findings

- [Trade Route Economic Considerations](#)

## Batching of (Un)Delegations - Code Correctness and Specification Alignment Review

### Code review to perform:

- Confirm that implementation is aligned with shared design (documents shared and additional information over slack).
- New design should enable delegation and undelegation in batches
- The most critical use case to be handled is failure of undelegation ICAs and including failed remaining undelegated amount to the next epoch unbonding processing
- Make sure that update of HZU records for the epochs processed is performed correctly
  - The status of the HZU follows expected transitions → if there are still funds to be undelegated, the status remains as last received status for the record (
  - Confirm that amount of `stTokensToBurn` and `nativeTokensToUndelegate` are decremented only when needed ACK success is received and undelegation was successful.
  - Confirm that retries will not be triggered if there is still ICA ACK to be retrieved for the HZU processed.

### Conclusions

The analysis resulted with two issues within batching undelegation implementation, one of which the development team as well noticed during the audit.

Auditing team suggested to re-audit the Delegation and Redemption (as well as `x/stakeibc` module implementation) as a whole once again due to changes introduced and planned with further scope.

### Findings

- [Incorrect Status Update for Referenced Epoch EXIT\\_TRANSFER\\_QUEUE HZU Records](#)
- [Iterative Update of HZU Records Despite totalNativeTokensUnbonded Reaching Zero](#)

## Findings

Title	Type	Severity	Status	Issue
Incorrect Status Update for Referenced Epoch EXIT_TRANSFER_Q UEUE HZU Records	IMPLEMENTATION	CRITICAL	RESOLVED	<a href="https://github.com/Stride-Labs/stride/commit/577fd8f2eb32c217b03942566221d07e63274c3b">https://github.com/Stride-Labs/stride/commit/577fd8f2eb32c217b03942566221d07e63274c3b</a>
Exec Account on Trade Zone Capable of Performing Any Token Pair Swaps if Compromised	PROTOCOL	MEDIUM	ACKNOWLEDGED	
Exec Account Could Be Drained of OSMO Tokens Needed for Swap Fees	PROTOCOL	LOW	DISPUTED	
Possible Accumulation of USDC rewards on Trade ICA account due to off chains script logic	PROTOCOL	LOW	ACKNOWLEDGED	
Possibility of Front-Running	PROTOCOL	LOW	ACKNOWLEDGED	
Custom PFM ACK Processing Instead of or in Addition to Monitoring	IMPLEMENTATION	INFORMATIONAL	DISPUTED	
Trade Route Economic Considerations	OTHER	INFORMATIONAL	ACKNOWLEDGED	

Title	Type	Severity	Status	Issue
Iterative Update of HZU Records Despite totalNativeTokens Unbonded Reaching Zero	IMPLEMENTATION	INFORMATIONAL	PATCHED WITHOUT RE_AUDIT	<a href="https://github.com/Stride-Labs/stride/commit/cd8d9765158f69c2ca21bc5251ef58f901d0cae3">https://github.com/Stride-Labs/stride/commit/cd8d9765158f69c2ca21bc5251ef58f901d0cae3</a>

## Exec Account on Trade Zone Capable of Performing Any Token Pair Swaps if Compromised

Project	Stride 2024 Q2: Trade Route and Batching (Un)Delegations
Type	PROTOCOL
Severity	MEDIUM
Impact	HIGH
Exploitability	LOW
Status	ACKNOWLEDGED
Issue	

### Involved artifacts

- [/x/stakeibc/keeper/msg\\_server.go](#)
- [/x/stakeibc/keeper/reward\\_converter.go](#)

### Description

Currently there is no limitation for exec account to be able to perform only USDC to DYDX swaps. The exec account could actually swap anything. This poses a risk.

