Audit Report

Comdex

v1.0

December 7, 2022

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

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

Introduction

Purpose of This Report

Oak Security has been engaged by Comdex to perform a security audit of Comdex's Asset, Vault, Collector, and Locker Cosmos SDK modules.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed in multiple stages on the following GitHub repositories:

https://github.com/comdex-official/comdex

Commit hash for asset, vault, collector, and locker modules:
`07007a346f669b9a885c1c9cb55b61576ca3b71a`

Commit hash for rewards, auction, tokenmint, bandoracle, market, and esm modules:
`a2d03e037491dc2779eafebfac1eb4dd14eaa24c`

Commit hash for the liquidity module:
`765f866950a1b994b8dabcfaf74a82c2f28914de`

NOTE: Due to a high number of critical issues found during our audit of the lend and liquidation modules, the client decided to exclude these modules from the scope of this audit

report, perform fundamental changes to their architecture and have them re-audited subsequently. Consequently, this audit report does not cover the lend and liquidation modules of the Comdex codebase. A separate audit report is available for those modules.

https://github.com/comdex-official/gov-contracts

Commit hash: `42a6c34ec541312d1213f9be3c29d2b636e12a0a`

Methodology

The audit has been performed in the following steps:

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

Functionality Overview

Comdex aims to deliver a robust infrastructure layer that supports seamless creation and deployment of DeFi applications in the Cosmos ecosystem. The Comdex chain enhances investor's access to a broad range of assets that help investors diversify and generate yield on their investments. This audit covers the functionality associated with the following Comdex modules: Asset, Vault, Locker, Collector, Rewards, Auction, Tokenmint, Bandoracle, Market, Liquidity and Esm as well as the Comdex Governance contracts.

As described in the Codebase Submitted for the Audit section above, the codebase was audited in three separate stages. Stage 1 consisted of the asset, vault, collector, and locker modules. Stage 2 covered the rewards, auction, tokenmint, bandoracle, market, and esm modules. And the final stage 3 covered the liquidity module as well as the Comdex Cosmwasm Governance contracts.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	Unbounded iteration allows attackers to attack validators, slowing down or even halting block production	Critical	Resolved
2	Transaction gas price is not related to execution complexity	Critical	Resolved
3	Vault ID collisions could lead to the loss, tampering or overwriting of existing vault data	Critical	Resolved
4	Locker ID collisions could lead to the loss, tampering or overwriting of existing locker data	Critical	Resolved
5	Users cannot redeem their collateral assets regardless of the ESM status	Critical	Resolved
6	`GetPriceForAsset` returns price of `2000000` for assets when the market is not found	Critical	Resolved
7	Computationally heavy operations in `BeginBlocker` and `EndBlocker` may slow down or stop block production	Critical	Acknowledged
8	Multi-purpose `BeginBlocker` error handling may disable critical functionality if an error is raised	Critical	Resolved
9	Error handling during slices scroll terminates the execution without executing the logic for all items	Critical	Resolved
10	An attacker could send `MsgRemoveMarketForAssetRequest` messages every 20 blocks in order to make the `Market` module unusable	Critical	Resolved
11	Permissionless `Rewards` module Whitelisting process allows attackers to manipulate the `Asset Whitelist` and `App Vault Whitelist`	Critical	Resolved
12	Gas is not consumed if the transaction returns an error	Critical	Acknowledged
13	CosmWasm - `State` query binding can perform a GRPC call to an arbitrary URL	Critical	Resolved

14	Error raised in `BeginBlocker` could lead to state corruption	Critical	Partially Resolved
15	CosmWasm - Deposited funds of other `denom` will be stuck in the contract	Critical	Resolved
16	`UpdateAssetsRecords` is not enforcing a unique `Denom` for an `Asset`	Major	Resolved
17	`AddAppRecords` may overwrite existing app name entries	Major	Resolved
18	`MsgAddWhiteListedAsset` allow any caller to add an asset id to a locker's whitelist	Major	Resolved
19	`AddAppRecords` may accept invalid `GenesisToken` array	Major	Resolved
20	Inverse pairs in `AddPairsRecords` can create duplicate issues for `StableMint` vaults	Major	Resolved
21	Whitelisted assets for rewards get overwritten when adding new assets, which may cause unexpected behavior	Major	Resolved
22	`HandleUpdateAdminProposal` does not properly handle admin update and admin is unchangeable as a result	Major	Resolved
23	Hard-coded admins increase the potential of unauthorized privileged activity	Major	Resolved
24	Base gas fee is insufficient for functions with iteration	Major	Acknowledged
25	Orders execution batching design could expose a surface for frontrunning and price manipulation	Minor	Acknowledged
26	Messages missing `ValidateBasic` will allow spam transactions to enter the mempool	Minor	Resolved
27	Unbounded iteration in `ValidateBasic` may cause node timeout	Minor	Resolved
28	Missing validation in `AddAppRecords` may cause unexpected behavior	Minor	Resolved
29	`Bank` keeper wrapper is not returning errors for zero values	Minor	Resolved
30	Hard-coded addresses inside functions are prone to human errors during upgrades and refactors	Minor	Resolved

