CERTIK

Final CertiK Audit Report for Kava

Contents

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Verification Services Agreement between CertiK and Kava (the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the "Agreement"). This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK's mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that the project is checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and verifications on the project, in turn creating a more secure and robust software system.

CertiK has served more than 100 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance's BGBP and Paxos Gold to decentralized oracles such as Band Protocol and Tellor. CertiK customizes its engineering tool kits, while applying

cutting-edge research on smart contracts, for each client on its project to offer a high quality delivery. As it utilizes technologies from blockchain and smart contracts, the CertiK team will continue to support the project as a service provider and collaborator.

Executive Summary

Kava is a cross-chain DeFi (Decentralized Finance) project built on top of the Cosmos SDK. In a day and age where waves of security incidents have happened on DeFi protocols and millions of dollars have been lost, it's mission critical to identify security vulnerabilities that carry both intrinsic and extrinsic risks.

To that end, the sole objective of the audit is to verify Kava Labs' implementation of the CDP and Auction modules against the provided specifications. A series of thorough security assessments have been carried out, the goal of which is to help the said project protect their users by finding and fixing known vulnerabilities that could cause unauthorized access, loss of funds, cascading failures, and/or other vulnerabilities. Alongside each security finding, recommendations on fixes and best practices have also been given.

Testing Summary

SECURITY LEVEL

Cross-Chain Defi Audit

This report has been prepared as a product of the DeFi Audit request by Kava Labs.

This audit was conducted to discover issues and vulnerabilities in the source code of Kava Lab's blockchain implementation.

TYPE	Blockchain Implementation
SOURCE CODE	https://github.com/Kava-Labs/kava/tree/8f3858509a0aff6ed26767d35c6ea5f64d808e03
PLATFORM	Cosmos SDK v0.38.4
LANGUAGE	Golang
REQUEST DATE	May 13, 2020
REVISION DATE	June 28, 2020
METHODS	A comprehensive examination has been performed using Whitebox Analysis. In detail, Dynamic Analysis, Static Analysis, and Manual Review were utilized.

Review Notes

Overview

A primary focus for the audit is to have a thorough look at two main parts of each application module, namely the **State** and the **Message** types. Specifically we analyze how the state machines are defined and how state transitions are triggered by messages, the goal of which is to check the implementation against the specs and hence minimize the possibilities of unintentional state behaviors taking place.

Following a modular design approach outlined in the SDK, we inspect each and every module within the scope to ensure that:

1. Modules have their own message (or transaction) processors in place
2. The SDK is utilized in a least-authority manner, primarily for routing messages to their intended modules
3. Modules are properly aggregated into one functional application

Scope of Work

- The audit work was strictly scoped to a specific commit of the source code per the agreement
- Modules within the scope include: CDP module; Auction module
- State transitions in each module were carefully verified against their specification
- Test code was analyzed and assumed to hold true for the purpose of auditing. Efforts on ensuring the correctness or effectiveness of the tests were beyond the scope of this audit

- Go programming best practices were enforced to improve general performance and minimize the chances of run-time panicking

Audit Approach

Our audit approach revolves around ensuring that the security model in Kava is done in a secure and functionally correct manner so that it aids the encapsulation of the modules of the blockchain and helps safeguard the application against unintentional state changes. Apart from assessing the security model, best practices in Go programming will also be applied. The practices include:

- Correct simulation implementation for fuzzy testing to avoid incorrect assumptions
- Secure module inter-dependency instantiation on a need-to-know basis
- Proper and meaningful definition of application invariants

Following the unique structural properties and security models of a Cosmos SDK application, our audit approach largely favors modularity and encapsulation in code design. At a high level we analyze each object by their interfaces and references to other objects. This ultimately ensures that the same security properties can be extended to new objects added to the system, which in return minimizes the attack surface of the application down to the implementation of specific objects. In the following sections, we give a detailed look on some of the key checks performed in our evaluation process.

Type Definition

There are four primitives in a custom application's type definitions that we look into:

- **baseapp**: The application needs to correctly implement the ABCI interface via the boilerplate implementation provided in the SDK, namely the baseapp. Two main things we look out for here: 1) routes are properly defined; 2) states are correctly initiated
- **Stores**: Each module's state is persisted and individually managed in distinct compartments via Multistore
- **Keepers**: As a key piece in a module's interaction with its stores, the Keepers need to be properly declared and exported as interfaces to other modules for inter-module interactions to work
- **codec**: The encoding format persists data stores in byte slices deterministically

Constructor Function

Following the type definitions, we check the constructor functions against the following criterias:

- A new app instance can be created
- Keepers are correctly ordered as they are declared in type definitions
- A Module Manager can be correctly instantiated
    - All modules are included
    - Order of execution between key functions of each module is specified
    - routes (to handler) and query routes (to a querier) are in place
    - Invariants of each module are registered
    - Stores are mounted