- It is possible, if exec account is compromised that unfavorable swaps are performed.
- Currently x/authz module allows only generic authorizations to be defined. With this type of the authorization it is possible only to grant certain messages, but not the exact content of the message. With this specific grant case:

```
swapMsgTypeUrl := "/" + proto.MessageName(&types.MsgSwapExactAmountIn{})

switch permissionChange {
case types.AuthzPermissionChange_GRANT:
    authorization := authz.NewGenericAuthorization(swapMsgTypeUrl)
    expiration := ctx.BlockTime().Add(time.Hour * 24 * 365 * 100) // 100
years
    grant, err := authz.NewGrant(ctx.BlockTime(), authorization,
&expiration)
    if err != nil {
        return nil, errorsmod.Wrapf(err, "unable to build grant struct")
    }
    authzMsg = []proto.Message{&authz.MsgGrant{
        Granter: tradeRoute.TradeAccount.Address,
```

```

        Grantee: grantee,
        Grant:  grant,
    }}

    case types.AuthzPermissionChange_REVOKE:
        authzMsg = []proto.Message{&authz.MsgRevoke{
            Granter:    tradeRoute.TradeAccount.Address,
            Grantee:     grantee,
            MsgTypeUrl:  swapMsgTypeUrl,
        }}

```

## Problem Scenarios

Since creating a pool in osmosis is permissionless (the only condition is that the pool creation fee is paid), in the worst case scenario a malicious user could create a pool, define weights and swap-fee/exit-fee values such that it ensures profit. After providing the liquidity in the pool, after the swap is performed the user could exit the pool, basically “stealing” USDC tokens and use them further as more valuable.

## Recommendation

As discussed with the Stride team, there are several solutions possible:

### 1. Define custom authz authorization type for Stride purposes:

Update osmosis Cosmos SDK [x/authz](#) module to add type to limit which swaps could be performed. The specific grant type would limit the swaps to USDC<=>DYDX token denominations.

- After this, there is still an issue of swaps being executed with different amounts and timings than planned with the off-chain agent, if script is compromised.

### 2. Monitoring improvements: Monitoring currently tracks:

- **Summary monitoring: Each swap triggered** by the python off chain script, and provides summary of the USDC amount traded and DYDX amount retrieved with the price achieved. Improve monitoring to capture all the other trades that are triggered, with other token pairs or through other (osmosis) pools.



The screenshot shows a Stridebot alert titled "Stridebot APP 15 hours ago" with a table of swap transactions. The table has four columns: Timestamp, USDC In, DYDX Out, and Price. The data shows multiple swaps occurring between May 12, 2024, at 16:50:51Z and 17:30:27Z, with USDC In values around 50.62 and DYDX Out values around 24.60.

Timestamp	USDC In	DYDX Out	Price
2024-05-12T17:30:27Z	50.62	24.6024	\$2.06
2024-05-12T17:25:29Z	50.62	24.6061	\$2.06
2024-05-12T17:20:38Z	50.62	24.6164	\$2.06
2024-05-12T17:15:06Z	50.62	24.6043	\$2.06
...			
2024-05-12T17:10:31Z	50.62	24.6096	\$2.06
2024-05-12T17:00:39Z	50.62	24.6039	\$2.06
2024-05-12T16:55:26Z	50.62	24.6074	\$2.06
2024-05-12T16:50:51Z	50.62	24.6109	\$2.06
...			

This could be implemented as alert to go off in cases when other pairs are swapped through other pools. Stride team would then revoke any grants that exec account has, asap.

- **USDC/DYDX TRADE ICA balance** - monitoring could be changes/extended to list all the tokens present in the TRADE ICA account. It is not expected that this account holds any other tokens besides USDC, DYDX.
- ### 3. Consider changing this exec account to multisig account:
- **The time needed to collect all the signatures should be taken into the account (5-10 minutes frequency swaps).**
  - This could also be a way to additionally introduce randomness - since each signature could be retrieved with different pace and this could not be influenced.



- Probably Stride holds specific accounts on the osmosis, so those accounts could be used to provide multi-signature.

## Mitigation strategy

After the discussion with the Stride team, it was concluded that improvement of the monitoring would provide additional value and could prevent detecting too late that exec account was compromised.

- The Stride team would implement this.

Related to the multisig exec account idea, the implementation of the suggestion is currently not possible, since the stTIA multisig was implemented entirely through a manual process (there is a trusted operator providing signatures for a multisig account). This would mean that automatic process should be implemented, but once again there are issues related to the key storing and signing options - one machine, multiple machines, operators....

