Audit Report

Comdex Lend and Liquidation Modules

v1.0

November 23, 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 Lend and Liquidation 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 on the following GitHub repository:

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

Commit hash: `73614898c261dd769f6bbf7833927fbf467a9853`

This audit was performed only on the `x/lend` and `x/liquidation` directories.

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 investors' access to a broad range of assets helping them in diversify and generate yield on their investments.

The audit scope comprehends Comdex's Lend and Liquidity Cosmos SDK modules.

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	Gas is not consumed if the transaction returns an error	Critical	Resolved
2	Input-dependent iteration in `Lend` module's `BeginBlocker` may slow down or stop block production	Critical	Resolved
3	Batch mechanism in `Liquidation` module's `BeginBlocker` may allow malicious manipulations	Critical	Resolved
4	Error raised in `BeginBlocker` could lead to state corruption	Critical	Resolved
5	Incorrect conditions when rebalancing stable rates	Major	Resolved
6	`FundModAcc` allows funds to be sent to any module which may result in permanently lost funds	Major	Resolved
7	Borrow position's `InterestAccumulated` is not updated before liquidation, leading to an incorrect interest calculation	Major	Resolved
8	`BeginBlocker` allows for errors to silently pass without being logged	Minor	Resolved
9	Missing validation checks in the codebase	Minor	Resolved
10	Vault `collateralitzationRatio` does not account for newly accumulated interest	Minor	Acknowledged
11	Incorrect value provided to `VerifyCollateralizationRatio` function	Minor	Resolved
12	`UpdateLendPairsRecords` allows governance to update lend pairs even if they have liquidity	Minor	Resolved
13	`AddAuctionParamsData` does not prevent existing data from being overwritten	Minor	Acknowledged
14	`AddAssetRatesParams` returns incorrect error message	Informational	Resolved
15	`CLI` uses flags instead of arguments to parse proposals	Informational	Acknowledged

16	Unneeded positivity validation for unsigned integers	Informational	Resolved
17	Unnecessary aliases pattern increase technical debt	Informational	Resolved
18	Duplicated `ValidateBasic` invocation in `CLI`	Informational	Resolved
19	Missing usage description for all transaction and query commands `CLI`	Informational	Resolved
20	Empty `Lend` module genesis state	Informational	Resolved
21	`DepositAsset` allows callers to deposit into positions they do not own	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Low-Medium	Inconsistent and incorrect naming conventions impair the code readability and clarity. The code also lacks consistent commenting.
Level of documentation	Medium	-
Test coverage	Low	20.5% test coverage for the `Lend` module. The `Liquidation` module contains failing tests.

Detailed Findings

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

Severity: Critical

The `ConsumeGas` function is used in the `lend` module to increase the `GasMeter`'s consumed gas by a set amount. Within the Comdex appchain, these amounts are pre-defined based on anticipated values. This method is problematic because gas is only consumed after successful execution and not in the event of an error. Even if the function returns an error, computation still occurs and gas should be consumed.

The `ConsumeGas` function is called in lines:

- `x/lend/keeper/msg_server.go:30`
- `x/lend/keeper/msg_server.go:44`
- `x/lend/keeper/msg_server.go:58`
- `x/lend/keeper/msg_server.go:72`
- `x/lend/keeper/msg_server.go:84`
- `x/lend/keeper/msg_server.go:96`
- `x/lend/keeper/msg_server.go:108`
- `x/lend/keeper/msg_server.go:121`
- `x/lend/keeper/msg_server.go:135`
- `x/lend/keeper/msg_server.go:147`
- `x/lend/keeper/msg_server.go:176`
- `x/lend/keeper/msg_server.go:190`

In these occurrences, the predefined gas amount is not consumed in the event of an error because the `ConsumeGas` invocations are located at the end of the transaction execution.

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

Recommendation

We recommend moving the mentioned `ConsumeGas` function calls to the beginning of the function bodies in order to ensure that sufficient gas is consumed.

Status: Resolved

2. Input-dependent iteration in `Lend` module's `BeginBlocker` may slow down or stop block production

Severity: Critical

`BeginBlocker` is a function that is executed at the beginning of every block, even if there are no transactions. It should have a light and constant computational footprint to avoid any negative impacts on block production. Too much computational weight at the beginning of each block may cause block production to slow down, or even surpass Tendermint's propose timeout, which results in a halt of the blockchain.

Consequently, it is best practice to make the `BeginBlocker` execution independent, or at least sub-linearly dependent, from the amount of data stored on-chain.