Gas, Fees & the AnteHandler

Analogous to the gas model in most modern blockchains, gas in the Cosmos SDK is a unit that tracks the consumption of computational resources. The SDK provides a block gas meter implementation that ensures blocks can be finalized without consuming an excessive amount of gas. However by default the SDK does not enforce gas pricing. It is the project's responsibility to prevent a gas fee mechanism to prevent spam and abuse from end users. A couple of key points we emphasize here:

- ctx.GasMeter() in the AnteHandler is set to zero at the beginning of each DeliverTx. Or infinite gas loop would be possible
- GasKv.Store is enabled for automatic resource consumption tracking in the application
- AnteHandler needs to meet the following requirements:
    - Correctly verifies the transaction types defined in the same module
    - Correctly verifies the signatures in each transaction
    - Correctly verifies the sender has enough funds to cover the fees, and that the gas price in each transaction is no less than a locally set min-gas-prices

Audit Revision

The Kava team took our Exhibits into account and decided to proceed optimizing their codebase according to our recommendations. After multiple commits, a second GitHub repository revision was assessed accessible at the following link:

- Kava-Labs/kava:master [e913dc2ff0ed5fe61dd14705f1a15615c939826b]

The changes to the codebase were evaluated and represented in the Exhibits that follow wherever applicable.

State Transition Evaluation

CDP

FUNCTIONALITY	TYPE	PASS
CreateCDP	• Whenever a new CDP is created, the sender becomes the CDP owner (🔗) • Collateral is taken from a Sender and sent to the CDP module account, creating a new Deposit (🔗) • Principal coins are minted and sent to the Sender (🔗) • Amount of internal debt coins created and stored in CDP module account are equal (🔗) • When CDPs are updated, the database index is as well (🔗)	✔
Deposit	• Collateral is taken from the Depositor and sent to the CDP module account (🔗) • The Depositor's internal Deposit struct is either updated or newly created (🔗) • CDP fees are updated (🔗) • When CDPs are updated, the database index is as well (🔗)	✔
Withdraw	• Collateral coins are sent from the CDP module account to the Depositor (🔗) • An amount equal to the withdrawal is subtracted from the Deposit struct. If the value is now zero, the struct is deleted (🔗) • When CDPs are updated, the database index is as well (🔗)	✔

DrawDebt	• Principal coins are minted and sent to the sender, updating the CDP's Principal field in the process (🔗) • An equal amount of coins are minted in debt and stored in the module account (🔗) • The total principal is increased for the principal denominator (🔗) • When CDPs are updated, the database index is as well (🔗)	✔
RepayDebt	• Burn Payment coins from the Sender, updating the CPD by reducing its Principal field by the amount (🔗) • Burn an equal amount of debt coins (🔗) • The total principal is decreased for the principal denominator (🔗) • If fees and remaining principal are equal to zero, the collateral is returned to its Depositors and the CDP struct is deleted (🔗) • When CDPs are updated, the database index is as well (🔗)	✔

Auction

FUNCTIONALITY	TYPE	PASS
Bidding	• The Bidder is updated if its different than the previous bidder (🔗) • The auction is extended by BidDuration, up to MaxEndTime (🔗 & 🔗)	✔
Surplus Auction	• Bid is updated to msg.Amount (🔗) • The previous Bidder is refunded (🔗) • The increment between bids is burned (CurrentBid - PreviousBid) (🔗)	✔
Debt Auction	• Lot amount is updated to msg.Amount (🔗) • The previous Bidder is refunded (🔗)	✔
Collateral Auction	• The previous Bidder is refunded (🔗 & 🔗) • If the auction is in forward phase, the bid amount is updated to msg.Amount (🔗) • If the auction is in reverse phase, the lot amount is updated to msg.Amount (🔗)	✔

Audit Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Inconsistent Object Instantiation	Coding Style	Informational	cdp/keeper/auctions.go L20, cdp/types/deposit.go L16 & more

[INFORMATIONAL] Description:

Objects in Go can be instantiated either by specifying key-value pairs or passing the variables to the declaration directly. Within the codebase of Kava, both patterns are observed.

Recommendations:

We advise that the codebase is updated to conform to one of the two patterns to ensure consistency within the codebase.

Alleviation:

The method via which objects are instantiated was partially streamlined across the codebase to the key-value declaration format, however certain positional declarations still exist in the codebase.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Inconsistent Calculation Paradigms	Coding Style	Informational	cdp/keeper/auctions.go L30, cdp/types/deposit.go L55

[INFORMATIONAL] Description:

The functions that exist within the lines specified above possess the same function name “SumCollateral” yet conduct different logical operations within. In detail, only one of the two checks whether the amount to be added to the sum is equal to zero and if so, skips it.

Recommendations:

We advise that these functions be updated to contain the same logical checks within their body to ensure that they are consistent.

Alleviation:

The “partialDeposits” paradigm was removed as a side-effect of another Exhibit and as such, only the “SumCollateral” function of “Deposits” exists which correctly checks whether the amount to be added to the sum is equal to zero.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Inconsistent Naming Conventions	Coding Style	Informational	General Comment

[INFORMATIONAL] Description:

The structs “Deposit” and “partialDeposit” do not follow the same naming convention whereby one conforms to the partial camel-case format and the other to the full camel-case format. Additionally, both structs possess a member called “Amount” which is of type “sdk.Coin”, itself having a member called “Amount”. This leads to ambiguous accessors such as “d.Amount.Amount”.