- The team considers this to have no ROI.

## Custom PFM ACK Processing Instead of or in Addition to Monitoring

Project	Stride 2024 Q2: Trade Route and Batching (Un)Delegations
Type	IMPLEMENTATION
Severity	INFORMATIONAL
Impact	NONE
Exploitability	NONE
Status	DISPUTED
Issue	

### Description

With current design, Stride team has established robust monitoring to track transfers and trades occurring along the dYdX trade route. The auditing team was exploring ways to enhance the existing monitoring system or implement automated mechanisms that would provide timely information about both successful and errored steps.

### Problem Scenarios

Stride protocol currently has no logs or information about errors that happen for PFM transfers.

### Recommendation

If Stride should process ACK received from PFM transfers with a custom middleware logic (additional layer to be introduced), the Stride protocol would contain additional info about ACK failures and logs, which would improve monitoring chances with tracking time needed to perform PFM transfers - success and ACK errors.

Both the auditing team and the Stride team agreed that these additional development efforts do not have a sufficient return on investment, as the existing monitoring is already detailed enough.

## Exec Account Could Be Drained of OSMO Tokens Needed for Swap Fees

Project	Stride 2024 Q2: Trade Route and Batching (Un)Delegations
Type	PROTOCOL
Severity	LOW
Impact	MEDIUM
Exploitability	LOW
Status	DISPUTED
Issue	

### Involved artifacts

- monitoring
- OSMO funds on the exec account

### Description

Trade Route swaps for the dYdX route are conducted on the Osmosis chain and paid in OSMO tokens. The Exec account, which has the permissions to execute these swaps, covers the fees.

### Problem Scenarios

While the account is initially funded with OSMO tokens, it is crucial to monitor the balance and issue an alert when it falls to an insufficient level, preventing further swaps from being triggered.

Currently, monitoring only detects that funds are accumulating in the Trade account after a swap fails due to insufficient funds to cover the fees.

### Recommendation

Query the present OSMO tokens amount on the exec account and alert if the funds become insufficient.

Following the discussion with the development team, it was confirmed that monitoring of the OSMO balance in the exec account is already established, which is why the status of this issue was updated to **DISPUTED**.

## Possible Accumulation of USDC rewards on Trade ICA account due to off chains script logic

Project	Stride 2024 Q2: Trade Route and Batching (Un)Delegations
Type	PROTOCOL
Severity	LOW
Impact	LOW
Exploitability	MEDIUM
Status	ACKNOWLEDGED
Issue	

### Involved artifacts

- swap amounts and frequency of swaps in `dydx_rewards.py`

### Description

With current design, swaps are performed carefully with several constraints in mind:

- swaps of higher amounts would move the price of DYDX and pose a front running attack risk
- swaps need to be performed with randomized frequency to prevent a front running attack

Goal is to reinvest as fast as possible having these two points in mind.

### Problem Scenarios

Stride team could decide on fixed parameters and the floating ones, depending on the time left until the Stride epoch and amount of the rewards on the Trade ICA account:

- fixed trade sizes - min amount 50 USDC and max amount 500 USDC
- frequency and random component - all the swaps will be triggered within 5 -10 mins of the previous one

The issue is that there is no optimal strategy an it is not possible to define one. A lot of external and non controlled factors are influencing defining one: amount of DYDX osmosis pool inflow, amount of USDC rewards for DYDX host zone, CEX price, slippage, epoch time remained....

If trading happens to fast it impacts the prices - on the other side, reinvestment speed is held down.

Currently, there is monitoring in place of accumulated funds on the Trade ICA account followed by the off chain script update: increasing the hardcoded thresholds.

## Recommendation

The auditing team's goal was to potentially define automatic approach of script's adjustment to the liquidity, price and epoch time left to perform swaps.

Since the liquidity of the osmosis pool (or any other pool that will be used for other trading routes) depends on other users as well, the off-chain script could query the amount of tokens in the pool at some constant frequency (less than 5 -10mins) and trigger with some additional randomized parameter once the amount of DYDX increases for some threshold amount of DYDX tokens.

