CERTIK

Harvest by Kava Labs

Security Assessment

October 16th, 2020

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Kava
Description	Multi-chain DeFi lending platform
Platform	Cosmos SDK v0.39.1
Codebase	kava
Commits	1.04946493ae927ee7c6dab027f444d32177d82540 2.0ded29678e4e168d20ad89292c470fcff244db95 3.e8a12b5b71ff9115ffe641418011cf63534456e8 4.b6e8759b86a1d75a12dde5a7fd022327f5701588 5.9bbdf1410802a148e61684ef6965a3b345acab2b

Audit Summary

Delivery Date	Oct. 16, 2020
Audit Methods	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Oct. 05, 2020 - Oct. 16 2020

Vulnerability Summary

Total Issues	4
Total Critical	0
Total Major	0
Total Minor	0
Total Informational	4

Executive Summary

Preliminary:

Built on top of the Cosmos SDK, Kava is a multi-asset, interoperable Decentralized Finance (DeFi) platform offering collateralized loans and stablecoins (e.g. USDX), to end-users and other blockchains. The sole objective of the audit is to verify Kava Labs' implementation of the Harvest module, a cross-chain money market, against the provided specifications. A series of thorough security assessments were 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 will also be given.

Alleviations:

All recommendations were addressed and fully attended to in pull request #686.

Review Notes

The primary focus for the audit is to have a thorough look into the following parts of the application:

- Code Structure
- Application Module Interfaces
- Messages and Queries
- Invariants (if present)
- Keepers
- Module Interfaces
- Module Genesis
- Errors

Following a modular design approach outlined in the Cosmos SDK, we carefully inspect the module(s) within scope to ensure that:

1. Application module interfaces (`AppModuleBasic` and `AppModule` at least) are correctly implemented
2. Order of execution between key components of the module are properly manager by `Module Manager`
3. Messages are accompanied by constructor functions, have proper type definition, and correctly implement the `Msg` interface
4. Queries are accompanied by queriers, query commands and query return types
5. Handlers and their corresponding handler functions are properly added and implemented
6. Keepers appropriately expose getter/setter methods for the store(s) managed by the module
7. Invariants are properly implemented and registered
8. Module-specific errors are wrapped to provide additional specific execution context
9. The SDK is utilized in a least-authority manner, primarily for routing messages to their intended modules

Specifically in the Harvest module 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.

State Transition Checks

Claim

Function	Check	Reference	Pass
`ClaimReward`	get a claim from the store	claim.go L25-L28	✓
	validate the claim owner and the reward receiver	claim.go L30-L33	✓
	claim the rewards according to the `depositType` and send them to the reward owner	claim.go L35-L45	✓
	delete the claim from the store	claim.go L56	✓
`GetPeriodLength`	check if lockup period in multiplier is zero	claim.go L64-L66	✓
	calculate the length of the period based on block time and multiplier	claim.go L67-L78	✓
`claimLPReward`	get the liquidity provider's distribution schedule and the multiplier for it	claim.go L83-L90	✓
	check if the claim had expired	claim.go L91-L93	✓
	calculate the reward amount and mint new coins for it	claim.go L94-L98	✓
	get the length of reward period	claim.go L99-L102	✓
	send the rewards to the recipient	claim.go L104	✓

State Transition Checks

Claim

Function	Check	Reference	Pass
`claimDelegatorReward`	get the delegator's distribution schedule and the multiplier for it	claim.go L108-L115	✓
	check if the claim had expired	claim.go L116-L118	✓
	calculate the reward amount and mint new coins for it	claim.go L119-L123	✓
	get the length of reward period	claim.go L125-L128	✓
	send the rewards to the recipient	claim.go L130	✓
`validateSenderReceiver`	get the sender account from the store	claim.go L134-L137	✓
	if the sender account is a validator vesting account, check if the sender address matches to the receiver address. Otherwise, check if the two addresses don't match, in which case an error should be raised	claim.go L138-L147	✓