Recommendations:

We advise that the “Amount” member of the structs is renamed to something more legible and that the codebase is re-scanned to fix any inconsistent naming conventions that may reside within.

Alleviation:

The “partialDeposits” paradigm was removed as a side-effect of another Exhibit and as such, the camel-case inconsistency is rendered null. The double “Amount” accessor issue still exists, however, as the “Amount” attribute was not renamed.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Controversial Variable Naming	Coding Style	Informational	cdp/keeper/auctions.go L11

[INFORMATIONAL] Description:

The factor via which the “governance” tokens are sold at auctions is called “dump”.

Recommendations:

We believe this should be renamed to avoid the negativity around the word “dump” in the crypto space although it is valid as a term.

Alleviation:

This Exhibit has not been dealt with, however it has been taken into consideration by the Kava team and may be fixed in a future commit.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Inconsistent Logical Assumptions	Coding Style	Informational	cdp/keeper/auctions.go L48, L57

[INFORMATIONAL] Description:

The “AuctionCollateral” function of “auctions.go” retrieves the permitted “auctionSize” from the configuration based on the denomination of the first deposit. This assumes that all partial deposits are of the same denomination. However, L57 initializes a local variable with the denomination of each deposit processed by the loop the statement is contained within, implying that multiple denominations may be processed by the loop.

Recommendations:

Since it is assumed that all partial deposits are of the same denomination, the statement of L57 is unnecessary and as such we propose its removal and the storage of the denomination of all partial deposits in a variable outside the scope of the loop blocks.

Alleviation:

The denomination of a deposit is still retrieved from the “collateral” within “CreateAuctionsFromDeposit”, however this is an expected side-effect of splitting the logic of the auction creation loop into multiple functions and as such we consider this Exhibit dealt with.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Logical Operand	Ineffectual Code	Informational	cdp/keeper/auctions.go L51

[INFORMATIONAL] Description:

The “for” loop of the statement checks whether the “totalCollateral” variable is greater than “zero”, in essence checking whether it is positive.

Recommendations:

This line can be simplified to “totalCollateral.IsPositive()”, a native function of the Cosmos SDK which internally checks the low-level sign of the numerical representation of the number which is more efficient than comparing it with another number.

Alleviation:

This Exhibit has been dealt with as a side-effect of refactoring the auction creation process. Additionally, any new code in the refactored segment that needs to check whether a value is positive, such as L67, correctly invokes the “IsPositive” method rather than comparing to an instantiated SDK zero integer.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Potential Resource Exhaustion	Ineffectual Code	Major	cdp/keeper/auctions.go L51 - L103

[MAJOR] Description:

The code block contains nested loops that iterate recursively over an array, conducting expensive instructions within. As the code block at its current state is highly unoptimized, its space and time complexity are very high exposing a Denial-of-Service attack vector which can be exploited.

Recommendations:

The code block can be highly optimized.

At its current state, it loops through all deposits, creates as many auctions as possible for each deposit amount (this number is equal to the total times the variable “auctionSize” can “fit” in the deposit amount) and stores the remainder, if any, in a “partialDeposits” array.

Additionally, on each iteration it checks whether the current deposit’s remainder plus the total amount stored in “partialDeposits” will exceed the threshold (“auctionSize”) of creating an auction and, if so, creates an auction from the partial deposits and the new remainder replaces the deposit currently being iterated in the “deposits” array.

Overall, the procedure is highly inefficient because it essentially iterates through all deposits and replaces them with whatever remainder is left if “partialDeposits” exceed the auction threshold. Simplified, this means that for the following array:

[10, 89, 200]

And an auction threshold of 100, the whole array will be iterated twice (thrice if we include the “SumCollateral” call of the function), with the second iteration skipping the first two elements due to their values being zero.

The outer “for” loop can be avoided altogether by simply storing the new remainder after the consumption of “partialDeposits” in the L78-L98 “else” block in the overridden “partialDeposits” array on L96. The remainder is mathematically guaranteed to be less than “auctionSize”. Additionally, this makes the statements of L100 & L101 redundant.

As a supplementary suggestion, we advise that the “if-else” block of L73-L98 be replaced by a single “if” statement as they are quite similar and this would also reduce the complexity a bit more. The check of L73 is possible to optimize by storing the previous result of “SumCollateral()” and adding the new value each time instead of iterating through the whole “partialAuctionDeposits” array each time.

Alleviation:

This Exhibit was dealt with in full by applying the mathematical assumptions we laid out in the “Recommendations” section and keeping the bare minimum of code within the code loops. The code block was once again refactored to be optimized to the greatest extent possible conducting a single loop for the fulfillment of the function's purposes. Our notes in the current iteration would be the two superfluous local variable initializations (L60 & L64) and the

superfluous assignment to the unallocatedDebt variable on L95 which is not used past that line. We would also advise a sanity assertion to be introduced before the function ends whereby the variable unallocatedDebt is ensured to be zero once the function concludes.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Code Block	Ineffectual Code	Minor	cdp/keeper/auctions.go L51

[MINOR] Description:

The function of the statement contains a set of instructions that are executed in each loop iteration with different values based on sequential mathematical operations. Since the loops essentially change how the calculations are carried out, it is possible to simplify certain aspects of the function via optimized math.

