



# Security Audit Report

dYdX 2024 Q2+: LP vault

Authors: Aleksandar Ljahovic, Josef Widder, Mirel Dalcekovic

Last revised 8 July, 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>Findings .....</b>	<b>3</b>
Unlimited number of vaults and layers can impact system performances	6
Potential state bloating may occur as a result of invalid vaults being included on the chain	8
Unnecessary computation when determining the order ids for cancellation	11
Incorrect metrics: unspecified vaults counted as active	13
Various minor code improvements	14
Undocumented decommissioning and recreation of vaults at the same height	16
Vaults containing zero shares and amount of equity are not decommissioned	17
Undocumented delayed depositing and creation of the vault	19
No differentiation between internally and externally cancelled orders	21
Shares should not be minted in case of depositing to a vault containing negative shares	23
Skip refreshing vault's orders for a referenced wound-down market	25
Potential outdated vault orders matching scenarios and mitigations in place	27
Vault module does not implement correct Genesis state and InitGenesis and ExportGenesis functions	30
QueryVaultResponse proto defines incorrect data types for Equity, Inventory and TotalShares	32
<b>Appendix: Vulnerability Classification .....</b>	<b>34</b>
Impact Score	34
Exploitability Score	34
Severity Score	35

# Audit Overview

## The Project

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

- LP Vaults

During this audit, the primary objectives were to identify potential flaws and improvements in the current design and implementation, with a particular focus on detecting possible panics, error handling, ensuring deposit flow correctness, evaluating system impact and dependencies, and confirming correctness related to shares calculation and quoting order strategy implementation.

## Conclusions

LP Vaults feature was audited with standard Informal Security methodology and resulted in several informational and couple of higher severity issues as presented in the [Findings](#).

The audited version of the vaults are established on the protocol layer but are not available yet for traders via the frontend. Subsequent releases will introduce additional functionalities, including withdrawals and performance enhancements. Overall, our conclusion is that the feature under ongoing development, and the audited version is intended for limited use case.

# Audit Dashboard

## Target Summary

- **Type:** Protocol and Implementation
- **Platform:** Go
- **Artifacts:**
  - x/vault module over release tag: [v5.0.0](#)

## Engagement Summary

- **Dates:** 03.06.2024. - 28.06.2024.
- **Method:** code review

## Severity Summary

Finding Severity	#
Critical	0
High	0
Medium	1
Low	6
Informational	7
Total	14

## Findings

Title	Type	Severity	Status	Issue
Potential outdated vault orders matching scenarios and mitigations in place	IMPLEMENTATION	MEDIUM	ACKNOWLEDGED	
Vault module does not implement correct Genesis state and InitGenesis and ExportGenesis functions	IMPLEMENTATION	LOW	RESOLVED	[TRA-446] include vault shares in genesis state
Unlimited number of vaults and layers can impact system performances	IMPLEMENTATION	LOW	ACKNOWLEDGED	
Potential state bloating may occur as a result of invalid vaults being included on the chain	IMPLEMENTATION	LOW	ACKNOWLEDGED	
Unnecessary computation when determining the order ids for cancellation	IMPLEMENTATION	LOW	ACKNOWLEDGED	
Skip refreshing vault's orders for a referenced wound-down market	IMPLEMENTATION	LOW	ACKNOWLEDGED	

Title	Type	Severity	Status	Issue
QueryVaultResponse proto defines incorrect data types for Equity, Inventory and TotalShares	IMPLEMENTATION	LOW	RESOLVED	[TRA-471] update shares in vault query response to use NumShares type  [TRA-380] use serializable int for equity and inventory in vault query
Incorrect metrics: unspecified vaults counted as active	IMPLEMENTATION	INFORMATIONAL	ACKNOWLEDGED	
Various minor code improvements	IMPLEMENTATION	INFORMATIONAL	ACKNOWLEDGED	
Undocumented decommissioning and recreation of vaults at the same height	DOCUMENTATION PROTOCOL	INFORMATIONAL	ACKNOWLEDGED	
Vaults containing zero shares and amount of equity are not decommissioned	IMPLEMENTATION	INFORMATIONAL	ACKNOWLEDGED	
Undocumented delayed depositing and creation of the vault	DOCUMENTATION PROTOCOL	INFORMATIONAL	ACKNOWLEDGED	
No differentiation between internally and externally cancelled orders	IMPLEMENTATION	INFORMATIONAL	ACKNOWLEDGED	

Title	Type	Severity	Status	Issue
Shares should not be minted in case of depositing to a vault containing negative shares	IMPLEMENTATION	INFORMATIONAL	ACKNOWLEDGED	

Unlimited number of vaults and layers can impact system performances

Project	dYdX 2024 Q2+: LP Vaults
Type	IMPLEMENTATION
Severity	LOW
Impact	MEDIUM
Exploitability	LOW
Status	ACKNOWLEDGED
Issue	