State Transition Checks

Deposit

Function	Check	Reference	Pass
`ValidateDeposit`	check if the `depositType` is liquidity provider (lp)	deposit.go L54-L62	✓
`ValidateLPDeposit`	check if the `depositDenom` matches the liquidity provider's denom	deposit.go L67-L75	✓
`Deposit`	validate the deposit and transfer the coins from the depositor's account to the module account ONLY IF the depositType is liquidity provider (lp)	deposit.go L14-L27	✓
	check if there's existing deposits in the store under the depositor's account. If there is, add to it. If not, open a new deposit	deposit.go L29-L34	✓
	store the deposit	deposit.go L36	✓
`Withdraw`	get from the store the deposit to be withdrawn, and check if the requested withdrawal amount is larger than the actual amount in the store	deposit.go L79-L85	✓
	transfer the coins from the module account to the original depositor's account ONLY IF the `depositType` is liquidity provider (lp)	deposit.go L88-L96	✓
	subtract the withdrawn amount from the original value and update the deposit amount in the store	deposit.go L121-L122	✓

State Transition Checks

Rewards

Function	Check	Reference	Pass
`ApplyDepositRewards`	get `previousBlockTime` and `Params` from the store; error out if not found and/or inactive	rewards.go L15-L24	✓
	range over `LiquidityProviderSchedules` (lps) and calculate `rewardsDistributed`. Skip if 1) the lps is inactive; 2) lps ends before (or starts after) `blockTime`; 3) `totalDeposited` is zero; or 4) `rewardsToDistribute` is zero	rewards.go L27-L67	✓
	Update `previousBlockTime` in the store	rewards.go L68	✓
`ShouldDistributeValidatorRewards`	get `previousDelegatorDistribution` and `Params` from the store; error out if not found and/or inactive	rewards.go L73-L81	✓
	range over `DelegatorDistributionSchedules` (dds) and check if `timeElapsed` exceeds `DistributionFrequency`. Skip if 1) denom doesn't match the `DepositDenom` in dds; or 2) the `DistributionSchedule` in dds ends before the `blockTime`	rewards.go L82-L93	✓

State Transition Checks

Rewards

Function	Check	Reference	Pass
`ApplyDelegationRewards`	get `delegatorSchedule` from the store; error out if not found, `DistributionSchedule` is inactive, or `DistributionSchedule` starts after `blockTime`	rewards.go L99-L108	✓
	get the coin amount in in the boned pool and check if it is zero	rewards.go L109-L113	✓
	get `previousDelegatorDistribution` from the store and return if not found	rewards.go L114-L117	✓
	calculate `rewardToDistribute`	rewards.go L119	✓
	iterate over all validators and store the key-value pair `ValAddress→conversion` factor in a map (`sharesToTokens`); Continue iteration when the validator has zero tokens and/or is unbonded	rewards.go L123-L134	✓
	iterate over all delegations; calculate and add `rewardsEarned` to the claim if it is not zero	rewards.go L138-L153	✓

State Transition Checks

Timelock

Function	Check	Reference	Pass
`addCoinsToVestingSchedule`	get `PeriodicVestingAccount` from the store	timelock.go L77-L78	✓
	add the new vesting coins to `OriginalVesting`	timelock.go L80	✓
	1) if all vesting periods under the vesting account have completed before blockTime, append a new period to the vesting account → update EndTime → update the account in the store; 2) if the earliest vesting period under the vesting account starts after blockTime, update all vesting periods → set StartTime to now	timelock.go L81-L104	✓
	insert a new vesting period into the existing vesting schedule	timelock.go L107-L143	✓
`SendTimeLockedCoinsToPeriodicVestingAccount`	send time-locked coins from the module account to the recipient	timelock.go L47-L50	✓
	add coins to the input account's vesting schedule	timelock.go L51	✓
`SendTimeLockedCoinsToBaseAccount`	send time-locked coins from the module account to the recipient	timelock.go L57-L60	✓
	transition the account to a periodic vesting account and update it in the store	timelock.go L63-L70	✓
`SendTimeLockedCoinsToAccount`	get the senderModule account from the store and check if it has sufficient balance	timelock.go L19-L22	✓
	get the recipient account from the store; check if the recipient account is a valid account and if the input length is greater than zero; send time-locked coins to the recipient account according to its account type	timelock.go L25-L42	✓