31	CosmWasm - `beta` version dependencies shouldn't be used in production	Minor	Resolved
32	`CreateGauge` transactions are not checking `ESM` status or `killSwitch`	Informational	Acknowledged
33	Iterative calculations should be avoided in `BeginBlocker` when an analytic solution is available in order to reduce the function execution time	Informational	Resolved
34	Duplicated denom validation is inefficient	Informational	Resolved
35	Remove unused commented code blocks	Informational	Resolved
36	Spelling errors found in codebase	Informational	Resolved
37	Date ranges util function can return false negatives	Informational	Resolved
38	Use of magic numbers is prone to human errors	Informational	Resolved
39	`ValidateMsgCreateCreateGauge` performs redundant gauge validations	Informational	Resolved
40	`MsgCreatePair` message execution always returns an error	Informational	Resolved
41	Outstanding TODOs are present in the codebase	Informational	Resolved
42	Remove redundant validation logic	Informational	Resolved
43	CosmWasm - Overflow checks not enabled for release profile	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	Some room for improvement, due to highly nested loops and redundant validity checks.
Code readability and clarity	Low-Medium	Lack of comments for complex functionality negatively impacts the readability and clarity of the codebase.
Level of documentation	Low-Medium	The documentation has high variance of quality and completeness between different modules.
Test coverage	Low-Medium	17.0 % code coverage calculated with `go test`

Detailed Findings

1. Unbounded iteration allows attackers to attack validators, slowing down or even halting block production

Severity: Critical

App modules implement logic that needs to iterate through slices in order to find a specific element with the selected ID. This has the consequence of having unbounded loops with an asymptotic cost of `O(n)`. This is even worse when a loop is done over arrays that contain other arrays. In this case, the asymptotic cost is `O(n^2)` or `O(n^3)`.

Unbounded loops can allow two different types of attack:

1. **Validator slashing**

   A malicious actor could spam a transaction, that he knows performs an unbounded loop, to a particular node trying to force it to be not able to compute the transaction before the `BroadcastTxCommit` timeout.

   This can prevent that node from processing further ABCI messages such that it has to pause and contact peers to get the latest correct blocks.

   Also, the `NewTxTimeoutHeightDecorator` will discard all messages with an elapsed `heightTimeout`. This could be used in a particular event in order to manipulate it.

   As the validator was not able to sign the block, it implies that this event is decreasing its sign ratio. That could lead the validator to be slashed.

2. **Stop or slow down the block production**

   A malicious actor could spam the previous message to a set of validators. If a significant number of them hit the timeout and halt simultaneously, block production may stop or slow down.

Iterations vulnerable to this attack are:

- `x/asset/keeper/app.go:117`
- `x/asset/keeper/app.go:262`
- `x/asset/keeper/asset.go:160`
- `x/collector/keeper/collector.go:104`
- `x/locker/keeper/msg_server.go:238`
- `x/locker/keeper/msg_server.go:310`
- `x/vault/keeper/vault.go:47`
- `x/vault/keeper/vault.go:57`
- `x/vault/keeper/vault.go:347`
- `x/vault/keeper/vault.go:421`

- `x/vault/keeper/vault.go:441`
- `x/rewards/keeper/keeper.go:86`
- `x/rewards/keeper/keeper.go:156`

Recommendation

We recommend using a different data structure in order to reduce the asymptotic cost of some iterations. For instance, arrays could be reworked to maps in order to access them in `O(1)` instead of `O(n)`.

If in some cases this is not possible, we also recommend implementing a more efficient array search algorithm. For example, in sorted ID arrays, binary search could replace linear search in order to decrease the cost from `O(n)` to `O(log(n))`.

Finally, an upper limit could be imposed on arrays, which still comes with some inefficiency, but prevents attacks on the chain.

Status: Resolved

`x/asset/keeper/app.go:117` and `x/asset/keeper/app.go:262` are still affected but since they are executable only through governance and the impact is minimal, the Comdex team acknowledges these remaining issues.

2. Transaction gas price is not related to execution complexity

Severity: Critical

In `app/ante.go:47`, `NewConsumeGasForTxSizeDecorator` is taking care of calculating the gas cost of a transaction.

This decorator computes the gas cost of a transaction by multiplying the size of the transaction in bytes with `TxSizeCostPerByte`.

Even if this is a good heuristic to estimate the potential transaction cost, that's not enough. Transaction cost should be proportional to the computational complexity of its execution.

For instance, a transaction that has an asymptotic complexity of `O(1)` and that is not doing I/O operations, should not have the same cost as another one that has a `O(n)` complexity and is executing extensive I/O operations.

Messages vulnerable to this attack are:

- `MsgCreateLockerRequest` in `x/locker/handler.go:18`
- `MsgCreateRequest` in `x/vault/handler.go:18`
- `MsgWithdrawRequest` in `x/vault/handler.go:24`
- `MsgDrawRequest` in `x/vault/handler.go:27`
- `MsgRepayRequest` in `x/vault/handler.go:30`
- `MsgCloseRequest` in `x/vault/handler.go:33`
- `MsgCreateGauge` in `x/rewards/handler.go:19`

- `MsgMintNewTokensRequest` in `x/tokenmint/handler.go:21`
- `MsgDepositESM` in `x/esm/handler.go:18`

The current lack of accounting for execution complexity can make it affordable for an attacker to perform DoS spamming with expensive messages that have a small payload.