Recommendations:

In its current implementation, the “for” loop of L121-L139 runs as long as the value of “depositAmount” is greater than or equal to “auctionSize”. Internally, it sends a transaction equal to “auctionSize” and removes that from the “depositAmount” value. This renders all statements inside the “for” loop (apart from the auction creation) redundant as they can be calculated using mathematical operations i.e. “div”, “modulo” etc.

Additionally, the function retains the “debtChange” and “collateralChange” variables that signify how much value was set for auction and subsequently reduced off the debt. In its current iteration, L132-L137 “Add” and “Sub” the exact same values on multiple variables. This is unnecessary as the “debt” and “totalCollateral” themselves could be returned by the function.

If the “Change” variables are desired instead, statements L134 & L134 are redundant since both of them are utilized in the calculation of a percentage, meaning the resulting division (“despoitDebtAmount”) will remain the same through all iterations.

To simplify the code block, simply calculate the result of the “div” operation between “depositAmount” and “auctionSize” and create as many auctions by running the statement located between L126-L128 that many times.

Alleviation:

The optimizations laid out in this Exhibit were fully assimilated in the refactored codebase, creating the auctions in a simple loop with no extraneous statements.

Exhibit 9

TITLE	TYPE	SEVERITY	LOCATION
Inconsistent Logical Assumption	Ineffectual Code	Informational	cdp/keeper/auctions.go L154

[INFORMATIONAL] Description:

Consult Exhibit 5.

Recommendations:

Consult Exhibit 5.

Alleviation:

The “CreateAuctionFromPartialDeposits” function was completely removed based on the refactoring of the auction creation process, rendering this Exhibit null.

Exhibit 10

TITLE	TYPE	SEVERITY	LOCATION
Ineffectual Sum Calculation	Ineffectual Code	Informational	cdp/keeper/auctions.go L159

[INFORMATIONAL] Description:

The sum of the all partial deposits is calculated to be returned to the caller of the function via the “SumCollateral” function of the “partialDeposits” type. However, the auction created therein uses the “auctionSize” variable rather than the calculated sum that is returned.

Recommendations:

The “collateralChange” will always mathematically be equal to “auctionSize” since that is also the amount used in the statement of L154. We advise that “auctionSize” either be returned or the auction that invokes this function is revised accordingly to not require an input variable to be returned as an output at all.

Alleviation:

The “CreateAuctionFromPartialDeposits” function was completely removed based on the refactoring of the auction creation process, rendering this Exhibit null.

Exhibit 11

TITLE	TYPE	SEVERITY	LOCATION
Illegible Nested Calls	Ineffectual Code	Informational	cdp/keeper/auctions.go L173, L181 & L187

[INFORMATIONAL] Description:

The specified lines contain statements in the form of “NewCoins(NewCoin(…))” nested calls. The “NewCoins” constructor internally conducts certain operations on its inputs that are better suited for actual arrays of coins rather than singleton arrays, thus leading to unnecessary statements being executed. Specifically, it internally sorts them, filters any ones that are zero and checks whether duplicate denominators occur inside all of which are null when the input is a single “Coin”.

Recommendations:

The internal invocations of “NewCoins” can be avoided altogether by constructing the “[]Coin” directly or perhaps assessing whether “BurnCoins” must be able to accept multiple coins or not, thus nullifying the need for “[]Coin” altogether.

Alleviation:

As this Exhibit would potentially require breaking changes to functions such as the “BurnCoins” of “supplyKeeper”, we consider this Exhibit as null.

Exhibit 12

TITLE	TYPE	SEVERITY	LOCATION
Redundant “if” chains	Ineffectual Code	Informational	cdp/keeper/auctions.go L179 - 191

[INFORMATIONAL] Description:

The specified lines contain multiple logical checks that can be skipped altogether if the “sdk.MinInt” functions are properly utilized.

Recommendations:

We advise that the statements be reduced in half by burning an amount equal to the result of “sdk.MinInt(balance, netAmount)” as utilized in L168.

Alleviation:

The “MinInt” function was properly utilized according to our recommendations and the code segment was optimized.

Exhibit 13

TITLE	TYPE	SEVERITY	LOCATION
Inconsistent Code Blocks	Ineffectual Code	Informational	cdp/keeper/auctions.go L199 - L202, L205 - L209

[INFORMATIONAL] Description:

These two code blocks are inconsistent. The former creates a variable that is initialized at zero and adds the result of “AmountOf” to it whereas the latter directly returns the value of “AmountOf”.

Recommendations:

Both blocks are identical and as such, “GetTotalSurplus” should be adapted to be effectively the same as “GetTotalDebt”.

Alleviation:

The code blocks of “GetTotalSurplus” and “GetTotalDebt” were updated to be consistent according to our recommendations.