Findings

Status Icon Definitions

✓ Resolved	🚧 In Progress	ℹ️ Ignored (pro)
❌ Not Resolved	❓ Incorrect	🚫 Ignored (con)

Findings Overview

ID	Title	Type	Severity	Status
KAV-01	Verbose Code	Language Best Practices	Informational	✓
KAV-02	Inefficient Conditional Statement	Language Best Practices	Informational	✓
KAV-03	Inefficient Conditional Statement	Language Best Practices	Informational	✓
KAV-04	Unused Parameter	Language Best Practices	Informational	✓

KAV-01: Verbose Code

Type	Severity	Location
Language Best Practices	Informational	querier.go L81

Description:

In the following snippet, the declaration and assignment for a variable (e.g. `depositDenom`) before/after a given logical statement can be replaced with a one-liner.

```
depositDenom := false
owner := false
depositType := false
if len(params.DepositDenom) > 0 {
        depositDenom = true
}
if len(params.Owner) > 0 {
        owner = true
}
if len(params.DepositType) > 0 {
        depositType = true
}
```

Recommendation:

Replace with the following for better readability.

```
depositDenom := len(params.DepositDenom) > 0
owner := len(params.Owner) > 0
depositType := len(params).DepositType > 0
```

Alleviation:

The recommendation was applied in commit 0ded29678e4e168d20ad89292c470fcff244db95.

KAV-02: Inefficient Conditional Statement

Type	Severity	Location
Language Best Practices	Informational	querier.go L97

Description:

The following snippet exhibits a fair amount of complexity with the nested if/else statements, which sacrifices code readability.

```
if depositDenom && owner && depositType {
...
} else if depositDenom && owner {
...
}
...
```

Recommendation:

Replace the nested if/else statments with a switch statement.

```
switch {
case depositDenom && owner && depositType:
...
case depositDenom && owner:
...
case depositDenom && depositType:
...
default:
...
}
```

Alleviation:

The recommendation was applied in commit e8a12b5b71ff9115ffe641418011cf63534456e8.

KAV-03: Inefficient Conditional Statement

Type	Severity	Location
Language Best Practices	Informational	querier.go L249

Description:

The following snippet exhibits a fair amount of complexity with the nested if/else statements, which sacrifices code readability.

```
if depositDenom && owner && depositType {
...
} else if depositDenom && owner {
...
}
...
```

Recommendation:

Replace the nested if/else statments with a switch statement.

```
switch {
case depositDenom && owner && depositType:
...
case depositDenom && owner:
...
case depositDenom && depositType:
...
default:
...
}
```

Alleviation:

The recommendation was applied in commit b6e8759b86a1d75a12dde5a7fd022327f5701588.

KAV-04: Unused Parameter

Type	Severity	Location
Language Best Practices	Informational	querier.go L33

Description:

The following snippet parameter `req` is passed in function `queryGetParams` but not used.

```
func queryGetParams(ctx sdk.Context, req abci.RequestQuery, k Keeper) ([]byte, error) {
    // Get params
    params := k.GetParams(ctx)
    // Encode results
    bz, err := codec.MarshalJSONIndent(k.cdc, params)
    if err != nil {
        return nil, sdkerrors.Wrap(sdkerrors.ErrJSONMarshal, err.Error())
    }
    return bz, nil
}
```

Recommendation:

Remove parameter `req`.

Alleviation:

The recommendation was applied in commit 9bbdf1410802a148e61684ef6965a3b345acab2b.