Involved artifacts

- </x/vault/types/params.go>

Description

Currently there are no explicit limitations defined with the protocol on how many vaults there could be defined in the system or number of layers defined with the parameters for the vault module.  
Vault module's parameters define strategy for the vaults and [default parameter values](#) are:

```
/ DefaultParams returns a default set of `x/vault` parameters.
func DefaultParams() Params {
    return Params{
        Layers:                2,                // 2 layers
        SpreadMinPpm:          10_000,           // 100 bps
        SpreadBufferPpm:       1_500,           // 15 bps
        SkewFactorPpm:         2_000_000,       // 2
        OrderSizePctPpm:       100_000,         // 10%
        OrderExpirationSeconds: 2,                // 2 seconds
        ActivationThresholdQuoteQuantums: dtypes.NewInt(1_000_000_000), // 1_000 USDC
    }
}
```



The only constraint that exists and is being checked within [Validate function](#) is:

```
// Layers must be less than or equal to MaxUint8.  
if p.Layers > math.MaxUint8 {  
    return ErrInvalidLayers  
}
```

Currently implemented version of LP Vaults holds several assumptions:

- Vaults would be used only by the dYdX committee - they will deposit and create vaults.
- There will be no governance proposals to change number of layers or other parameter values.

In future releases, additional features and improvements are planned for the upcoming versions of the LP Vaults.

## Problem Scenarios

Current implementation, without protocol-level constraints, enables any user to create a vault, allowing for an unlimited number of vaults. Additionally, governance proposals, if approved, could alter parameters in a way that may be harmful to the protocol.

There are no limitations on number of statefull orders than can be injected by existing vaults and this could impact the system performances and increase block production time.

## Recommendation

dYdX team has defined, as shared, soft limits for the currently planned users of the vault - committee. It is assumed and expected there will be no more than 20 vaults with only 2 layers defined. These soft limits are defined with one main goal to prevent the block production time to increase and limit number of statefull orders created by vaults.

We suggest defining hard limits with the protocol in order to ensure that expected system performances are not violated. Constraints needed are maximum number of vaults and layers per vault. This means that maximum possible values for these parameters should be defined on a protocol level. Those could be updated with governance proposals once the system performances are improved to support more vaults and layers - that is - statefull orders.

Potential state bloating may occur as a result of invalid vaults being included on the chain

Project	dYdX 2024 Q2+: LP Vaults
Type	IMPLEMENTATION
Severity	LOW
Impact	MEDIUM
Exploitability	LOW
Status	ACKNOWLEDGED
Issue	

Involved artifacts

- [/proto/dydxprotocol/vault/vault.proto](#)
- [/x/vault/types/msg\\_deposit\\_to\\_vault.go](#)
- [/protocol/x/vault/keeper/orders.go](#)

Description

As [defined](#) with the vault.proto

```
// VaultType represents different types of vaults.
enum VaultType {
    // Default value, invalid and unused.
    VAULT_TYPE_UNSPECIFIED = 0;

    // Vault is associated with a CLOB pair.
    VAULT_TYPE_CLOB = 1;
}

// VaultId uniquely identifies a vault by its type and number.
message VaultId {
    // Type of the vault.
    VaultType type = 1;

    // Unique ID of the vault within above type.
    uint32 number = 2;
}
```

With current design and implementation, vaults are created upon initial deposit made by a user providing specific vault Id.

Vault Id is consisted of VaultType and a number - unique id referencing the clob pair id.

Since currently, there is only one vault type expected, each deposit made should be for a vault of type other than `VAULT_TYPE_UNSPECIFIED`.

`ValidateBasic` function for `MsgDepositToVault`:

```
// ValidateBasic performs stateless validation on a MsgDepositToVault.
func (msg *MsgDepositToVault) ValidateBasic() error {
    // Validate subaccount to deposit from.
    if err := msg.SubaccountId.Validate(); err != nil {
        return err
    }

    // Validate that quote quantum is positive and an uint64.
    quoteQuantums := msg.QuoteQuantums.BigInt()
    if quoteQuantums.Sign() <= 0 || !quoteQuantums.IsUint64() {
        return errorsmod.Wrap(ErrInvalidDepositAmount, "quote quantum must be strictly positive and less than 2^64")
    }

    return nil
}
```

`MsgDepositToVault` could be triggered by any user and needs to be signed with the subaccount owner.

This would lead to:

- creating a vault - if non existing at the moment, new subaccount module address is defined and stored in the subaccounts KV store.
- creating a share entry for a user in the vault's KV share store.

## Problem Scenarios

Any user could create unspecified vault type ( `VAULT_TYPE_UNSPECIFIED` ) with very small amount of quantum as a deposit. When placing a deposit, vault will be created with `VaultId` - where there are no constraints on the Id number sent externally, by a potentially malicious user.

A malicious user - e.g. a competitor or user simply aiming to attack dYdX reputation, could create numerous vaults for non existing types and clob pair ids. This is not for free - the transaction fees and the minimum deposit would need to be created, but it could lead to the impact described with this finding.