Exhibit 14

TITLE	TYPE	SEVERITY	LOCATION
Misconfiguration via String Literal	Ineffectual Code	Minor	cdp/keeper/auctions.go L219

[MINOR] Description:

The specified line uses the hard-coded string literal “usdx” as the denominator of debt, whereas throughout the rest of the codebase this is retrieved via the configuration. This would lead to incorrect auctions being made should the configuration mismatch the “usdx” parameter.

Recommendations:

We advise that this statement be adapted to utilize the configuration of the chain rather than a string literal for setting up the debt auctions.

Alleviation:

The code segment was adapted to correctly retrieve the denominator of debt from the contextual parameters of the blockchain rather than a hard-coded literal.

Exhibit 15

TITLE	TYPE	SEVERITY	LOCATION
Inconsistent Description of Functionality	Misleading Comment	Informational	cdp/keeper/seize.go L91

[INFORMATIONAL] Description:

The specified line states that the function calculates the liquidation penalty, which is correct, and then mints the debt coins in the CDP module account, which is false.

Recommendations:

We advise that the comment be revised to reflect the actual functionality of the function being described.

Alleviation:

The comment was properly corrected to remove the latter statement which was false.

Exhibit 16

TITLE	TYPE	SEVERITY	LOCATION
Inconsistent Naming Convention	Coding Style	Informational	cdp/types/keys.go L27 & L30

[INFORMATIONAL] Description:

The Liquidator Module Account is abbreviated to “LiquidatorMacc” within variables, a name that does not conform to the camel-case specification.

Recommendations:

We advise that the name be adjusted to either “LiquidatorMAcc” or “LiquidatorAcc” to ensure it conforms to the camel-case convention.

Alleviation:

This Exhibit has not been dealt with, however it has been taken into consideration by the Kava team and may be fixed in a future commit.

Exhibit 17

TITLE	TYPE	SEVERITY	LOCATION
Magic Number	Coding Style	Informational	cdp/keeper/fees.go L17

[INFORMATIONAL] Description:

The line utilizes the Magic Number “10^18”.

Recommendations:

Although its meaning is widely known within the cryptocurrency community, it should be replaced by a variable that describes its significance to conform to the latest coding practices.

Alleviation:

This Exhibit has not been dealt with, however it has been taken into consideration by the Kava team and may be fixed in a future commit.

Exhibit 18

TITLE	TYPE	SEVERITY	LOCATION
Ambiguous Comment	Misleading Comment	Informational	cdp/keeper/fees.go L41

[INFORMATIONAL] Description:

The specified line contains a comment that does not relate to the current execution scope.

Recommendations:

The comment should be rephrased or relocated to better define what it is meant to describe.

Alleviation:

The misleading comment was completely removed from the specified line.

Exhibit 19

TITLE	TYPE	SEVERITY	LOCATION
Redundant “if” Clause	Ineffectual Code	Informational	cdp/keeper/cdp.go L118 - L121

[INFORMATIONAL] Description:

The “err” return variable is compared against “nil” and, if different from “nil”, it is returned.

Otherwise, the literal “nil” is returned.

Recommendations:

The “if” clause is redundant as the variable “err” can be returned directly.

Alleviation:

The “err” variable is correctly returned directly in this instance.

Exhibit 20

TITLE	TYPE	SEVERITY	LOCATION
Redundant “if” Clause	Ineffectual Code	Informational	cdp/keeper/cdp.go L128 - L131

[INFORMATIONAL] Description:

See Exhibit 19.

Recommendations:

See Exhibit 19.

Alleviation:

See Exhibit 19.

Exhibit 21

TITLE	TYPE	SEVERITY	LOCATION
Unoptimized “for” Loop	Ineffectual Code	Informational	cdp/keeper/cdp.go L288 - L292

[INFORMATIONAL] Description:

The aforementioned “for” loop iterates through all the CDP IDs of a member and appends them to a newly instantiated array as long as they are different from the ID of the CDP currently being removed.

Recommendations:

Since the IDs of each CDP are unique, it is possible to simply find the index at which the removed CDP’s ID is located within the array and perform a “splice” using the “copy” Golang operand instead of appending each element on a new array and comparing the IDs of all elements.

Alleviation:

This Exhibit has not been dealt with, however it has been taken into consideration by the Kava team and may be fixed in a future commit.

Exhibit 22

TITLE	TYPE	SEVERITY	LOCATION
Return Variable Ignored	Ineffectual Code	Minor	cdp/keeper/cdp.go L303 - L310

[MINOR] Description:

The lines above ignore the “found” return variable of the associated calls, meaning that the value actually being accessed may be equal to “0x00”.

Recommendations:

As this can lead to unintended consequences, we advise that the “found” variable is properly assigned and evaluated.

Alleviation:

The “GetDenomPrefix” calls were updated to “panic” if the prefix is not found where appropriate.

Exhibit 23

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Comment	Misleading Comment	Informational	cdp/keeper/cdp.go L322

[INFORMATIONAL] Description:

The comment possesses no relation to the code block that follows it.

