CERTIK

CertiK Report Kava Labs

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 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 utilizes technologies from blockchain and smart contracts, CertiK team will continue to support the project as a service provider and collaborator.

Executive Summary

This report has been prepared for **Kava Labs** to review the implementation, security and soundness of their **Kava Validator Vesting Module**. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Review the implementation of the validator-vesting mechanism.
- Review the codebase to ensure compliance with current best practices and industry standards.
- Ensuring logic meets the specifications and intentions of the client.
- Cross referencing structure and implementation against specifications given by the client.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Testing Summary

SECURITY LEVEL

Smart Contract Audit

This report has been prepared as a product of the Smart Contract Audit request by Kava Labs. This audit was conducted to discover issues and vulnerabilities in the source code of Kava Labs.

TYPE	Defi
SOURCE CODE	https://github.com/Kava-Labs/kava/releases/tag/v0.3.2
PLATFORM	Cosmos
LANGUAGE	Golang
REQUEST DATE	March 06.2020
DELIVERY DATE	March 10, 2020
METHODS	Dynamic Analysis, Static Analysis, and Manual Review, has been performed.

Findings

TITLE	TYPE	SEVERITY	LOCATION
Error return check missing	Return Value Check	Info	Keeper.go Line 182

[INFORMATIONAL] Description:

One issue was found at keeper.go line 182 k.stakingKeeper. The Undelegate function returns two values sdk.Time and sdk.Error, the code discards both returned values.

Recommendations:

Even given the fact that the code will always return false at that block of the code, handling that error there is a good practice.

Manual Review Notes

Introduction

CertiK team has been engaged by Kava Labs team to audit the design and implementations of its validator-vesting mechanism. The audited source code link:

- https://github.com/Kava-Labs/kava/releases/tag/v0.3.2

The goal of this audit is to review Kava Labs implementation of its business model, general design and architecture, study potential security vulnerabilities, and uncover bugs that could compromise the software in production.

Documentation

We used the following sources in respect to our work:

1. Website: https://www.kava.io/
2. Whitepaper: https://github.com/Kava-Labs/kava
3. Specs : https://github.com/Kava-Labs/kava/tree/master/x/validator-vesting/spec

Summary

The most relevant aspect to consider, and the first that stands out, is the extremely low number of findings, with just one in the total codebase. The focus of the core review was spent on the validator-vesting module, specifically on the proper implementation of the protocol. It is a critical section from a security point of view as it handles funds, accounts that are validating the network and finally rewards or slashes nodes for their participation in the block propagation.

Another interesting aspect to highlight is the use of the Go language in this code. The code was found to have **very good readability, extensive testing casing, and extremely good code design and ethics.**

The use of the "testify" package and the variety of testing cases encapsulated every possible normal scenario. **The documentation of the test cases themselves was also a plus, and showed a standard for code at the highest level.**

To conclude, the review confirmed that the code has **a very good level of protocol implementation and security**, with only one finding, neither critical or low-medium-high risk in nature. The team has made sure that the testing **is at the highest standard along with code ethics and readability**.

Kava Labs Specs Implementation Analysis

The audit focused on the implementation of the validator vesting module of Kava Lab's protocol. Starting off, the codebase presents the various SDK related keeper's interfaces under expected_keepers.go properly encapsulating the functionality expected from those types that are part of the main keeper structure.

Main keeper declaration in keeper.go contains all the necessary types and functions to interact with the SDK, chain and underlying types.

The vesting account functionality is implemented in validator_vesting_account.go and contains the structure of the account, the periodic vesting account and all the necessary collections and functionality for accounting and validation of the vesting nodes.

Finally, the participation checking is implemented in abci.go with the BeginBlocker function that is responsible for the validators accounting on block participation for a period of vesting.

Concepts

[1] The validator-vesting module is responsible for managing Validator Vesting Accounts, a vesting account for which the release of coins is tied to the validation of the blockchain. Validator Vesting Accounts implement the cosmos-SDK vesting account spec, which can be found here.

[2] The main concept the Validator Vesting Account introduces is that of *conditional vesting* or vesting accounts in which it is possible for some or all of the vesting coins to fail to vest. For

Validator Vesting Accounts, vesting is broken down into user-specified **vesting periods**. Each vesting period specifies the number of coins that could vest in that period, and how long the vesting period lasts.

[3] For each vesting period, a **signing threshold** is specified, which is the percentage of blocks that must be signed for the coins to successfully vest. After a period ends, coins that are successfully vested become freely spendable. Coins that do not successfully vest are burned, or sent to an optional return address."

[1] Validator Vesting Account

Implemented in validator_vesting_account.go as ValidatorVestingAccount, the structure contains the periodic vesting account, validators main address, return address to burn tokens, a signing threshold for block signing participation, the current vesting progress for the current period, a collection of the vesting progress and finally the amount of coins in debt after a failed vesting period. All the necessary functions to interact with the chain are in place, well-implemented with proper checks and documentation.