Shares are processed with:

1. **BeginBlocker** - `DecommissionNonPositiveEquityVaults` function in order to delete non positive equity vaults. Since the equity will remain as initial and the number of user's shares is different than 0 - the vaults would remain in the system, leading to useless shares and subaccounts created for the vault not being deleted.
2. **EndBlocker** - `RefreshAllVaultOrders` function in order to cancel and place new orders. Since the vaults are referencing non existing clob pair ids and there are no perpetual positions created, they will be skipped during the processing (code ref). No new orders will be created and there are no orders for deletion.

```
// Skip if vault has no perpetual positions and strictly less than
`activation_threshold_quote_quantums` USDC.
    vault := k.subaccountsKeeper.GetSubaccount(ctx,
    *vaultId.ToSubaccountId())
    if vault.PerpetualPositions == nil || len(vault.PerpetualPositions) ==
0 {
        if
        vault.GetUsdcPosition().Cmp(params.ActivationThresholdQuoteQuantums.BigInt())
== -1 {
            continue
        }
    }
}
```

Potential consequences with increase of number of subaccounts representing the vaults without any use to the automatic injection of orders and with increase of number of shares are:

- **Store Size Increase:** Over time, as more subaccounts are created, the number of key-value pairs stored under `types.SubaccountKeyPrefix` will grow. If left unchecked, this can lead to a significantly bloated KVStore.
- **Performance Impact:** A bloated KVStore can impact read and write performance. Retrieving data ( `store.Get` ) may become slower as the number of key-value pairs increases, especially if the store is not efficiently managed or optimized. These stores are accessed from `BeginBlocker` and `EndBlocker` , which could lead to slowing down the block production time.
- **Storage Costs:** If the number of subaccounts and number of shares grows excessively, it could increase the overall storage requirements for running a node, potentially affecting node operators and validators.

Consequences are proportional to the number of subaccounts depositing to the vaults and the funds willing to be spent on this attack.

## Recommendation

Proper validations should be introduced:

1. Creating a vault of unspecified type should not be allowed. This check should be introduced in the `MsgDepositToVault` `ValidateBasic` function.
2. Creating a vault should be possible only if vault references existing clob pair id. This ensures that vault's subaccount will participate in the market as planned and implemented with automatic vault's order placement and cancelling. However, after discussing the feasibility of introducing this validation, the dYdX team representative mentioned that a vault might be necessary to create a clob pair with future design decisions.
3. Cover expected behavior with test cases in `TestMsgDepositToVault` and `abci_test.go` `TestEndBlocker` `TestBeginBlocker` to test thoroughly and document expected and allowed values.
4. Continuously monitor:
  - a. creation of vaults and amounts deposited, as well as total equity and
  - b. the size and performance of the KVStore.

# Unnecessary computation when determining the order ids for cancellation

Project	dYdX 2024 Q2+: LP Vaults
Type	IMPLEMENTATION
Severity	LOW
Impact	LOW
Exploitability	LOW
Status	ACKNOWLEDGED
Issue	

## Involved artifacts

- </protocol/x/vault/keeper/orders.go>

## Description

`RefreshVaultClobOrders` cancels orders from the previous block and initiates new orders for all vaults that hold shares and have sufficient equity exceeding the activation threshold. To cancel orders, a list of `orderIds` is required.

`OrderId` is created with (code [ref](#)):

- `SubaccountId` - equals to `vaultId`.
- `ClientId` - created with `GetVaultClobOrderClientId` (code [ref](#)): `sideBit | blockHeightBit | layerBits`
- `OrderFlags` - const identifier for long term orders.
- `ClobPairId` - clob pair id referenced with vault id.

```
return &clobtypes.Order{
    OrderId: clobtypes.OrderId{
        SubaccountId: *vault,
        ClientId:     k.GetVaultClobOrderClientId(ctx, side, uint8(layer)),
        OrderFlags:   clobtypes.OrderIdFlags_LongTerm,
        ClobPairId:   clobPair.Id,
    },
    Side:        side,
    Quantums:    orderSizeBaseQuantumsRounded,
    Subticks:    lib.BigRatRoundToNearestMultiple(
```

```
        orderSubticks,  
        clobPair.SubticksPerTick,  
        side == clobtypes.Order_SIDE_SELL, // round up for asks and down for  
bids.  
    ),  
    GoodTilOneof: goodTilBlockTime,  
}
```

## Problem Scenarios

One of the main goals is to keep block production time as consistent and small as possible. Heavy computation placed in `GetVaultClobOrders` and in `constructOrder` (code [ref1](#) - except the `params` and number of layers defined, [ref2](#)) is needed for the new order placement, but for the order cancellation it is enough to only prepare list of order ids or each vault to cancel.