Recommendations:

As the comment is a duplicate of L314, we advise its revision to properly reflect the code block below it.

Alleviation:

The misleading comment was corrected to properly reflect the code block that follows it.

Exhibit 24

TITLE	TYPE	SEVERITY	LOCATION
Misleading Function Name	Coding Style	Informational	cdp/keeper/cdp.go L349

[INFORMATIONAL] Description:

The function of L349 is named “ValidateCollateral”, however its function body simply checks whether the Collateral’s denominator exists in the system and nothing else.

Recommendations:

As the “Collateral” struct possesses a few more members that remain untouched, we advise this function be renamed so as to not mislead users to believe that a Collateral is valid if the function does not return an error.

Alleviation:

Additional statements were introduced to the “ValidateCollateral” code block to also check whether the asset’s price feed is operating, thus properly validating the collateral rather than checking whether its denominator simply exists.

Exhibit 25

TITLE	TYPE	SEVERITY	LOCATION
Empty Comment Line	Misleading Comment	Informational	cdp/keeper/cdp.go L401

[INFORMATIONAL] Description:

L401 possesses an empty comment.

Recommendations:

We advise that either the comment is fully fleshed out or removed from the codebase.

Alleviation:

The empty line comment was removed.

Exhibit 26

TITLE	TYPE	SEVERITY	LOCATION
Integer Overflow	Ineffectual Math	Major	cdp/keeper/cdp.go L434 - L436

[MAJOR] Description:

The lines specified convert the numerical representation of the Cosmos SDK to an int64 which possesses significantly less points of precision. Although the Cosmos SDK internally checks whether its integers when converted to int64 can be represented by it, the addition included in the code segments can lead to an overflow if cumulatively they sum to a number that exceeds the int64 limit, which is roughly equivalent to 8 ethereum represented in wei.

Recommendations:

We advise that the numerical representations provided by the Cosmos SDK are properly utilized in all operations to ensure that no overflow occurs and to ensure that the highest numerical precision is retained.

Alleviation:

The function was re-written to properly utilize the native “sdk.Int” rather than “int64” which are prone to the aforementioned overflow issue.

Exhibit 27

TITLE	TYPE	SEVERITY	LOCATION
Inconsistent Handling of Errors	Coding Style	Informational	cdp/keeper/draw.go L36 - L38, L40 - L42 & more

[INFORMATIONAL] Description:

In certain aspects of the codebase, errors are returned to the caller of a function whereas in other cases the program “panics” immediately.

Recommendations:

The error handling of the application should be streamlined throughout to ensure that errors can consistently be retraced to their origin.

Alleviation:

The Kava Labs team convened with the Cosmos-SDK designers and the surrounding development ecosystem to ensure that the error handling of their application is done so in a best-practices compliant manner and stated that they will review it on an on-going basis.

Exhibit 28

TITLE	TYPE	SEVERITY	LOCATION
Redundant “if” Clause	Ineffectual Code	Informational	cdp/keeper/draw.go L72 - L75

[INFORMATIONAL] Description:

See Exhibit 19.

Recommendations:

See Exhibit 19.

Alleviation:

See Exhibit 19.

Exhibit 29

TITLE	TYPE	SEVERITY	LOCATION
Redundant Duplicate Calculation	Ineffectual Code	Informational	cdp/keeper/draw.go L87 - L93

[INFORMATIONAL] Description:

The statement “cdp.Principal.Add(cdp.AccumulatedFees)” is calculated and used twice.

Recommendations:

The result of the operation could instead be stored in a variable that can be referenced twice.

Alleviation:

Our recommendation was applied in full by storing the result of the calculation in an in-memory valuable which is subsequently utilized in the code block twice.

Exhibit 30

TITLE	TYPE	SEVERITY	LOCATION
Redundant Conversions	Ineffectual Code	Informational	cdp/keeper/draw.go L109 - L113

[INFORMATIONAL] Description:

Multiple conversions between “sdk.Int” and “sdk.Coin” occur between those statements whereas the functions that are utilized are already exposed by both interfaces. Additionally, the statement “k.GetDebtDenom(ctx)” is evaluated twice.

Recommendations:

Operations directly on the “sdk.Coin” structs could be carried out. Additionally, the evaluation of “k.GetDebtDenom(ctx)” could be stored in a local variable that is subsequently accessed twice.

Alleviation:

This Exhibit was remediated in accordance to our recommendations, however one more optimization step is possible. The variable “coinsToBurn” could be declared outside the “if” block of L114-L116 and an “else” clause could be introduced that assigns the statement of L112 to “coinsToBurn”. This will ensure that only one “NewCoin” instantiation occurs in this code block as currently, two instantiations can occur when “paymentAmount.GT(cdpDebt)” evaluates to true .

Exhibit 31

TITLE	TYPE	SEVERITY	LOCATION
Redundant Mathematical Calculations	Ineffectual Math	Informational	cdp/keeper/draw.go L203, L209 - L210

[INFORMATIONAL] Description:

These statements are redundant as mathematically “payment” will always be equal to “owed”.