In the `Lend` module in `x/lend/abci.go:13-23`, the execution loops through all stored `borrows` in order to balance stable rates. Since `borrows` cardinality will grow with chain adoption or due to a spam attack, the issue described above will occur.

Recommendation

We recommend removing the iteration over `borrows`. A pull-over-push approach can be used instead.

Status: Resolved

3. Batch mechanism in `Liquidation` module's `BeginBlocker` may allow malicious manipulations

Severity: Critical

In the `Liquidation` module's `BeginBlocker`, specifically in `x/liquidation/keeper/liquidate_borrow.go:11-113` and `x/liquidation/keeper/liquidate_vaults.go:12-93`, the execution loops through all the borrow positions in order to check the collateral ratio health.

Since positions cardinality will grow with chain adoption or due to a spam attack, in order to defend against issues like "Input dependent iteration in Lend module's BeginBlocker may slow down or stop block production", functions implement a batching mechanism. It splits borrowing positions slice into different partitions of `params.LiquidationBatchSize` size.

This implies that it could happen that positions that are not healthy in a block interval but are not included in a batch of that interval are not liquidated.

This implementation also exposes a vulnerability where a malicious actor could manipulate the mentioned slice in order to create positions that will not be liquidated in a defined interval.

Recommendation

We recommend removing the batching mechanism and reworking the existing borrow data structure with a more efficient one in order to avoid large computations.

For example, a `min-heap` data structure may be used, with positions as nodes and the liquidation price as keys. Using this approach, the `BeginBlocker` execution could traverse the `min-heap` until it found a liquidation price greater than the current one instead of iterating through all positions. The `min-heap` should be maintained valid during transactions when adding or updating a position using a bubble-up algorithm and the `BeginBlocker` should take responsibility for the deletion of liquidated position as well as the `sub min-heaps` merge.

Alternatively, the liquidation logic could be moved to off-chain monitoring with an exposed transaction message for external parties to liquidate unhealthy positions.

Status: Resolved

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

Severity: Critical

In `x/liquidation/abci.go:13-27` and `x/lend/abci.go:13-23` both `Lend` and `Liquidation` modules wrap their `BeginBlocker`'s execution logic in an `ApplyFuncIfNoError` function in order to gracefully handle errors.

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: Resolved

5. Incorrect conditions when rebalancing stable rates

Severity: Major

In `x/lend/keeper/iter.go:203-209`, the `RebalanceStableRates` function checks conditions to see if rebalancing must be triggered.

The `Commodo` documentation states that stable rates should be

decreased if

$$S \geq St + 20\%$$

and increased if

$$S + 20\% \leq St \ \vee \ utilization \geq \ 90\%.$$

However, the actual implementation is using different conditions. Stable rates are decreased if

$$S \leq St + 20\%$$

and increased if,

$$S + 20\% \leq St \ \wedge utilization > \ 90\%.$$

This leads to a malfunctioning stable interest rate model.

An attacker may create multiple accounts with borrowing positions with a stable interest rate for an asset and ensure that the borrowing rate is slightly less than 20% so that the increase condition will never be triggered to rebalance the stable rate even though the utilization ratio hits more than 90%. When the utilization ratio hits 100%, it impacts lenders unable to withdraw their stable assets.

Recommendation

We recommend reworking conditions in order to be consistent with the documentation.

Status: Resolved

6. `FundModAcc` allows funds to be sent to any module which may result in permanently lost funds

Severity: Major

The `FundModAcc` function in `x/lend/keeper/keeper.go:1388` allows for the caller to specify any module name for funding.

This is problematic because funds could be sent to a module that is not designed to handle them making them stuck in the module. The `BankKeeper` will return an error if the module name is invalid, but as long as the module name specified is valid, this function will allow for funds to be sent.

While any address may call this function, we do not classify it as critical since users would not normally be sending funds directly to a module.

Recommendation

We recommend creating a list of modules that can be funded, and making the list be controlled by governance.

In addition, we recommend specifying a list of authorized callers to this function to ensure normal users will not unexpectedly lose their funds by sending this message. The list could be limited to governance or specified by governance.

Status: Resolved

7. Borrow position's `InterestAccumulated` is not updated before liquidation, leading to an incorrect interest calculation

Severity: Major

The `LiquidateBorrows` function in `x/liquidation/keeper/liquidate_borrow.go:11` does not update the borrow position's `InterestAccumulated` before performing the liquidation operations.