## Recommendation

Considering that only `OrderIds` are needed from that vault for order cancellation we suggest creating a new function that bypasses all the computations currently performed in `GetVaultClobOrders` and retrieves only the `OrderIds`.

This would significantly improve performance and reduce gas consumption (no computations, fewer KV store calls, fewer object creations, etc.).

After discussions with the dYdX team, it was concluded that this issue will be mitigated in the following weeks. The code will be adapted, and the changes will be incorporated.

## Incorrect metrics: unspecified vaults counted as active

Project	dYdX 2024 Q2+: LP Vaults
Type	IMPLEMENTATION
Severity	INFORMATIONAL
Impact	NONE
Exploitability	NONE
Status	ACKNOWLEDGED
Issue	

### Involved artifacts

- </protocol/x/vault/keeper/orders.go>

### Description

Vaults are considered active if:

- The number of shares is greater than zero;
- The subaccount asset position of the vault exceeds `ActivationThresholdQuoteQuantums`.

The number of active vaults is monitored through metrics. Presently, the active vault counter increases irrespective of the vault type.

### Problem Scenarios

Since, currently it is possible to create `VAULT_TYPE_UNSPECIFIED` type of vault. These types of vaults would also be counted as active, even though no orders are quoted for those vaults. (code [ref](#)).

### Recommendation

Only vaults that place orders should be considered active.

## Various minor code improvements

Project	dYdX 2024 Q2+: LP Vaults
Type	IMPLEMENTATION
Severity	INFORMATIONAL
Impact	NONE
Exploitability	NONE
Status	ACKNOWLEDGED
Issue	

### Involved artifacts

- [/protocol/x/vault/keeper/shares.go](#)
- [/protocol/x/vault/keeper/vault.go](#)
- [/protocol/x/vault/keeper/orders.go](#)

### Description

Here is the list of aesthetic and minor code improvements and issues around logging found during the code inspection. 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.

- This part of the code be refined a bit - the logic in the if and else statements is almost identical, the only difference is in `ownerShares` argument (the 4th parameter in `SetOwnerShares` ), so this could be determined in the if/else statement. The function call and the check for `err != nil` can be placed below the new if/else.
- Improve code readability and perform minor optimization with reducing how many times and where the `subaccountId` string for a vault is constructed.
  - Currently the subaccount string is created at several points when depositing to the vault (code ref):
    - when minting shares (code ref) - after trying to get the vault equity (code ref);
    - when transferring/depositing funds to the vault (code ref) → Suggestion is to determine the vault subaccounts string once at the beginning of the `DepositToVault` function and propagate it further to `MintShares` as a parameter. This improves code readability by clarifying the point at which the subaccounts string is defined.
  - When preparing the list of order ids or exact order to cancel/place them in `GetVaultClobOrders` code ref:

```
vault := vaultId.ToSubaccountId()  
// Calculate leverage = open notional / equity.
```



```
equity, err := k.GetVaultEquity(ctx, vaultId)
```

`vault` could be propagated to `GetVaultEquity` - in order to skip doing the same thing of constructing the subaccount string for a vault code [ref](#):

```
func (k Keeper) GetVaultEquity(
    ctx sdk.Context,
    vaultId types.VaultId,
) (*big.Int, error) {
    netCollateral, _, _, err :=
    k.subaccountsKeeper.GetNetCollateralAndMarginRequirements(
        ctx,
        satypes.Update{
            SubaccountId: *vaultId.ToSubaccountId(),
        },
    )
    if err != nil {
        return nil, err
    }
    return netCollateral, nil
}
```

- When decommissioning vaults, improve readability code [ref](#) with defining subaccount and propagating it to `GetVaultEquity`:

```
// Get vault equity.
    vaultId, err :=
    types.GetVaultIdFromStateKey(totalSharesIterator.Key())
    if err != nil {
        log.ErrorLogWithError(ctx, "Failed to get vault ID from state
key", err)
        continue
    }
    equity, err := k.GetVaultEquity(ctx, *vaultId)
```

## Problem Scenarios

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

## Recommendation

As explained in the Description section.

## Undocumented decommissioning and recreation of vaults at the same height

Project	dYdX 2024 Q2+: LP Vaults
Type	<b>DOCUMENTATION</b> <b>PROTOCOL</b>
Severity	<b>INFORMATIONAL</b>
Impact	<b>NONE</b>
Exploitability	<b>NONE</b>
Status	<b>ACKNOWLEDGED</b>
Issue	

### Involved artifacts

- </protocol/x/vault/abci.go>
- [/x/vault/keeper/msg\\_server\\_deposit\\_to\\_vault.go](/x/vault/keeper/msg_server_deposit_to_vault.go)
- existing documentation shared with the auditing team

### Description

Vaults with negative equity will be decommissioned during `BeginBlocker` function.

New vaults are created during `DeliverTx` phase, after `BeginBlocker` is executed for each block.

### Problem Scenarios

Protocol would decommission needed vaults and currently it is possible for an external user to deposit and recreate the vault in the same block height.