Recommendation

We recommend using `GasMeter.ConsumeGas` to charge gas proportional to the execution complexity in order to economically discourage attackers from spamming computationally expensive transactions.

Also, we recommend having a mechanism in place to charge gas for I/O transactions using a `GasKv` store.

Status: Resolved

3. Vault ID collisions could lead to the loss, tampering or overwriting of existing vault data

Severity: Critical

The `MsgCreate` function, in `x/vault/keeper/msg_server.go`, creates a new CDP or vault with a unique `vaultId`. In line `148`, an ID is assigned to the `newVault`, concatenating the app's `Shortname` and the `ExtendedPair`'s vault counter. However, this formula can lead to collisions, either accidental or intentional, which could cause a loss or tampering of existing vault data. Example:

- `ShortName1=Test1` and `updatedCounter=1`, with resulting `vaultId=Test11`
- `ShortName2=Test` and `updatedCounter=11`, with resulting `vaultId=Test11`

It should be noted that this also affects stable vault IDs.

Recommendation

We recommend implementing a collision-resistant formula when creating new vault IDs. A potential avenue for this would be using a collision-resistant hashing function on the apps `ShortName`, and concatenating the `updatedCounter` to the hashed result. An alternative could be usage of a separator character, which should be forbidden in the `ShortName`.

Status: Resolved

4. Locker ID collisions could lead to the loss, tampering or overwriting of existing locker data

Severity: Critical

Similarly to the vault ID collision issue, `lockerIds` are subject to collision due to the concatenation formula that is used to derive new `userLocker.lockerId`, in the function `MsgCreateLocker`, in `x/locker/keeper/msg_server.go:85`. This could lead to the loss or overwriting of key locker data, either accidentally or intentionally.

Recommendation

We recommend implementing a collision-resistant formula when creating new locker Ids. A potential avenue for this would be using a collision-resistant hashing function on the apps `ShortName`, and concatenating the `counter` to the hashed result. An alternative could be usage of a separator character, which should be forbidden in the `ShortName`.

Status: Resolved

5. Users cannot redeem their collateral assets regardless of the ESM status

Severity: Critical

The `if` condition in `x/esm/keeper/msg_server.go:71`, in the `MsgCollateralRedemption` handler, will never be executed, which blocks users from redeeming their collateral assets regardless of the ESM status. If the `esmStatus.status` is `True`, the condition in line `68` will be `True`, returning the `ErrCoolOffPeriodRemains`. And if the `esmStatus.status` is `False`, the condition in line `71` will be also `False`, preventing the execution of the if-block which contains the collateral redemption logic.

Recommendation

We recommend revisiting the if conditions in lines `68` and `71`, implementing a path for execution for user collateral redemptions when the desired `esmStatus.EndTime` and `esmStatus.Status` are fulfilled.

Status: Resolved

6. `GetPriceForAsset` returns price of `2000000` for assets when the market is not found

Severity: Critical

The `GetPriceForAsset` function in `x/market/keeper/oracle.go:175` returns a static price of `2000000` for assets when the market is not found. This is problematic because

instead of returning an error for this case, the `GetPriceForAsset` will silently return an incorrect value for the asset's price. This can have unintended consequences that may allow attackers to economically exploit the protocol.

Recommendation

We recommend returning `False` in `GetPriceForAsset` `x/market/keeper/oracle.go:175` when the market for an asset is not found so the calling context can handle the case where a price cannot be supplied for a specific asset.

Status: Resolved

7. Computationally heavy operations in `BeginBlocker` and `EndBlocker` may slow down or stop block production

Severity: Critical

`BeginBlocker` and `EndBlocker` are functions that are executed at the start/end of each block, even if there are no transactions. To not have a negative impact on block production, it should have a light and constant computational footprint.

In fact, it is wise to be cautious about adding too much computational weight at the start of each block, as blocks arrive at approximately seven-second intervals. Also, it should be a good practice to make the `BeginBlocker/EndBlocker` execution independent, or at least with a sub-linear dependency, from the amount of data stored on-chain.

A huge workload may slow down the block production, in the worst case so much that Tendermint's proposal timeout is surpassed.

The codebase implements multiple `BeginBlocker/EndBlocker` functions that are computationally heavy and/or depend on on-chain state:

- `x/auction/abci.go:8`
- `x/esm/abci.go:10`
- `x/market/abci.go:10`
- `x/rewards/abci.go:11`
- `x/liquidity/abci.go:12`
- `x/liquidity/abci.go:29`

Recommendation

We recommend reworking the mentioned `BeginBlocker/EndBlocker` functions in order to reduce their computational complexity.

Status: Acknowledged

Although the Comdex team has improved the efficiency of the iteration in `x/auction/abci.go:8`, they acknowledge that the `BeginBlocker` still contains unbounded iterations.

8. Multi-purpose `BeginBlocker` error handling may disable critical functionality if an error is raised

Severity: Critical

The `BeginBlocker` functions in:

- `x/rewards/abci.go:11`
- `x/auction/abci.go:8`

use an error-handling approach that stops the execution if one of the functions returns an error. This means that if the first function execution returns an error, the other ones that come after that one will not be executed.

While this can be a good approach for interdependent functions, it is not a good design in cases when functions are independent.