As a result, a stale value of `borrowPos.InterestAccumulated` is being used in `x/liquidation/keeper/liquidate_borrow.go:60`, `76`, and `94` to calculate the `currentCollateralizationRatio`, which leads to an incorrect value.

In addition, when a locked borrow is being created in `x/liquidation/keeper/liquidate_borrow.go:115`, the stale value is used to set `LockedVault.UpdatedAmountOut`. This can become especially problematic when the borrow position is large and the interest has not been updated for a long time period, in which case the `LockedVault.UpdatedAmountOut` will be incorrect by a considerable amount.

Recommendation

We recommend ensuring that the borrow position is updated before performing the liquidation operations. For example, in a vault liquidation, the interest is updated in `x/liquidation/keeper/liquidate_vaults.go:71` by calling the `CalculateVaultInterest` function.

Status: Resolved

8. `BeginBlocker` allows for errors to silently pass without being logged

Severity: Minor

The functions called during `BeginBlocker` do not properly propagate error information. Instead, they continue execution and silently pass on the condition that caused the error. This results in suboptimal visibility of the conditions that are causing errors.

For example, the `LiquidateVaults` function in `x/liquidation/keeper/liquidate_vaults.go:12` simply executes `continue` when errors are encountered.

While we acknowledge that an error or panic should not necessarily halt the `BeginBlocker`, it is best practice to log the conditions that are causing errors. In this example, a liquidatable vault could be silently passed because there is no visibility into which errors are being encountered or even which vaults caused the errors.

Recommendation

We recommend emitting events with information about specific vaults that throw errors during the `BeginBlocker` operations.

Status: Resolved

9. Missing validation checks in the codebase

Severity: Minor

There are some missing validation checks in the following transaction message types as well as the `Lend` module's governance handler messages:

- In `x/lend/types/pair.go:18`, validation is done only on `CPoolName` field.
- In `x/lend/types/pair.go:25`, validation on `AssetToPairMapping` is not implemented.
- In `x/lend/types/pair.go:29`, validation on `AssetRatesParams` is not implemented.
- In `x/lend/types/gov.go:195`, validation on `AuctionParams` is not implemented.

Recommendation

We recommend implementing these missing validations.

Status: Resolved

10. Vault `collateralitzationRatio` does not account for newly accumulated interest

Severity: Minor

The `LiquidateVaults` function in `x/liquidation/keeper/liquidate_vaults.go:12` uses an outdated value of `totalOut` to calculate `collateralitzationRatio` because `vault.InterestAccumulated` is not updated.

In fact, the vault interest is updated in line `71` when the `CalculateVaultInterest` function is called. This results in `collateralitzationRatio` not reflecting the current accumulated interest.

We report this as minor because unlike the issue "Borrow position's InterestAccumulated is not updated before liquidation", in this case, `LiquidateVaults` updates the position so the `LockedVault.UpdatedAmountOut` is correctly evaluated.

Recommendation

We recommend executing `CalculateVaultInterest` before the calculation of `collateralitzationRatio` to ensure that the ratio reflects the current accrued interest.

Status: Acknowledged

11. Incorrect value provided to `VerifyCollateralizationRatio` function

Severity: Minor

The `VerifyCollateralizationRatio` function takes `liquidationThreshold` as the last parameter and uses it in order to perform validation for whether or not the collateralization ratio is greater than the liquidation threshold.

In four different instances, the function is incorrectly called with the `LTV` instead of the `liquidationThreshold`:

- `x/lend/keeper/keeper.go:541`
- `x/lend/keeper/keeper.go:662`
- `x/lend/keeper/keeper.go:720`
- `x/lend/keeper/keeper.go:1159`

This implies that depending on the value of `LTV` set by governance, the function either always returns `ErrorCollateralizationRatio` or bypasses the validation which could lead to unexpected results.

Recommendation

We recommend calling `VerifyCollateralizationRatio` with the correct parameter value.

Status: Resolved

12. `UpdateLendPairsRecords` allows governance to update lend pairs even if they have liquidity

Severity: Minor

The `UpdateLendPairsRecords` function in `x/lend/keeper/pair.go:39` allows governance to update either `AssetIn` or `AssetOut` for an existing lend pair.

If that existing lend pair is currently being utilized, this could have unintended consequences.

Based on the documentation it does not seem that there is any specific reason for allowing updates of lend pairs.

Recommendation

We recommend removing the ability to update existing lend pairs.