Recommendations:

We advise their omission and proper usage of existing variables.

Alleviation:

This Exhibit has not been dealt with, however it has been taken into consideration by the Kava team and may be fixed in a future commit.

Exhibit 32

TITLE	TYPE	SEVERITY	LOCATION
Redundant “if” Clause	Ineffectual Code	Informational	cdp/keeper/deposit.go L50 - L53

[INFORMATIONAL] Description:

See Exhibit 19.

Recommendations:

See Exhibit 19.

Alleviation:

See Exhibit 19.

Exhibit 33

TITLE	TYPE	SEVERITY	LOCATION
Unoptimized Mathematical Operations	Ineffectual Math	Informational	cdp/keeper/savings.go L31, L47 - L49 & L60

[INFORMATIONAL] Description:

The statements included above utilize the “surplusDistributed” variable, however it is never utilized as is and the results of mathematical operations on it are used instead.

Recommendations:

We advise their simplification by simply storing the remaining surplus rather than the “surplusDistributed” up to each point of the iteration.

Alleviation:

This Exhibit was fully dealt with by following our recommendation of storing the remaining surplus and using it in the various statements of the surrounding code.

Exhibit 34

TITLE	TYPE	SEVERITY	LOCATION
Inconsistent Variable Naming	Coding Style	Informational	cdp/keeper/savings.go L72

[INFORMATIONAL] Description:

The return of “store.Get” is stored in a variable named “bz” throughout the codebase of the audit scope except for this line / statement.

Recommendations:

We advise the uniformity of the codebase by renaming the variable from “b” to “bz”.

Alleviation:

This Exhibit has not been dealt with, however it has been taken into consideration by the Kava team and may be fixed in a future commit.

Exhibit 35

TITLE	TYPE	SEVERITY	LOCATION
Instantiation of “sdk.Coins” with a Single “sdk.Coin”	Ineffectual Code	Informational	auction/keeper/auctions.go L101 & L105

[INFORMATIONAL] Description:

See Exhibit 11.

Recommendations:

See Exhibit 11.

Alleviation:

See Exhibit 11.

Exhibit 36

TITLE	TYPE	SEVERITY	LOCATION
Misleading Logging	Ineffectual Code	Informational	auction/keeper/auctions.go L182 & L246

[INFORMATIONAL] Description:

The “Wrapf” function is incorrectly utilized here as numbers are using the “%s” replacer which is for strings. Additionally, the operator in “Wrapf” is “less than or equal” whereas the conditional on L181 & L245 simply checks “less than”.

Recommendations:

We advise their adaptation to properly log what they are meant to.

Alleviation:

The Kava Labs team correctly stated that the Cosmos SDK implements the fmt.Stringer replacer in all its numerical types and as such the remediation regarding the %s replacer is rendered null. The misleading logging statements were fully corrected in the revised commit hash.

Exhibit 37

TITLE	TYPE	SEVERITY	LOCATION
Redundant Transfers	Ineffectual Code	Informational	auction/keeper/auctions.go L188 & L195

[INFORMATIONAL] Description:

Whenever a bid is replaced, the coins are sent from the new bidder to the module account and then from the module account to the old bidder.

Recommendations:

This can be conducted in a single transaction whereby the funds are sent directly between the bidders.

Alleviation:

This Exhibit has not been dealt with, however it has been taken into consideration by the Kava team and may be fixed in a future commit.

Exhibit 38

TITLE	TYPE	SEVERITY	LOCATION
Redundant Burn Operations	Ineffectual Code	Informational	auction/keeper/auctions.go L197 - L205

[INFORMATIONAL] Description:

Tokens are burnt on each bid and the tokens are first sent to the module account before being burned.

Recommendations:

The tokens could be burned directly from the accounts rather than being transferred to the module account first. Additionally, instead of burning on each bid a single burn transaction could be conducted at the end of the bid cycle.

Alleviation:

This Exhibit has not been dealt with, however it has been taken into consideration by the Kava team and may be fixed in a future commit.

Exhibit 39

TITLE	TYPE	SEVERITY	LOCATION
Redundant Duplicate Calculation	Ineffectual Code	Informational	auction/keeper/auctions.go L198 - L202

[INFORMATIONAL] Description:

Consult Exhibit 29 with regards to statement “”.

Recommendations:

Consult Exhibit 29.

Alleviation:

This Exhibit has not been dealt with, however it has been taken into consideration by the Kava team and may be fixed in a future commit.

Exhibit 40

TITLE	TYPE	SEVERITY	LOCATION
Maximum Bid Race Condition	Race Condition	Major	auction/keeper/auctions.go L244

[MAJOR] Description:

The line above sets the value that the newly placed bid should at least be equal to or greater than to be accepted as a valid bid. To ensure the step does not exceed the maximum permissible bid, a “MinInt” calculation is conducted whereby the minimum of the maximum bid and the previous bid plus the percentage increase is assigned to be the lower-bound value.