This could be slightly confusing for users whose shares were deleted with the decommissioned step - but the vault still exists, with the new update equity state.

### Recommendation

The auditing team believes this issue could demotivate users and potentially drive them to other DeFi protocols for trading. After discussions with the dYdX team representatives, it was concluded that this is expected behavior. Therefore, the auditing team recommends clearly documenting this as expected behavior to ensure users are not surprised by the outcome.

## Vaults containing zero shares and amount of equity are not decommissioned

Project	dYdX 2024 Q2+: LP Vaults
Type	IMPLEMENTATION
Severity	INFORMATIONAL
Impact	NONE
Exploitability	NONE
Status	ACKNOWLEDGED
Issue	

### Involved artifacts

- </protocol/x/vault/keeper/vault.go>

### Description

In the `DecommissionNonPositiveEquityVaults` function, vaults containing zero or negative amount of shares are skipped from the decommissioning (code [ref](#)):

```
// Skip if TotalShares is non-positive.
if totalShares.NumShares.BigInt().Sign() <= 0 {
    continue
}
```

If the number of shares is positive and if vault contains zero or negative amount of the equity, it will be decommissioned at the end (code [ref](#)):

```
// Decommission vault if equity is non-positive.
if equity.Sign() <= 0 {
    k.DecommissionVault(ctx, *vaultId)
}
```

## Problem Scenarios

It is questionable whether or not it makes sense to have a vault present in the system with:

- positive equity but no shares or even negative amount of shares - shouldn't be possible with the regular use of LP vaults, but the withdrawal is still not implemented;
- the equity and shares are both equal to zero.

It is unclear what would be the purposes of such vaults.

## Recommendation

Decommissioning could serve as an effective automated mechanism to delete invalid and unwanted vaults and shares from the state. After discussions with the development team representative, it was agreed that this should be considered when designing and implementing the withdrawal functionality. At that point, the vaults would reduce their total number of shares, and the system should ensure that:

- Invalid vaults do not persist in the state. The total number of shares should never become negative. This is already ensured during the minting of shares via the `SetTotalShares` function (code [ref](#)). However, this safeguard should also be implemented for withdrawals by using the same function to maintain the total shares state.
- Empty vaults do not persist in the state. Whether empty vaults should persist in the state is still undecided. As discussed with the dYdX team representative, this needs to be determined during the withdrawal design phase and clearly documented.

Additionally, another property that needs to be upheld and should be considered when implementing the withdrawal functionality is:

*Total shares number in a vault ( `TotalSharesKeyPrefix` ) must be equal to the number of all individual owner's shares from the ( `OwnerSharesKeyPrefix` ).*

## Undocumented delayed depositing and creation of the vault

Project	dYdX 2024 Q2+: LP Vaults
Type	DOCUMENTATION PROTOCOL
Severity	INFORMATIONAL
Impact	NONE
Exploitability	NONE
Status	ACKNOWLEDGED
Issue	

### Involved artifacts

- </protocol/app/app.go>
- existing documentation shared with the auditing team

### Description

Governance module is the authority for the `x/delaymsg` module, so for all the other modules in the system, including the `x/vault` module - both `x/gov` and `x/delaymsg` modules are authorities (code [ref](#)).

```
app.VaultKeeper = *vaultmodulekeeper.NewKeeper(  
    appCodec,  
    keys[vaultmoduletypes.StoreKey],  
    app.ClobKeeper,  
    app.PerpetualsKeeper,  
    app.PricesKeeper,  
    app.SubaccountsKeeper,  
    []string{  
        lib.GovModuleAddress.String(),  
        delaymsgmoduletypes.ModuleAddress.String(),  
    },  
)  
vaultModule := vaultmodule.NewAppModule(appCodec, app.VaultKeeper)
```

This means that all the other externally sent messages could be delayed - including the `MsgDepositToVault`.

### Problem Scenarios

The potential delayed depositing and creation of the vaults should be documented.

## Recommendation

Clearly state that it is possible to delay depositing to the vault and creation of the vault.

## No differentiation between internally and externally cancelled orders

Project	dYdX 2024 Q2+: LP Vaults
Type	IMPLEMENTATION
Severity	INFORMATIONAL
Impact	NONE
Exploitability	NONE
Status	ACKNOWLEDGED
Issue	

### Involved artifacts

- [/protocol/x/clob/keeper/msg\\_server\\_cancel\\_orders.go](#)
- [/protocol/x/clob/keeper/msg\\_server\\_place\\_order.go](#)

### Description

For each placement and cancellation there is an event emitted for the indexer to track transactions sent. Vault quotes orders internally, and basically it performs replacements of the orders from the `EndBlocker`.

Placement orders performs checks if an order is internally created, and does not emit event in that case (code [ref](#)).