For example, in the `Rewards` module's `BeginBlocker`, `SurplusActivator` `DebtActivator` and `DutchActivator` are independent functions but an error in `SurplusActivator` will deny the execution of the other two.

Recommendation

We recommend implementing a different error handling mechanism in order to still collect errors and handle them but without stopping the execution of other independent functions. Instead, an event should be emitted describing the error so off-chain services can process errors and act accordingly.

Status: Resolved

9. Error handling during slices loop terminates the execution without executing the logic for all items

Severity: Critical

In

- `x/auction/keeper/surplus.go:27`,
- `x/auction/keeper/surplus.go:33`,
- `x/auction/keeper/dutch.go:56`,
- `x/auction/keeper/dutch_lend.go:49`,
- `x/auction/keeper/debt.go:26`,
- `x/auction/keeper/debt.go:32`, and
- `x/esm/abci.go:32-35`,

error handling logic during a slice iteration terminates the execution without iterating through all items.

This behavior could lead to incoherent information inside the slice and to disabled functionality if the loop triggers an error in the initial slice items.

This has the consequence of not checking all the items after the faulty one, and in the worst case, if the error is raised in the 0 position of the slice, to not check at all liquidation for collateralized positions.

Recommendation

We recommend using a different error handling mechanism that collects and handles errors without preventing the execution to loop the entire slice.

Status: Resolved

10. An attacker could send `MsgRemoveMarketForAssetRequest` messages every 20 blocks in order to make the `Market` module unusable

Severity: Critical

In `x/market/handler.go:25`, anyone is allowed to send a `MsgRemoveMarketForAssetRequest`.

An attacker could use this transaction to delete all the `Market` instances saved in the store, making the `Market module` unusable. The transaction can be executed every 20 blocks, which is the oracle data fetch interval.

Recommendation

We recommend removing the `MsgRemoveMarketForAssetRequest` messages or restrict the accounts that can send the transaction to the `Governance` or an `Administrator`.

Status: Resolved

11. Permissionless `Rewards` module `Whitelisting` process allows attackers to manipulate the `Asset Whitelist` and `App Vault Whitelist`

Severity: Critical

In, `x/rewards/handler.go`, `WhitelistAsset`, `RemoveWhitelistAsset`, `WhitelistAppIdVault` and `RemoveWhitelistAppIdVault` messages are permissionless.

Since those messages are responsible for the management of the `Asset Whitelist` and `App Vault Whitelist`, they should be executable only from privileged users or through governance.

In fact, having those messages executable by anyone makes the `Whitelist` process ineffective and exposes an attack surface.

For example a malicious actor could delete all `Assets` from the `AssetWhitelist` and make the module unusable, or could add an `Asset` to the whitelist that should not be allowed.

Recommendation

We recommend implementing an access control mechanism to execute the mentioned transactions.

Status: Resolved

12. Gas is not consumed if the transaction returns an error

Severity: Critical

The `ConsumeGas` function calls in lines:

- `x/liquidity/keeper/pool.go:182`,
- `x/liquidity/keeper/pool.go:255`,
- `x/liquidity/keeper/pool.go:313`,
- `x/liquidity/keeper/swap.go:123`,
- `x/liquidity/keeper/swap.go:233`,

- `x/liquidity/keeper/swap.go:292`,
- `x/liquidity/keeper/swap.go:362`,
- `x/liquidity/keeper/rewards.go:312`, and
- `x/liquidity/keeper/rewards.go:424`,

are located at the end of the transaction execution and are not called if an error occurs.

Consequently, a malicious actor is allowed to spam transactions that trigger an error in the middle of the execution without being charged of the defined gas fees.

Recommendation

We recommend moving mentioned `ConsumeGas` function calls to the beginning of the function to charge gas independent of the execution flow.

Status: Acknowledged

13. CosmWasm - `State` query binding can perform a GRPC call to an arbitrary URL

Severity: Critical

In `contracts/governance/src/msg.rs:12`, the `InstantiateMsg` is expecting a `target` field of type `String`.

This field is then used by the contract to perform `State` queries in

- `contracts/governance/src/contract.rs:128` and
- `contracts/governance/src/contract.rs:442`

in order to get the state at a particular `BlockHeight`.

The CosmWasm `State` query binding is resolved in the Cosmos SDK side by the `QueryPlugin` in `app/wasm/query_plugin.go:54` and handled by the `QueryState` function in `x/locker/keeper/locker.go:328`.

In `x/locker/keeper/locker.go:335` the `target` field propagated from the CosmWasm contract is used as URL to create a GRPC insecure connection. This mechanism is intended to work to call a Full Node but it has some possible problems:

- Every `target` URL is allowed so an attacker could use this to perform a `DOS` or to create a botnet using chain nodes to perform a `DDOS`.
- Exposed GRPC port could be not the same in different nodes as it can be configured by node operators, see https://docs.cosmos.network/master/core/grpc_rest.html#grpc-server, or it could be behind a Load Balancer, Reverse Proxy, Docker network, Ingress, etc.