[2] Vesting Periods

A period, declared in period.go is a structure that tracks the length and the amount of coins to be vested in the period.

Implemented in validator_vesting_account.go as Periodic Vesting Account, this account is responsible for the amount of vesting and vesting periods. It contains a Base Vesting Account, a start time of the vesting and a collection of vesting periods (declaration: vesting_account.go line

374). All the necessary functionality is present, well-implemented and with extensive documentation.

[3] Check of Signing Threshold.

The CurrentPeriodProgress implemented in validator_vesting_account.go (line: 33) keeps track of the current period missed and total blocks.The system uses the functions GetSignedPercentage to get the percent of signed blocks within a period of vesting and with SignedPercentageIsOverThreshold to check if the validator has passed the current threshold set. All the functionality is present, well-implemented and with extensive documentation.

State

Validator Vesting Account type

[1] Validator Vesting Accounts implement the cosmos-SDK vesting account spec and extends the PeriodicVestingAccountType:

type ValidatorVestingAccount struct {...}

[1] Implemented in validator_vesting_account.go line 61 contains the periodic vesting account, the validator address, the return address, the signing threshold along the current period progress, vesting period progress and debt after failing vesting. Implementation is well-implemented and designed with extensive documentation.

[Info]

The Signing threshold is an int64 while representing a percentage value in a range 0 to 100.

Int64 represents -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. The code makes sure that this value is within the bounds 0 to 100 in line 187 of validator_vesting_account.go. This is not an issue, as the team most likely would like to keep the type for soundness and readability instead of casting it since sdk. NewDec takes an int64 as argument.

Stores

There is one KVStore in validator-vesting which stores

- **[1]** A mapping from each ValidatorVestingAccount address to Byte{0}
- **[2]** A mapping from previous_block_time_prefix to time.Time

The use of []Byte{0} value for each address key reflects that this module only accesses the store to get or iterate over keys, and does not require storing a value.

[1]

Implemented in keeper.go line 61 function SetValidatorVestingAccountKey.

Store.Set(types.ValidatorVestingAccountKey(addr), []byte{0} and Initialized in genesis.go line 16 function keeper.SetValidatorVestingAccountKey(ctx, vv.Address)

[2] Implemented in keeper.go line 53 with SetPreviousBlockTime.

Store.Set(types.blocktime, b) takes the key and the block time and sets it to the store.

Function GetPreviousBlockTime line 43 returns the previous blocktime from the store.

Both implementations are properly handled and well-implemented.

Begin Block

[1] At each BeginBlock, all validator vesting accounts are iterated over to update the status of the current vesting period. Note that the address of each account is retrieved by iterating over the keys in the validator-vesting store, while the account objects are stored and accessed using the auth module's AccountKeeper. For each account, the block count is incremented, the missed sign count is incremented if the validator did not sign the block or was not found in the validator set. By comparing the block time of the current BeginBlock, with the value of previousBlockTimestored in the validator-vesting store, it is determined if the end of the current period has been reached. If the current period has ended, the VestingPeriodProgress field is updated to reflect if the coins for the ending period successfully vested or not. After updates are made regarding the status of the current vesting period, any outstanding debt on the account is attempted to be collected. If there is enough SpendableBalance on the account to cover the debt, coins are sent to the return address or burned. If there is not enough SpendableBalance to cover the debt, all delegations of the account are Unbonded. Once those unbinding events reach maturity, the coins freed from the undoing will be used to cover the debt. Finally, the time of the previous block is stored in the validator vesting account keeper, which is used to determine when a period has ended.

[1] Implemented in abci.go in the BeginBlocker function by using all the keeper functionality to perform the necessary checks and properly count the block signing participation of the validator with UpdateMissingSignCount accordingly. Additionally, the code handles the outcome of the vesting period after the checks with UpdateVestedCoinsProgress and handles any debt with HandleVestingDebt. Finally it sets the previous block time to the store. The implementation is well orchestrated with proper checking and error handling.All the functionality is present and well documented.

Summary

The audit reviewed the implementation and found it **to be totally aligned with the specs**. Declarations and design **are very clear due to the extensive documentation**. Adding to that the code **is complete, with very good code design and ethics**.

Test Cases and Coverage

The test cases provided by the team in the codebase were found to be more than efficient **with very good code coverage and impressive ethics**. Test cases for every component were found with extensive documentation. The use of "testify" for mocking and the proper error handling in the tests added to our general feeling **of a very thoughtful team regarding the security of the implementation**.

validator-vesting/

coverage: 100.0% of statements

internal/keeper/

coverage: 96.0% of statements

internal/types

coverage: 96.0% of statements