Cancellation orders event code does not make any difference (code [ref](#)):

```
// 4. Add the relevant on-chain Indexer event for the cancellation.
k.GetIndexerEventManager().AddTxnEvent(
    ctx,
    indexerevents.SubtypeStatefulOrder,
    indexerevents.StatefulOrderEventVersion,
    indexer_manager.GetBytes(
        indexerevents.NewStatefulOrderRemovalEvent(
            msg.OrderId,
            indexershared.OrderRemovalReason_ORDER_REMOVAL_REASON_USER_CANCELED,
        ),
    ),
)
```

## Problem Scenarios

Tracking internally created (vault injected) orders could be misleading and indexer will provide faulty information to number of placed or cancelled orders per each block.

This could be perceived as spamming of the chain, while the orders are injected internally by the system.

Additionally, performance enhancements could be achieved by abstaining from emitting events for internally produced orders.

## Recommendation

Introduce another type of events in case of vault orders.



## Shares should not be minted in case of depositing to a vault containing negative shares

Project	dYdX 2024 Q2+: LP Vaults
Type	IMPLEMENTATION
Severity	INFORMATIONAL
Impact	HIGH
Exploitability	NONE
Status	ACKNOWLEDGED
Issue	

### Involved artifacts

- </protocol/x/vault/keeper/shares.go>

### Description

A vault with a negative total shares amount should not exist in a system. Vault should either be non existing or existing with zero or positive number of shares.

### Problem Scenarios

Since the withdrawals are not implemented, the analysis whether or not this could be potentially possible situation could not be performed.

With only deposits being made, the number of shares in a vault can only increase. Regardless, the conclusion is that condition is problematic (code [ref](#)):

```
if !exists || existingTotalShares.Sign() <= 0 {  
    // Mint `quoteQuantums` number of shares.  
    sharesToMint = new(big.Int).Set(quantumsToDeposit)  
    // Initialize existingTotalShares as 0.  
    existingTotalShares = big.NewInt(0)  
} else {  
    ...  
}
```

## Recommendation

Do not mint shares for existing vaults if they contain negative total amount of shares. Return error, log this vault and potentially implement mechanisms for purging this invalid state.

## Skip refreshing vault's orders for a referenced wound-down market

Project	dYdX 2024 Q2+: LP Vaults
Type	IMPLEMENTATION
Severity	LOW
Impact	LOW
Exploitability	LOW
Status	ACKNOWLEDGED
Issue	

### Involved artifacts

- [/protocol/x/clob/keeper/clob\\_pair.go](/protocol/x/clob/keeper/clob_pair.go)
- </protocol/x/vault/keeper/orders.go>

### Description

It is possible to update the clob pair within the `DeliverTx` phase and perform the final settlement of the clob pair - market (winding down of the market is performed with the `UpdateClobPair()` function).

All the referenced orders will be removed.

- If there is a vault existing in the system, referencing the clob pair being wounded down - the cancellation of the orders will be impossible from the x/vault `EndBlocker` because the orders are already removed.
- Placement of the new orders from the vault's `EndBlocker` will be triggered but the orders will not pass the validation `validateOrderAgainstClobPairStatus` (code [ref](#)) because of the clob pair marked with status `ClobPair_STATUS_FINAL_SETTLEMENT`:

```
case types.ClobPair_STATUS_FINAL_SETTLEMENT:
    return errorsmod.Wrapf(
        types.ErrOrderConflictsWithClobPairStatus,
        "Order %+v disallowed, trading is disabled for clob pair with
status %+v",
        order,
        clobPair.Status,
    )
}

return nil
```

## Problem Scenarios

If there is a vault referencing the wounded down market (clob pair id) - refreshing of the orders would be triggered from the `EndBlocker` regardless of both cancellation and placement will be rejected.

This would affect the processing duration of the `EndBlocker`, impacting performance and increasing logs, because nodes will continue to encounter order placement errors if a vault exists on a wound-down market and orders are quoted every block.

## Recommendation

The protocol could optimize by skipping order refreshing for wound-down clob pairs (markets) [here](#). Utilizing retrieving the information about the existing clob pairs from the KV store, we can check the status of each market. Specifically, the implementation should exclude refreshing vaults that reference clob pairs with the status

`ClobPair_STATUS_FINAL_SETTLEMENT`.

This approach would streamline operations and improve efficiency in managing market closures and active vaults quoting orders within the protocol. This approach will also reduce error logs produced.

## Potential outdated vault orders matching scenarios and mitigations in place

Project	dYdX 2024 Q2+: LP Vaults
Type	IMPLEMENTATION
Severity	MEDIUM
Impact	HIGH
Exploitability	LOW
Status	ACKNOWLEDGED
Issue	

### Involved artifacts

- </protocol/x/vault/types/params.go>
- </protocol/x/vault/keeper/orders.go>

### Description

Order is considered outdated if:

- it continues to live after the expiration time or after the explicit mechanism of purging implemented in the `EndBlocker` ,
- it is created with an old/non updated price.