- It is performing the call with `grpc.WithInsecure()` so there is no trust to the GRPC server contacted. It allows attacks like `MITM` https://grpc.io/docs/guides/auth/#with-server-authentication-ssltls
- Configurations and security enforcements rules through `iptables` or `ufw` or similar softwares on machines where the chain node is hosted could block this type of calls

That means that a CosmWasm contract is able to make the chain node perform a GRPC insecure connection to an arbitrary URL by simply using the `State` query binding.

Recommendation

We recommend not performing GRPC calls from CosmWasm contracts and to not rely in general on off-chain information like an `IP:PORT String` when executing on-chain logic.

Status: Resolved

14. Error raised in `BeginBlocker` could lead to state corruption

Severity: Critical

The modules that raise errors in `BeginBlocker` which could lead to a state where partial state changes are performed but other intended state changes are not committed. This function logs errors but has no control over the state and no mechanism to revert partial updates. This implies that if an error is raised in the middle of an execution flow, the execution will stop and log the error, but no action will be taken in order to revert state updates that have been done leading to state corruption.

Recommendation

We recommend implementing a caching mechanism letting the execution use a `CacheContext` and then if no errors are raised commit it to the actual state. This mechanism should be designed to segregate independent execution flows to be like atomic actions.

Status: Partially Resolved

The `bandoracle` and `market` modules are still affected but since the likelihood of this issue to happen is minimal, the Comdex team acknowledges these remaining issue instances.

15. CosmWasm - Deposited funds of other `denom` will be stuck in the contract

Severity: Critical

In `contracts/governance/src/contract.rs:534` in the `execute_deposit` message, if the voter has made a deposit to the given proposal before, the `Coins` that were previously added in the first deposit will be increased with the amount of the new deposit.

However, there is no check that `info.funds.len == 1`, so different denoms will be accepted and not accounted for, resulting in these funds remaining permanently stuck in the contract. The same issue occurs if a user only sends one coin of a certain denom to an existing proposal with a different denom. After the proposal has passed or been rejected, the `execute_refund` function in `570` will not return the coins whose denoms are different from the original deposit.

Recommendation

We recommend validating that the amount of `Coin` denoms sent with the `execute_deposit` message is equal to `1` and .that the denom sent matches the denom of the existing proposal.

Status: Resolved

16. `UpdateAssetsRecords` is not enforcing a unique `Denom` for an `Asset`

Severity: Major

The `UpdateAssetsRecords` function in `x/asset/keeper/asset.go:189` is not consistently handling the `Denom` to `Id` mapping when updating an `Asset` with a new `Denom` field.

Since `asset.Denom` is updated with `msg.Denom` in line `203`, the `DeleteAssetForDenom(ctx, asset.Denom)` in line `205` is not deleting the mapping with the old `Denom` but is instead trying to delete the mapping with the new one.

This has the consequences of not having a unique `Denom` for an `Asset Id` in the `Denom` to `Id` mapping.

Recommendation

We recommend flipping line `205` with line `203` in order to execute instructions in the correct order.

Status: Resolved

17. `AddAppRecords` may overwrite existing app name entries

Severity: Major

The `AddAppRecords` function in `x/asset/keeper/app.go:208-210` incorrectly performs an app name validation that will allow for the existing app Id to be overwritten in `SetAppForName`. In line `208` the `HasAppForName` check is passed `msg.ShortName` rather than `msg.Name`, so in effect the name is not checked currently.

Recommendation

We recommend changing the parameter passed in line `208` from `msg.ShortName` to `msg.Name`.

Status: Resolved

18. `MsgAddWhiteListedAsset` allow any caller to add an asset id to a locker's whitelist

Severity: Major

The `MsgAddWhiteListedAsset` function in `x/locker/keeper/msg_server.go:327` does not include any permission checks to ensure only authorized addresses are able to add asset Ids to the lockers whitelist. Currently any user may send this message to add asset Id's to the lockers whitelist.

Recommendation

We recommend performing an authorization check to ensure that only authorized addresses/governance/modules can call this function to add to the locker's whitelist.

Status: Resolved

19. `AddAppRecords` may accept invalid `GenesisToken` array

Severity: Major

The `AddAppRecords` function in `x/asset/keeper/app.go:203` does not perform proper validation on the `GenesisToken` array before setting it in the store.

At a minimum, the function should confirm that the slice of `MintGenesisToken` does not contain duplicates, and that the `Recipient` is a valid address.

An invalid stored `GenesisToken` array could break the handling of `AddAssetInAppRecords` messages during the execution of the `CheckIfAssetIsAddedToAppMapping` function.

Recommendation

We recommend validating the `GenesisToken` array in `AddAppRecords` or directly initialize it during `AddAssetInAppRecords`.

Status: Resolved

20. Inverse pairs in `AddPairsRecords` can create duplicate issues for `StableMint` vaults

Severity: Major

In `x/asset/keeper/asset.go:229`, in the `AddPairsRecords` function, there is a check that prevents the addition of duplicate pairs. However, it is possible to add the inverse pair (e.g. `assetIn1==assetOut2`, and `assetIn2==assetOut1`). Hence, it may be possible to create two `ExtendedPairVaults`, which are `StableMintVault` and point to inverse `PairId`. This would potentially allow `StableMintVault` to contain duplicate assets with different parameters and/or state.

Recommendation

We recommend either adjusting the documentation to state the possibility of having duplicate `StableMintVaults` for inverse `PairIds`, or preventing the addition of inverse pair records in `x/asset/keeper/asset.go:229`.