If the Maximum Bid has been placed for an auction, it is possible to send an unlimited number of bids that are equal to the maximum bid that will be treated as valid, successfully replacing the previous bidder with zero increase in bidding. As a result, a race condition is introduced whereby the last transaction to replace the previous last bidder of an auction will actually win the auction.

Recommendations:

We advise that the edge case of reaching the maximum bid is handled differently to ensure that this exploit is impossible to replicate.

Alleviation:

After discussion with the Kava team, the race condition indeed cannot occur due to the way "reverse" and "forward" phases work. A sanity check was added that panics, however the sanity

check cannot be evaluated as true under any (normal) circumstance of the code as is. Since this is a sanity check, it should be left as is since its performance impact is negligible.

Exhibit 41

TITLE	TYPE	SEVERITY	LOCATION
Redundant Assignments	Ineffectual Code	Informational	auction/keeper/auctions.go L214, L290 & L359

[INFORMATIONAL] Description:

The assignments of the statements in the aforementioned lines are conducted on each invocation of the function whereas they should be calculated once as they are meant to act as a guard against the execution of an if-block once.

Recommendations:

These assignments should be moved to the “if” blocks of L210-L212, L286-L288 & L355-L357 respectively.

Alleviation:

The redundant assignments were all removed in the codebase of auctions.go and were relocated within the if-block they are meant to guard.

Exhibit 42

TITLE	TYPE	SEVERITY	LOCATION
Misleading Logging	Ineffectual Code	Informational	auction/keeper/auctions.go L324 & L390

[INFORMATIONAL] Description:

Consult Exhibit 35 with a negative (“< 0”) operand and “Wrapf”’s states that it is less than or equal to zero operand. Additionally, zero could be set in the text of “Wrapf” directly.

Recommendations:

Consult Exhibit 35.

Alleviation:

This exhibit was fully dealt with by directly setting the zero within the text and representing the correct operand.

Exhibit 43

TITLE	TYPE	SEVERITY	LOCATION
Redundant Getter Invocation	Ineffectual Code	Informational	auction/keeper/auctions.go L36, L74, L118 & L475

[INFORMATIONAL] Description:

A getter function is invoked in the above statements when an in-memory variable already exists that is equivalent to the getter's return.

Recommendations:

We advise the replacement of the getter invocations with the actual in-memory variable already existent.

Alleviation:

The function call was replaced with the existing "auctionID" variable where applicable according to our recommendation.

Exhibit 44

TITLE	TYPE	SEVERITY	LOCATION
Illegible Variable Naming	Coding Style	Informational	auction/keeper/auctions.go

[INFORMATIONAL] Description:

Almost all instances of “types.CollateralAuctions” are assigned to a variable named “a”.

Recommendations:

We advise that a more descriptive variable name is utilized instead to increase the legibility of the codebase.

Alleviation:

The variable “a” was renamed to “auction” or “auctionType” according to the context, providing a better description of what it represents.

Exhibit 45

TITLE	TYPE	SEVERITY	LOCATION
Ineffectual Loop	Ineffectual Code	Informational	auction/keeper/math.go L18 - L22

[INFORMATIONAL] Description:

The loop checks whether each bucket is negative and if so, panics. However, this is already conducted in other areas of the codebase where this function is executed such as auction/keeper/auctions.go L550-L554.

Recommendations:

We advise the removal of the redundant loop unless the package is utilized in other areas of the codebase not within the scope of the audit.

Alleviation:

This Exhibit has not been dealt with, however it has been taken into consideration by the Kava team and may be fixed in a future commit.

Exhibit 46

TITLE	TYPE	SEVERITY	LOCATION
Illegible Variable Naming	Coding Style	Informational	auction/keeper/math.go L56 - L60

[INFORMATIONAL] Description:

The struct “quoRem” is meant to represent the “quotient” and the “remainder”, however its members are also abbreviated.

Recommendations:

We advise more descriptive names to be utilized for these variables to aid the readers in consuming the codebase. Variable shorthands do not optimize compiled languages as this is a step taken care of by the compiler itself.

Alleviation:

This Exhibit has not been dealt with, however it has been taken into consideration by the Kava team and may be fixed in a future commit.

Exhibit 47

TITLE	TYPE	SEVERITY	LOCATION
Inaccurate Variable Naming	Coding Style	Informational	auction/keeper/math.go L62 - L69

[INFORMATIONAL] Description:

This variable is named “total” whereas it represents a “sum”.

Recommendations:

We advise it be renamed to a more accurate name as “total” usually infers something else.

Alleviation:

This Exhibit has not been dealt with, however it has been taken into consideration by the Kava team and may be fixed in a future commit.

Exhibit 48

TITLE	TYPE	SEVERITY	LOCATION
Incorrect “if” Clause	Ineffectual Code	Informational	auction/keeper/invariants.go L67

[INFORMATIONAL] Description:

“Errorf” states that “endTime after current block time (%s)” whereas the conditional of the block is “a.GetEndTime().Before(currentTime)”. Subsequently, the “if” conditional is incorrect.

Recommendations:

The conditional should be negated to properly reflect what the logging depicts.

Alleviation:

The logging statement was corrected to properly represent the if clauses that precede it.