Refreshing vault orders is defined with the vault's quoting criteria - strategy. However, even after refreshing orders, they may still reflect outdated prices.

The use of stale prices depends on the price updates received from providers via the Oracle and could impact on vault losses. Additional scenarios include: Vaults losing money upon users continuously placing ask bids (keep buying from the vault) and followed with the price increases.

Since vaults should be considered as regular users with their subaccounts, the same risk and safety mechanisms are applicable (liquidations, withdrawals gating, speed and rate limits etc.).

**Default strategy parameters** are as defined (code [ref](#)):

```
// DefaultParams returns a default set of `x/vault` parameters.
func DefaultParams() Params {
    return Params{
        Layers:                2,                // 2 layers
        SpreadMinPpm:          10_000,           // 100 bps
        SpreadBufferPpm:       1_500,           // 15 bps
        SkewFactorPpm:         2_000_000,       // 2
        OrderSizePctPpm:       100_000,         // 10%
        OrderExpirationSeconds: 2,               // 2 seconds
        ActivationThresholdQuoteQuantums: dtypes.NewInt(1_000_000_000), // 1_000 USDC
    }
}
```

The key parameters for this analysis include:

- **Layers** : determining the number of orders a vault can quote, with this number included in the creation of a unique **orderId** (code [ref](#)):

```
orderId: clobtypes.OrderId{
    SubaccountId: *vault,
    ClientId:     k.GetVaultClobOrderId(ctx, side, uint8(layer)),
    OrderFlags:   clobtypes.OrderIdFlags_LongTerm,
    ClobPairId:   clobPair.Id,
},...
```

- **OrderExpirationSeconds** : setting the time value for **GoodTillBlockTime** ( GTBT ) when placing and canceling orders,

as these directly influence the expiration and explicit cancellation logic.

## Problem Scenarios

During analysis auditing team was looking for (additional, whereas to the already discussed) corner cases where the vault order would actually continue to live after it is expected to be expired or refreshed.

**Scenario 1:** Governance proposal is updating **layers** vault module parameter:

1. **Reduces layers value:** vault module will cancel only one ask and one bid! and there will be outdated orders remaining in the system but only until the order expiration kicks in (so one additional block at most, if block production speed is fast - 1sec)
2. **Increases layers value:** cancellation order ids are created for the non existing orders from the previous block, but will not introduce any issues, since the cancellation is triggered only if the order exists (code [ref](#)). No panics, no errors - only ignore.

**Scenario 2:** Governance proposal is updating **OrderExpirationSeconds** vault module parameter:

1. **Reduces OrderExpirationSeconds value:** vault module will create cancellation orders with GTBT depending on block speed production as defined (code [ref](#)):

```
err := k.clobKeeper.HandleMsgCancelOrder(ctx,
    clobtypes.NewMsgCancelOrderStateful(
```

```

    order.OrderId,
    uint32(ctx.BlockTime().Unix()+orderExpirationSeconds,
  ))

```

Since the cancellation is valid only if the **GTBT** for the cancellation is greater than **GTBT** of the order placed - the cancelling will not be valid and the order would continue to exist until the old expiration time (code [ref](#)):

```

if cancelGoodTilBlockTime < existingStatefulOrder.GetGoodTilBlockTime() {
    return errorsmod.Wrapf(
        types.ErrInvalidStatefulOrderCancellation,
        "cancellation goodTilBlockTime less than stateful order
goodTilBlockTime."+
        " cancellation %+v, order %+v",
        msgCancelOrder,
        statefulOrderPlacement,
    )
}

```

2. **Increases OrderExpirationSeconds value:** Without any impact.

## Recommendation

To mitigate the corner case and prevent instances where outdated orders impact vault losses, storing the old values of the parameters **layers** and **OrderExpirationSeconds** in the state, along with the governance proposal height, could be valuable. Cancellations at this “corner case” height would utilize the old parameter values, while new orders would adhere to the updated ones.

## Vault module does not implement correct Genesis state and InitGenesis and ExportGenesis functions

Project	dYdX 2024 Q2+: LP Vaults
Type	IMPLEMENTATION
Severity	LOW
Impact	MEDIUM
Exploitability	LOW
Status	RESOLVED
Issue	[TRA-446] include vault shares in genesis state

### Involved artifacts

- [/protocol/x/vault/types/genesis.pb.go](#)
- [/protocol/x/vault/genesis.go](#)

### Description

Genesis state is defined as (code [ref](#)):

```
// GenesisState defines `x/vault`'s genesis state.
type GenesisState struct {
    // The parameters of the module.
    Params Params `protobuf:"bytes,1,opt,name=params,proto3" json:"params"`
}
```

following with only parameters being exported and initialized during genesis.

### Problem Scenarios

The inability to export and initialize the current state of the vault module as necessary poses a significant risk of potential data loss. Moreover, this limitation makes it impossible to create backups of the state, further complicating data management and security measures.

### Recommendation

Implement proper genesis state for x/vault module containing vault shares and owner shares.