Status: Resolved

21. Whitelisted assets for rewards get overwritten when adding new assets, which may cause unexpected behavior

Severity: Major

In `x/rewards/keeper/keeper.go:98`, in the `WhitelistAsset` function, new assets are whitelisted for rewards. However, in line `93`, instead of appending the new `assetIDs` to the already whitelisted assets, the latter are overwritten. This may cause unexpected behavior, as assets with ongoing reward gauges may lose their whitelisted status.

Recommendation

We recommend appending the new `assetIDs` to the already existing assets when constructing the `InternalRewards` struct in line `93`.

Status: Resolved

22. `HandleUpdateAdminProposal` does not properly handle admin update and admin is unchangeable as a result

Severity: Major

The `HandleUpdateAdminProposal` function in `x/market/keeper/gov.go:9` does not properly perform an admin update. Instead, it simply gets the parameters from the store and directly performs `SetParams` with the same params. This means that it does not handle

the `UpdateAdminProposal` at all, and will not allow for the market module's admin to be updated.

Recommendation

We recommend handling the `UpdateAdminProposal` in `HandleUpdateAdminProposal` and updating the admin address with the one supplied in the message.

Status: Resolved

23. Hard-coded admins increase the potential of unauthorized privileged activity

Severity: Major

The `Admin` function `x/esm/keeper/klsw.go:42` is used to determine if the caller is an admin who is authorized to perform `SetKillSwitchData`. Currently this function uses hard coded addresses in `from_address`.

In addition, `from_address` contains an empty address entry. The `ValidateBasic` function for `MsgKillRequest` will error if `From` is empty, but it is still recommended to remove the empty entry in the slice.

Recommendation

If admins must be used over governance functionality, it is best practice to implement functionality for the removal/update of `ADMINS` if one were to get compromised.

Status: Resolved

24. Base gas fee is insufficient for functions with iteration

Severity: Major

The `liquidity` module implements a base gas rate for each operation. While this does provide a basic way to price a constant computational complexity, it does not accurately determine the gas prices when there is complex computation, for example in iterations. For instance, in `Unfarm` and `CancelAllOrders`, the number of iterations can increase well beyond the flat gas fee that is not able to scale with the operations that are being performed.

The liquidity module does define a `gasCostPerIteration` in `x/liquidity/types/params.go`, but this value is unused. `GasKv` is a `KVStore` wrapper that enables automatic gas consumption each time a read or write to the store is made. It is the solution of choice to track storage usage in Cosmos SDK applications.

The following provides an example of `GasKV GasConfig` operations:

```
HasCost:          1000,
DeleteCost:       1000,
ReadCostFlat:     1000,
ReadCostPerByte:  3,
WriteCostFlat:    2000,
WriteCostPerByte: 30,
IterNextCostFlat: 30,
```

Recommendation

We recommend using the already defined `gasCostPerIteration` to consume gas in functions where iterations occur beyond the current gas base price. Alternatively, a similar effect can be accomplished by implementing `GasKV` to appropriately charge gas for the above operations.

Status: Acknowledged

The client states that the parameter is experimental and the functionality is intended.

25. Orders execution batching design could expose a surface for frontrunning and price manipulation

Severity: Minor

The `Liquidity` module's `EndBlocker`, in line `x/liquidity/abci.go:40`, is designed to batch orders to ensure sequential execution.

As orders are not executed within a transaction and are publicly available and queryable, it opens a surface attack for frontrunning and price manipulations.

A possible scenario could be:

1. The attacker queries the orders at the end of the `batchSize` blocks window for a complete block's orderbook, or they could perform an ad-hoc query to detect specific orders they wish to arbitrage.
2. Using this knowledge of the orders they may be able to make counter trades or perform arbitrage on other Cosmos ecosystem AMMs.

Recommendation

We have two possible recommendations with various levels of effort to implement.

We recommend enforcing `params.BatchSize` to be equal to `1`. As orders are executed in the `EndBlocker`, an attacker cannot query the complete order list but only a partial one in the window of one block limiting the attack surface.

We recommend rethinking how orders are matched in order to defend against frontrunning without sacrificing performance.

Status: Acknowledged

26. Messages missing `ValidateBasic` will allow spam transactions to enter the mempool

Severity: Minor

There are multiple instances of messages that do not implement `ValidateBasic`. The `ValidateBasic` method of the `sdk.Msg` interface implemented by the module developer is run for each transaction. To discard obviously invalid messages, `ValidateBasic` is called early in the processing of the message in the `CheckTx` and `DeliverTx` functions. Due to the fact that `ValidateBasic` is not implemented, invalid transactions will be able to enter the mempool.

The following are instances of messages where `ValidateBasic` has not been implemented:

- `WhitelistAsset - x/rewards/types/tx.go:105`
- `RemoveWhitelistAsset - x/rewards/types/tx.go:143`
- `WhitelistAppIdVault - x/rewards/types/tx.go:179`
- `RemoveWhitelistAppIdVault - x/rewards/types/tx.go:215`
- `ActivateExternalRewardsLockers - x/rewards/types/tx.go:258`
- `ActivateExternalRewardsVault - x/rewards/types/tx.go:301`

Recommendation

We recommend performing checks in `ValidateBasic` for the messages mentioned above.