## Possibility of Front-Running

Project	Stride 2024 Q2: Trade Route and Batching (Un)Delegations
Type	PROTOCOL
Severity	LOW
Impact	LOW
Exploitability	LOW
Status	ACKNOWLEDGED
Issue	

### Involved artifacts

- off-chain agent performing randomized swaps

### Description

The analysis was performed in order to evaluate and think of worst case scenarios on how the front-running could be performed with the current Trade Route design, where an off-chain script is utilized to trigger swaps on the Trade Zone (Osmosis chain).

#### Facts:

- Swaps are performed over one osmosis pool.
- Exec account performs swaps in the name of the ICA account.
- Swaps are triggered with some randomization with frequency 5-10minutes.
- Swaps are performed for the amounts no less than 50 USDC and no larger than 500 USDC with predefined slippage.

### Problem Scenarios

Auditing team thought of all the works case scenarios that would make the front-running possible and more easy, as well as possible ways to prevent those from happening.

#### Front-running strategies:

- Observe “randomized” swaps?
  - The swaps are, as shared by the Stride team, “little” randomized.
  - Malicious users could observe these transactions using a block explorer for performing chain activity analysis and detecting the exec account activity?
    - same pool used for swaps
    - swaps performed relatively with the same freq - not a public info related to the freq span (5-10min)
    - amounts - not a public info: 50 - 500 USDC swaps

It seems hard, but could be done with detecting same user, performing one direction swaps constantly (USDC → DYDX).

- Malicious users could run their own full nodes - which would give them access visibility of the transactions gossiped to the node and the opportunity to spot potentially profitable transactions here before they are confirmed. While this does not give them a view of all validators' mempools, it allows them to see unconfirmed transactions that the node is aware of.
- Malicious users, both full node and validators, could intentionally filter out the swap transactions from gossiping the further in p2p network. The impact is probably not big, since the proposer would need to be left without the tx in the mempool and this could only postpone its inclusion in the block.
- The malicious user could take advantage of blocked swaps and transfers back to host zone and undelegations, since both lead to reducing the RR values and liquid stake at the appropriate moment. After the trade flow and reinvestment is done, the user could undelegate and earn once the RR is higher.

## Recommendation

Theoretically the pattern could be caught by the malicious user, having enough funds to “waste” some of the swaps profit if not hitting the exact point in time.

As additional mechanisms for preventing front running attacks:

- rate limiting could be introduced - this would represent the restriction on the frequency of transactions from a single address. Rate limiting can prevent front-runners who often operate by flooding the network with rapid, successive transactions.
- batching of swaps could be performed
- sliding windows - chain could introduce a mechanism where specific orders would be not executed immediately, or would be batched...
- threshold decryption mempool

All of these mitigation strategies should be implemented on the Trade zone chain (in case of DYDX Trade Route - that is Osmosis).

Considering that Stride has:

- introduced randomness in triggering swaps and
- also uses bounded, limited swap size trades with predefined minimal slippage rate when calculating the expected tokens amount out.

We can not see that Stride could further protect from front-running on a Stride protocol level with both on and off chain logic.

After discussing the results of this analysis with the Stride team, we concluded that even if front running occurs, the impact would be minimal due to the limited size of the swaps performed by Stride.

## Trade Route Economic Considerations

Project	Stride 2024 Q2: Trade Route and Batching (Un)Delegations
Type	OTHER
Severity	INFORMATIONAL
Impact	HIGH
Exploitability	UNKNOWN
Status	ACKNOWLEDGED
Issue	

### Involved artifacts

- economic considerations influencing Trade Route design

### Analysis

The aim of the analysis was to offer feedback on the economic considerations that influenced the design choices and approach adopted for the dYdX Trade Route.

The trade-off between sizes of amount being swapped in order to quickly reinvest vs not impact the slippage and mitigating the risk of front running had to be made.