Status: Resolved

13. `AddAuctionParamsData` does not prevent existing data from being overwritten

Severity: Minor

The `AddAuctionParamsData` function in `x/lend/keeper/pair.go:227` does not ensure that existing data with the same key does not exist before saving the provided `AuctionParams`. This allows governance to overwrite existing data.

We classify this as a minor issue because only governance can call this function.

Recommendation

We recommend validating that data with the key supplied does not already exist.

Status: Acknowledged

14. `AddAssetRatesParams` returns incorrect error message

Severity: Informational

The `AddAssetRatesParams` function in `x/lend/keeper/pair.go:200` returns the incorrect error "`Asset Rates Params not found`" when the record already exists.

Recommendation

We recommend returning an accurate error message.

Status: Resolved

15. `CLI` uses flags instead of arguments to parse proposals

Severity: Informational

In

- `x/lend/client/cli/tx.go:437`,
- `x/lend/client/cli/tx.go:523`,
- `x/lend/client/cli/tx.go:778`, and
- `x/lend/client/cli/tx.go:901`,

the commands for the `CmdAddNewLendPairsProposal`, `CmdAddPoolProposal`, `CmdAddNewAssetRatesParamsProposal`, and `CmdAddNewAuctionParamsProposal` governance proposals currently receive a proposal file as a flag.

Recommendation

We recommend passing a proposal file as an argument rather than a flag since flags imply optional values. Additionally, we recommend using a file of an appropriate typeto prevent type conversions.

Status: Acknowledged

16. Unneeded positivity validation for unsigned integers

Severity: Informational

All the `ValidateBasic` functions in `x/lend/types/tx.go` validate that the `ID` parameter is a positive number. As unsigned integer type can never be negative, those checks are redundant. Therefore, they can be removed.

Recommendation

We recommend removing positivity checks of unsigned integers in all `ValidateBasic` functions in `x/lend/types/tx.go`.

Status: Resolved

17. Unnecessary aliases pattern increase technical debt

Severity: Informational

Currently, there are some functions for external modules defined in `x/lend/types/expected_keepers.go`, while the keeper uses the functions from `alias.go` that reference `expected/keeper.go`. Having an `alias.go` file defining all functions of external modules is not necessary and decreases maintainability.

Recommendation

We recommend removing the `alias.go` file and specifying functions of external modules in `types/expected_keeper.go` or `expected/keeper.go`.

Status: Resolved

18. Duplicated `ValidateBasic` invocation in `CLI`

Severity: Informational

Some CLI commands defined in

- `x/lend/client/cli/tx.go:426`,
- `x/lend/client/cli/tx.go:517`,
- `x/lend/client/cli/tx.go:615`,
- `x/lend/client/cli/tx.go:693`,
- `x/lend/client/cli/tx.go:761`,
- `x/lend/client/cli/tx.go:894`, and
- `x/lend/client/cli/tx.go:986`,

call the `msg.ValidateBasic` function before calling `GenerateOrBroadcastTxCLI` or `GenerateOrBroadcastTxWithFactory`. As `msg.ValidateBasic` is already called inside `GenerateOrBroadcastTxWithFactory`, this is an unnecessary and duplicated invocation.

Recommendation

We recommend removing `msg.ValidateBasic` call as it is already called inside the `GenerateOrBroadcastTxWithFactory`.

Status: Resolved

19. Missing usage description for all transaction and query commands `CLI`

Severity: Informational

All the transaction and query commands for the `Lend` module in `x/lend/client/cli/tx.go:30-43` and `x/lend/client/cli/query.go:28-52` are missing a long message to provide description about their usage, which will be helpful for users and external developers.

Recommendation

We recommend specifying a long message for all transaction and query commands. Each command should provide a description of how to correctly use the command.

Status: Resolved

20. Empty `Lend` module genesis state

Severity: Informational

The `Lend` module currently has an empty genesis state. This appears to be temporarily commented out, but should be added before the code is released in production.

Recommendation

We recommend re-implementing the genesis state for the `Lend` module or removing the comments.

Status: Resolved

21. `DepositAsset` allows callers to deposit into positions they do not own

Severity: Informational

The `DepositAsset` function in `x/lend/keeper/keeper.go:319` does not verify that the caller is the owner of the `lendPos` being deposited into. This may allow users to

inadvertently deposit funds to a position that they do not own, resulting in an unexpected loss of funds.

Recommendation

We recommend verifying that the caller is the owner of the vault and return an error otherwise.

Status: Resolved