Status: Resolved

27. Unbounded iteration in `ValidateBasic` may cause node timeout

Severity: Minor

The `ValidateBasic` functions for `AddAssetsProposal` and `AddPairsProposal` in `x/asset/types/gov.go:72-76` and `143-147` both include unbounded iterations that may be exploited to cause a node timeout. Both of the functions loop over a caller-supplied slice that is not checked for duplicate entries and does not have a defined maximum size.

It is best practice to keep `ValidateBasic` checks simple as gas is not charged when `ValidateBasic` is executed. We recommend only performing all necessary stateless checks to enable middleware operations (for example, parsing the required signer accounts

to validate a signature) and stateless sanity checks that can be performed in constant time in the `CheckTx` phase. Other validation operations must be performed when handling a message in a module's `MsgServer`.

While this finding has a major impact we classify it as minor since these messages can only be passed by the governance router, and a large number of assets/pairs are required for this issue to have an impact.

Recommendation

We recommend simplifying the checks that are performed during the `ValidateBasic` step for these messages.

Status: Resolved

28. Missing validation in `AddAppRecords` may cause unexpected behavior

Severity: Minor

The `AddAppRecords` message in `x/asset/keeper/app.go:214` has minimal validation on the `AppData` provided through governance proposals, which may cause human errors and/or unexpected behavior. Similar examples of a lack of extensive input validation can also be found in `UpdateGovTimeInApp` in `x/asset/keeper/app.go:233` and `AddAssetInAppRecord` in `x/asset/keeper/app.go:244`.

Recommendation

We recommend adding more extensive input validation on `AppData`. Examples of these can be:

- Validate that `MinGovDeposit` is not negative or `0`.
- Validate that `GovTimeInSeconds` is within a valid range (e.g. `0 < x < maxTime`).

Even if these messages are executed through governance, validation can minimize human errors and unexpected behavior.

Status: Resolved

29. `Bank` keeper wrapper is not returning errors for zero values

Severity: Minor

The following locations:

- `x/vault/keeper/alias.go:10`
- `x/vault/keeper/alias.go:18`

- `x/vault/keeper/alias.go:26`
- `x/vault/keeper/alias.go:34`
- `x/vault/keeper/alias.go:41`

define the logic of a wrapper around several `Bank` keeper functions. Those wrappers implement a custom logic that returns always `nil` if the `Coin` parameter is a zero value.

As sending zero coins, minting zero coins or burning zero coin doesn't have effects and may imply errors from the caller, it would be better to return errors.

Recommendation

We recommend returning a custom error when `coin.IsZero() == true` to allow the caller to handle zero amounts.

Status: Resolved

30. Hard-coded addresses inside functions are prone to human errors during upgrades and refactors

Severity: Minor

In `x/esm/klsw.go:42`, in the `Admin` function, two hard-coded addresses are set to the `from_address` variable. Hard-coding addresses, especially for security sensitive logic such as an authorization, should be avoided as they are prone to human errors during upgrades or code refactoring.

Recommendation

We recommend adding the admin addresses as constants, to help increase the developer experience in future code changes and refactors.

Status: Resolved

31. CosmWasm - `beta` version dependencies shouldn't be used in production

Severity: Minor

In

- `contracts/governance/Cargo.toml:21`,
- `contracts/governance/Cargo.toml:28`,
- `packages/bindings/Cargo.toml:10`, and
- `packages/bindings/Cargo.toml:15`,

`beta` dependencies are specified.

Production ready code should not have `beta` dependencies as they could be not fully developed and audited and may contain (known) bugs or vulnerabilities.

Recommendation

We recommend updating the mentioned dependencies to the latest stable versions.

Status: Resolved

32. `CreateGauge` transactions are not checking `ESM status` or `killSwitch`

Severity: Informational

In `x/rewards/keeper/msg_server.go:23` during the execution of the `CreateGauge` transaction, there are no checks regarding `ESM status` or `killSwitch`.

As an `ESM status` disables all the withdrawal functionalities, this issue can lead to the loss of the funds deposited with the transaction.

Recommendation

We recommend implementing a check on the current `ESM status` and `killSwitch` and revert the transaction if one of them is triggered.

Status: Acknowledged

33. Iterative calculations should be avoided in `BeginBlocker` when an analytic solution is available in order to reduce the function execution time

Severity: Informational

In `x/rewards/keeper/keeper.go:39`, when calculating the number of `missedEpochs`, an iterative calculation approach is used.

Since this function is executed in a `BeginBlocker`, which is a time critical operation, iterative calculations should be replaced, if possible, with an analytic solution.

It is important to note also that indeterminate `for` loops with a conditional `break` guardian in line `41` should be avoided because they are prone to errors and difficult to debug.

In this specific case, `missedEpochs` can be calculated as follows:

$$missedEpochs = \frac{ctx.BlockTime - epoch.CurrentEpochStartTime}{epoch.Duration}$$

Recommendation

We recommend replacing the iterative calculation of `missedEpochs` with an analytic solution.

Status: Resolved

34. Duplicated denom validation is inefficient

Severity: Informational

The validation in `x/asset/types/asset.go:24-26` is unnecessary because the regex pattern in `ValidateDenom` will ensure the denom is at least 3 characters. While this is a small inefficiency, it is best practice to keep `ValidateBasic` checks simple and efficient.