After discussing this issue with the dYdX team representative, it was shared that the issue was fixed on the main branch and that currently export feature is not used, but will be probably at one point. These were the reasons to lower the exploitability classification from High to Low, which resulted in Low severity classification for this issue.



The resolution of the issue was confirmed by reviewing the protocol-level changes implemented in the [pull request](#) containing the fix.

## QueryVaultResponse proto defines incorrect data types for Equity, Inventory and TotalShares

Project	dYdX 2024 Q2+: LP Vaults
Type	IMPLEMENTATION
Severity	LOW
Impact	MEDIUM
Exploitability	LOW
Status	RESOLVED
Issue	[TRA-471] update shares in vault query response to use NumShares type [TRA-380] use serializable int for equity and inventory in vault query

### Involved artifacts

- [/proto/dydxprotocol/vault/query.proto](#)
- [/protocol/x/vault/keeper/grpc\\_query\\_vault.go](#)
- [/protocol/dtypes/serializable\\_int.go](#)
- [/proto/dydxprotocol/vault/vault.proto](#)

### Description

Equity (code [ref](#)) and inventory (code [ref](#)) in a vault could be negative. Even though it is considered invalid state, number of total shares in a vault could also become invalid - if withdrawals are not implemented correctly.

Total number of shares in general should never be negative, but it is represented with custom dYdX defined big.Int type of data defined as `SerializableInt` (code [ref](#)) in `vault.proto` (code [ref](#)):

```
// NumShares represents the number of shares in a vault.
message NumShares {
  // Number of shares.
  bytes num_shares = 2 [
    (gogoproto.customtype) =
      "github.com/dydxprotocol/v4-chain/protocol/dtypes.SerializableInt",
    (gogoproto.nullable) = false
  ];
}
```

## Problem Scenarios

Queries would provide incorrect state information about vaults, since all the values listed: equity, inventory and total number of shares are represented as `Uint64` in `query.proto` file (code [ref](#)):

```
// QueryVaultResponse is a response type for the Vault RPC method.
message QueryVaultResponse {
  VaultId vault_id = 1 [ (gogoproto.nullable) = false ];
  dydxprotocol.subaccounts.SubaccountId subaccount_id = 2
    [ (gogoproto.nullable) = false ];
  uint64 equity = 3;
  uint64 inventory = 4;
  uint64 total_shares = 5;
}
```

and explicitly casted to `Uint64` (code [ref](#)):

```
return &types.QueryVaultResponse{
  VaultId:      vaultId,
  SubaccountId: *vaultId.ToSubaccountId(),
  Equity:       equity.Uint64(),
  Inventory:    inventory.Uint64(),
  TotalShares:  totalShares.NumShares.BigInt().Uint64(),
}, nil
```

Both `Vault` and `AllVaults` queries would return invalid `UIn64` values, since `QueryAllVaultsResponse` uses `QueryVaultResponse`.

```
type QueryAllVaultsResponse struct {
  Vaults      []*QueryVaultResponse `protobuf:"bytes,1,rep,name=vaults,proto3" json:"vaults,omitempty"`
  Pagination  *query.PageResponse      `protobuf:"bytes,2,opt,name=pagination,proto3" json:"pagination,omitempty"`
}
```

## Recommendation

Change data types in `QueryVaultResponse` proto definition: `equity` and `inventory` should be of `SerializableInt` type, while `total_shares` can be defined with `NumShares` type.

The resolution of the issue was confirmed by reviewing the protocol-level changes implemented in the pull requests containing fixes: [PR1](#), [PR2](#).





## 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:

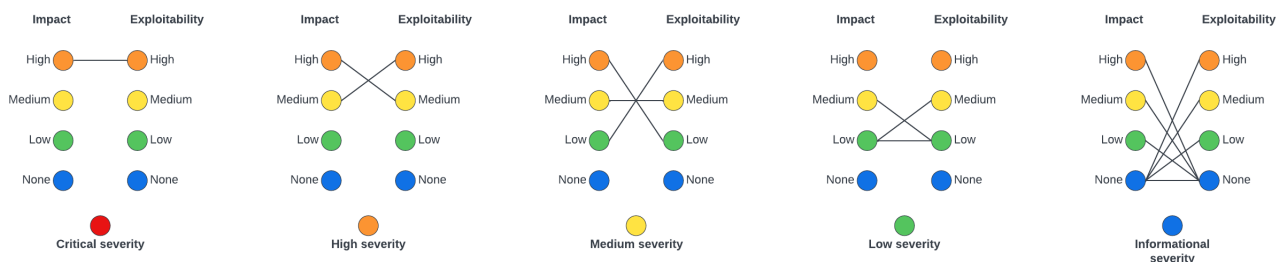
- *Actors* can be any entity that interacts with the system: other blockchains, system users, validators, relayers, but also uncontrollable phenomena (e.g. network delays or partitions).
- *Actions* can be

- *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