Potential decisions and consequences are:

- Impact on yield because of the later reinvestment - but should not have losses over longer period of time since monitoring would detect if the trade flow is backing up;
- Losing on swapped DYDX amounts due to price fluctuations - but these should never be greater than defined slippage;
- Trade sizes are selected to be 50 USDC min and 500 USDC max - in order to not impact the price more than 1% (expected max slippage during trades);

Performing a study on comparison of potential losses due to later reinvestment vs producing slippage and having losses due to those reasons could be valuable in order to come up with an engineering approach to the frequency of swaps and amounts of swaps, as well as when to halt trading if the prices are depegged. However, even after this study is performed, it is questionable if the algorithm would be an optimal strategy in 100% cases. Potential approach would be:

- Analysis of potentially accrued revenue due to the rewards collected if reinvested faster should be compared to the price fluctuations in osmosis supercharged pool used for swapping . This could be done with additional logic in the off-chain script:
  - compare the average of the accrued rewards for the potentially swapped amount - if larger or equal than current weight in the pool for USDC<>DYDX pool - perform the swap. If smaller, wait.



As mentioned, this is very hard to implement so that it works in 100% cases - it is highly likely that finding the perfect formula would take too much time and at the specific moment, once the issue with liquidity occurs this could mean permanent halt of trading.

We have concluded that Stride team made the correct-possible decisions when it comes to tweaking the trade parameters in order to perform the most profitable swaps. They have the possibility to easily change and adapt the off-chain script in case more tweaking is needed.

Find ways to **ensure greater liquidity** in the osmosis pool used for the trading.

- Current liquidity is not very high.
- There is a rebate for the rewards swapped to the DYDX transferred to the DYDX Community pool.
  - **All of the suggested are economic and business related decisions for Stride to make, auditing team researched what could be the potential ideas but those might not be the appropriate solutions to the problem:**
    - AEZ and [Hydro project](#) - potential partnerships ensuring the significant amount of the liquidity in the osmosis pool would solve the liquidity issue. Community pools could provide liquidity.
    - Users could also be incentivized to use the mentioned osmosis pool for USDC<=>DYDX trades and in return could receive stTOKEN airdrops.

Moving the price of the DYDX token

- Is it possible for Stride to move the value of DYDX token, taking in consideration that the size of the pool is relatively small comparing to CEX liquidity size? Depegging issue?
  - In case of depegging, the amount of DYDX could be transferred between CEX<=>DEX to align price.
  - Arbitrage bots could be created to perform these price fluctuations between CEX, DEX. However, the auditing team lacks the necessary expertise and is uncertain about the legal or practical feasibility of undertaking this task.

Front running and impact on the Osmosis pool

Every trade impacts the price movements in the pool, front runners will also make profit, but considering the liquidity in the pool - the impact should not be critical.

- Analysis of trades performed between the swaps triggered by the Stride off-chain agent and price movements could also be analyzed in order to conclude the overall activity in the pool.

Risk of pool becoming illiquid

Using multiple DEXes and different pools to perform swaps could be a solution from an engineering point of view, but the auditing team is not certain if there are any other DEX pools that could be used - Astroport, Avalanche, etc. The reasons why the selected Osmosis pool is currently the best (if not the only) option were discussed with the Stride team.

- Once again, providing enough liquidity in the pool is the solution.

# Incorrect Status Update for Referenced Epoch EXIT\_TRANSFER\_QUEUE HZU Records

Project	Stride 2024 Q2: Trade Route and Batching (Un)Delegations
Type	IMPLEMENTATION
Severity	CRITICAL
Impact	HIGH
Exploitability	HIGH
Status	RESOLVED
Issue	<a href="https://github.com/Stride-Labs/stride/commit/577fd8f2eb32c217b03942566221d07e63274c3b">https://github.com/Stride-Labs/stride/commit/577fd8f2eb32c217b03942566221d07e63274c3b</a>

## Involved artifacts

- [/x/stakeibc/keeper/icacallbacks\\_undelegate.go](#)
- [/x/records/keeper/epoch\\_unbonding\\_record.go](#)