Recommendation

We recommend removing the validation step on `x/asset/types/asset.go:24-26`.

Status: Resolved

35. Remove unused commented code blocks

Severity: Informational

The codebase contains several blocks of commented code. It is best practice to remove unused code comments to improve the readability and maintainability of the codebase.

The following are instances of unused code comments:

- `x/asset/types/asset.go:15-17`
- `x/asset/types/pair.go:8-10`
- `x/vault/keeper/msg_server.go:151`
- `x/vault/keeper/msg_server.go:339-342`
- `x/vault/keeper/msg_server.go:428-431`
- `x/vault/keeper/msg_server.go:557-560`
- `x/vault/keeper/msg_server.go:629-631`
- `x/locker/keeper/msg_server.go:221-223`
- `x/locker/keeper/msg_server.go:293-295`
- `x/vault/keeper/msg_server.go:515-517`

Recommendation

We recommend removing the lines mentioned above.

Status: Resolved

36. Spelling errors found in codebase

Severity: Informational

During the audit process, several spelling errors were found that impact readability:

- `x/vault/keeper/vault.go:303`, `307`, `318`, `325`, `329`, and `340`: "valut" instead of "vault"
- `x/vault/types/errors.go:33`: "ErrVaultAccessUnauthorised"
- `x/vault/keeper/vault.go:178`: "CalculateCollaterlizationRatio" instead of "CalculateCollateralizationRatio"
- `contracts/governance/src/contract.rs:328`: "propsal"

Recommendation

We recommend correcting typographical errors.

Status: Resolved

37. Date ranges util function can return false negatives

Severity: Informational

In `types/utils.go:33`, the function `DateRangesOverlap` is meant to return `true` when two date ranges overlap with each other. However, the case when `startTimeB.Before(endTimeA) && endTimeB.After(startTimeA)` is not covered, and this function will return erroneously `false` when the two dates ranges are actually overlapping.

Since this function is not being actively used, we have flagged this issue as informational.

Recommendation

We recommend covering the missing case, to ensure the function does not return false negatives and assure forward compatibility of this function.

Status: Resolved

38. Use of magic numbers is prone to human errors

Severity: Informational

There are several instances of numeric values without a label (so called “magic numbers”) across the codebase. The use of magic numbers is prone to human errors and can cause problems in future upgrades.

The following are examples of magic numbers:

- `x/rewards/keeper/keeper.go:187, 193`
- `x/market/abci.go:13`
- `x/bandoracle/abci.go:17`

Also, these are instances of magic numbers in the CosmWasm Gov Contract:

- `contracts/governance/src/state.rs:106, 141`
- `contracts/governance/src/validation.rs:13`

Recommendation

We recommend storing these numbers as constants for each given module, to increase developer experience and minimize the potential of human errors when upgrading the codebase.

Status: Resolved

39. `ValidateMsgCreateCreateGauge` performs redundant gauge validations

Severity: Informational

The `ValidateMsgCreateCreateGauge` function in `x/rewards/keeper/gauge.go:13` performs multiple checks that are stateless and have already be checked in the `MsgCreateGauge`’s `ValidateBasic` function.

The following fields have already been validated during `ValidateBasic` and can be removed from the `ValidateMsgCreateCreateGauge` function:

- `TriggerDuration`
- `DepositAmount`
- `StartTime`

Recommendation

We recommend removing the validations mentioned above from the `ValidateMsgCreateCreateGauge` function.

Status: Resolved

40. `MsgCreatePair` message execution always returns an error

Severity: Informational

In `x/liquidity/keeper/msg_server.go:26`, the function `CreatePair` that is responsible for handling `MsgCreatePair` messages, always returns an error.

Recommendation

We recommend removing `MsgCreatePair` as it is unusable.

Status: Resolved

41. Outstanding TODOs are present in the codebase

Severity: Informational

During the audit, `TODO` comments were found in the following lines:

- `x/liquidity/keeper/swap.go:563`
- `x/liquidity/keeper/batch.go:27`
- `x/liquidity/keeper/swap.go:563-567`

Recommendation

We recommend resolving `TODO` comments and/or removing them from the codebase.

Status: Resolved

42. Remove redundant validation logic

Severity: Informational

Both the `ValidateMsgFarm` and the `ValidateMsgUnfarm` functions in `x/liquidity/keeper/rewards.go:280-282` and `347-349` perform redundant validation that has already been performed in the `ValidateBasic` methods of these messages.

Recommendation

We recommend removing the validations mentioned above from both the `ValidateMsgFarm` and `ValidateMsgUnfarm` functions.

Status: Resolved

43. CosmWasm - Overflow checks not enabled for release profile

Severity: Informational

`contracts/governance/Cargo.toml` and `packages/bindings/Cargo.toml` do not explicitly enable `overflow-checks` for the release profile.

While enabled implicitly through the workspace manifest, a future refactoring might break this assumption.

Recommendation

We recommend enabling overflow checks in all packages, including those that do not currently perform calculations, to prevent unintended consequences if changes are added in future releases or during refactoring. Note that enabling overflow checks in packages other than the workspace manifest will lead to compiler warnings.

Status: Resolved