## Description

In the `UndelegateCallback` function (code [reference](#)):

```
// Check for a failed transaction (ack error)
// Reset the unbonding record status upon failure
if ackResponse.Status == icacallbackstypes.AckResponseStatus_FAILURE {
    k.Logger(ctx).Error(utils.LogICACallbackStatusWithHostZone(chainId,
        ICACallbackID_Undelegate,
        icacallbackstypes.AckResponseStatus_FAILURE, packet))

    // Set the unbonding status to RETRY
    if err := k.RecordsKeeper.SetHostZoneUnbondingStatus(
        ctx,
        chainId,
        undelegateCallback.EpochUnbondingRecordIds,
        recordstypes.HostZoneUnbonding_UNBONDING_RETRY_QUEUE,
    ); err != nil {
        return err
    }
    return nil
}
```

within the `UpdateHostZoneUnbondingsAfterUndelegation` function, we loop through all `epochUnbondingIds`, and all the `hostZoneUnbondings` are set to `UNBONDING_RETRY_QUEUE` (code reference).

```
// Updates the status for a given host zone across relevant epoch unbonding record IDs
func (k Keeper) SetHostZoneUnbondingStatus(ctx sdk.Context, chainId string,
epochUnbondingRecordIds []uint64, status types.HostZoneUnbonding_Status) error {
    for _, epochUnbondingRecordId := range epochUnbondingRecordIds {
        k.Logger(ctx).Info(fmt.Sprintf("Updating host zone unbondings on
EpochUnbondingRecord %d to status %s", epochUnbondingRecordId, status.String()))

        // fetch the host zone unbonding
        hostZoneUnbonding, found := k.GetHostZoneUnbondingByChainId(ctx,
epochUnbondingRecordId, chainId)
        if !found {
            return errorsmod.Wrapf(types.ErrHostUnbondingRecordNotFound, "epoch
number %d, chain %s",
epochUnbondingRecordId, chainId)
        }
        hostZoneUnbonding.Status = status

        // save the updated hzu on the epoch unbonding record
        updatedRecord, err := k.AddHostZoneToEpochUnbondingRecord(ctx,
epochUnbondingRecordId, chainId, *hostZoneUnbonding)
        if err != nil {
            return err
        }
        k.SetEpochUnbondingRecord(ctx, updatedRecord)
    }
    return nil
}
```

## Problem Scenarios

If one of the ICA acknowledgments (ACK) returns as `FAILURE`, all referenced epoch HZU records will be marked with the status `UNBONDING_RETRY_QUEUE`.

Prior to the failure, there could have been records marked with `EXIT_TRANSFER_QUEUE` - if previous ICA callbacks successfully undelegated certain amounts of funds.

In this case, the `EXIT_TRANSFER_QUEUE` records will be overwritten with `UNBONDING_RETRY_QUEUE` status, which should not be the case. In general, the transition for the records is done in batches, meaning all referenced records are updated simultaneously.

## Recommendation

If there are HZU records marked as `EXIT_TRANSFER_QUEUE`, acknowledgments failures should not update the status.

## Iterative Update of HZU Records Despite totalNativeTokensUnbonded Reaching Zero

Project	Stride 2024 Q2: Trade Route and Batching (Un)Delegations
Type	IMPLEMENTATION
Severity	INFORMATIONAL
Impact	NONE
Exploitability	NONE
Status	PATCHED WITHOUT RE_AUDIT
Issue	<a href="https://github.com/Stride-Labs/stride/commit/cd8d9765158f69c2ca21bc5251ef58f901d0cae3">https://github.com/Stride-Labs/stride/commit/cd8d9765158f69c2ca21bc5251ef58f901d0cae3</a>

### Involved artifacts

- [/x/stakeibc/keeper/icacallbacks\\_undelegate.go](/x/stakeibc/keeper/icacallbacks_undelegate.go)

### Description

When the `UndelegateCallback` (code [ref](#)) is executed successfully, the statuses of epoch unbonding records and the amounts of `NativeTokensToUnbond` for the HZU will be updated (reduced). This reduction is based on the smaller of two amounts: the `totalNativeTokensUnbonded` from the callback or the `NativeTokensToUnbond` amount for the HZU (code [ref](#)):

```
for _, epochNumber := range epochUnbondingRecordIds {
    hostZoneUnbonding, found :=
k.RecordsKeeper.GetHostZoneUnbondingByChainId(ctx, epochNumber, chainId)
    if !found {
        return totalStTokensToBurn,
errorsmod.Wrapf(recordstypes.ErrHostUnbondingRecordNotFound,
        "host zone unbonding not found for epoch %d and %s", epochNumber,
chainId)
    }

    // Determine the native amount to decrement from the record, capping at the
amount in the record
    // Also decrement the total for the next loop
    nativeTokensUnbonded :=
sdkmath.MinInt(hostZoneUnbonding.NativeTokensToUnbond, totalNativeTokensUnbonded)
```

```

    hostZoneUnbonding.NativeTokensToUnbond =
hostZoneUnbonding.NativeTokensToUnbond.Sub(nativeTokensUnbonded)
    totalNativeTokensUnbonded =
totalNativeTokensUnbonded.Sub(nativeTokensUnbonded)

    // Calculate the relative stToken portion using the implied RR from the
record
    // If the native amount has already been decremented to 0, just use the full
stToken remainder
    // from the record to prevent any precision error
    var stTokensToBurn sdkmath.Int
    if hostZoneUnbonding.NativeTokensToUnbond.IsZero() {
        stTokensToBurn = hostZoneUnbonding.StTokensToBurn
    } else {
        impliedRedemptionRate :=
sdk.NewDecFromInt(hostZoneUnbonding.NativeTokenAmount).Quo(sdk.NewDecFromInt(hostZone
Unbonding.StTokenAmount))
        stTokensToBurn =
sdk.NewDecFromInt(nativeTokensUnbonded).Quo(impliedRedemptionRate).TruncateInt()
    }

    // Decrement st amount on the record and increment the tota
hostZoneUnbonding.StTokensToBurn =
hostZoneUnbonding.StTokensToBurn.Sub(stTokensToBurn)
    totalStTokensToBurn = totalStTokensToBurn.Add(stTokensToBurn)

    // If there are no more tokens to unbond or burn after this batch, iterate
the record to the next status
    if hostZoneUnbonding.StTokensToBurn.IsZero() &&
hostZoneUnbonding.NativeTokensToUnbond.IsZero() {
        hostZoneUnbonding.Status =
recordstypes.HostZoneUnbonding_EXIT_TRANSFER_QUEUE
    }

    // Update the unbonding time if the time from this batch is later than what's
on the record
    if unbondingTime > hostZoneUnbonding.UnbondingTime {
        hostZoneUnbonding.UnbondingTime = unbondingTime
    }

    // Persist the record changes
    if err := k.RecordsKeeper.SetHostZoneUnbondingRecord(ctx, epochNumber,
chainId, *hostZoneUnbonding); err != nil {
        return totalStTokensToBurn, err
    }

    k.Logger(ctx).Info(utils.LogICACallbackWithHostZone(chainId,
ICACallbackID_Undelegate,
    "Epoch Unbonding Record: %d - Setting unbonding time to %d", epochNumber,
unbondingTime))
}

```

## Problem Scenarios

When `UpdateHostZoneUnbondingsAfterUndelegation` is executed, if the

`totalNativeTokensUnbonded` reduces to zero (code [ref](#)), there is no need to loop further through the epoch numbers (code [ref](#)) and update records in vain with zero values, also records will be updated with new unbonding time (code [ref](#)).

## Recommendation

Exit the loop in case of `totalNativeTokensUnbonded` reduced to zero.





## 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
 <b>High</b>	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
 <b>Medium</b>	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)
 <b>Low</b>	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)
 <b>None</b>	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
 <b>High</b>	illegitimate actions taken by a small group of actors; possibly coordinated with legitimate actions taken by a medium group of actors
 <b>Medium</b>	illegitimate actions taken by a medium group of actors; possibly coordinated with legitimate actions taken by a large group of actors
 <b>Low</b>	illegitimate actions taken by a large group of actors; possibly coordinated with legitimate actions taken by all actors
 <b>None</b>	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
 <b>Critical</b>	Halting of chain via a submission of a specially crafted transaction



Severity Score	Examples
 <b>High</b>	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
 <b>Medium</b>	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
 <b>Low</b>	2x increase in node computational requirements via coordinated withdrawal of all user tokens
 <b>Informational</b>	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