CERTIK

Kava Labs

Issuance Module

Security Assessment

February 9th, 2021

[Final Report]

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 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 v0.39.2
Codebase	GitHub Repository
Commits	1. d701ae8738502b2c8c379ef81d373d9528e6d81c 2. 118942cca602c6848212d3b84e641a5d1101cc23

Audit Summary

Delivery Date	Feb. 9, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	3
Timeline	Dec. 14, 2020 - Feb. 9, 2021

Vulnerability Summary

Total Issues	4
● **Total Critical**	0
● **Total Major**	0
● **Total Medium**	0
● **Total Minor**	2
● **Total Informational**	2

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 Issuance module, the main mechanism of which allows for a white-listed entity (i.e. issuer) to control the minting and burning of an asset, against the provided specifications. A series of thorough security assessments were carried out, the goal of which is to help 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, a recommendation on fixes and/or mitigation methods will also be given.

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

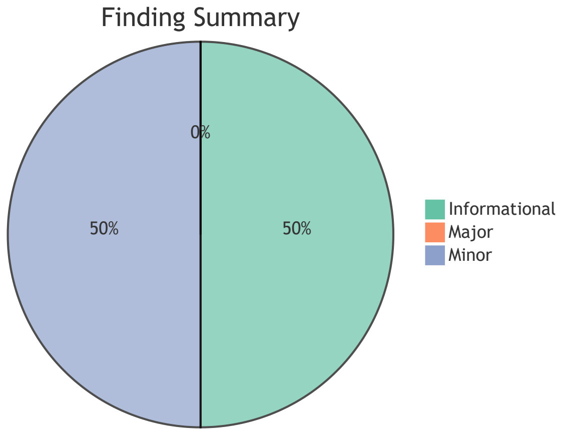
Claim

Function	Check	Reference	Pass
Issue Tokens	Get the asset specified by the denom that's passed in from the params in the store. If not found, return an error	issuance.go L14-L17	✓
	Check if the sender address matches the address of the asset's owner. If not, return an error	issuance.go L18-L20	✓
	Check if the asset has been paused. If true, return an error	issuance.go L21-L23	✓
	Check if the asset is blockable. If true, then check if the receiver is in the blocked list. If the receiver has been blocked, return an error	issuance.go L24-L29	✓
	Check if the receiver address matches that of the module account. If true, return an error	issuance.go L30-L34	✓
	Check if the asset's rate limit is active. If true, increase the asset's supply in the store	issuance.go L37-L42	✓
	If all previous checks are passed, mint the asset in the amount specified. Return any error occurred when minting	issuance.go L45-L48	✓
	Send the newly minted the asset to the receiver. Return any error occurred when sending	issuance.go L50-L53	✓
	Emit event	issuance.go L54-L59	✓
Redeem Tokens	Get the asset specified by the denom that's passed in from the params in the store. If not found, return an error	issuance.go L65-L68	✓
	Check if the sender address matches the address of the asset's owner. If not, return an error	issuance.go L69-L71	✓
	Check if the asset has been paused. If true, return an error	issuance.go L72-L74	✓
	If all previous checks are passed, send tokens from the owner address to the module account. Return any error occurred when sending	issuance.go L75-L79	✓
	Burn the tokens from last step in the module account. Return any error occurred when buring	issuance.go L80-L83	✓
	Emit event	issuance.go L84-L89	✓

Function	Check	Reference	Pass
Block Addesses	Get the asset specified by the denom that's passed in from the params in the store. If not found, return an error	issuance.go L95-L98	✓
	Check if the asset is unblockable. If not, return an error	issuance.go L99-L101	✓
	Check if the sender address matches the address of the asset's owner. If not, return an error	issuance.go L102-L104	✓
	Check if the supplied address has already been blocked. If yes, return an error	issuance.go L105-L108	✓
	Check if the supplied address exists in the state machine. If not, return an error	issuance.go L109-L112	✓
	If all previous checks are passed, add the supplied address to the blocked list and update the asset in the store	issuance.go L113-L114	✓
	Emit event	issuance.go L115-L121	✓
Unblock Addresses	Check if the supplied address exists in the state machine. If not, return an error	issuance.go L127-L130	✓
	Check if the asset is unblockable. If not, return an error	issuance.go L131-L133	✓
	Check if the sender address matches the address of the asset's owner. If not, return an error	issuance.go L134-L136	✓
	Check if the supplied address has already been unblocked. If yes, return an error	issuance.go L137-L142	✓
	If all previous checks are passed, remove the supplied address from the blocked list, update the blocked list for the asset, and finally update the asset in the store	issuance.go L144-L146	✓
	Emit event	issuance.go L147-L153	✓

Function	Check	Reference	Pass
Set Pause Status	Get the asset specified by the denom that's passed in from the params in the store. If not found, return an error	issuance.go L159-L162	✓
	Check if the sender address matches the address of the asset's owner. If not, return an error	issuance.go L163-L165	✓
	Check if the asset's existing Paused status matches the supplied status. If true, return back out	issuance.go L166-L168	✓
	Flip the asset's Paused status and update the asset in the store	issuance.go L169-L170	✓
	Emit event	issuance.go L171-L177	✓
Seize Coins	Get all assets from the params in the store and range over them. If an asset is blockable, attempt to seize from the blocked list the coins specified by the given denoms	issuance.go L183-L191	✓
	Get the asset specified by the denom that's passed in from the params in the store. If not found, return an error	issuance.go L197-L200	✓
	Range over all addresses in the blocked list for that asset, and perform the following checks:	issuance.go L201-L202	✓
	Check if the supplied address exists in the state machine. If not, continue	issuance.go L202-L207	✓
	Check if the balance of the blocked account is positive. If not, continue	issuance.go L208-L211	✓
	Send the coins from the blocked account to the module account. Return any error when sending	issuance.go L212-L216	✓
	Send the coins from the module account to the asset owner account. Return any error when sending	issuance.go L217-L220	✓
	Emit event	issuance.go L221-L227	✓

Findings



Status Icon Definitions

✓	**Resolved**	🚧	**In Progress**	**i**	**Ignored (pro)**
✗	Not Resolved	?	Incorrect	🛇	Ignored (con)

Findings Overview

ID	Title	Type	Severity	Status
KAV-01	Ambiguous Comments	General	Informational	✓
KAV-02	Redundant else clause	Language Usage	Informational	✓
KAV-03	Ambiguous Conditional Statement	Implementation	Minor	✓
KAV-04	Inefficient use of `append` Statement	Implementation	Minor	**i**

KAV-01: Ambiguous Comments

Type	Severity	Location
General	Informational	params.go L117-L118

Description:

Assets is best described as a slice of Asset.

Recommendation:

Correct the verbiage as described above.

Alleviation:

The recommendation was applied in commit 118942cca602c6848212d3b84e641a5d1101cc23.

KAV-02: Redundant `else` Clause

Type	Severity	Location
Language Usage	Informational	supply.go L59-L69

Description:

The `else` clauses can be eliminated by keeping the negative path in the `if` clause, and pulling the positive path out of the `else` clause.

Recommendation:

Keep the positive path in a straight line of sight for best readability.

Alleviation:

The recommendation was applied in commit 118942cca602c6848212d3b84e641a5d1101cc23.

KAV-03: Ambiguous Conditional Statement

Type	Severity	Location
Implementation	Minor	issuance.go L138-L142

Description:

The `if` clause on Line 139 is counterproductive as it blocks the `return` on Line 140 when the supplied address is already unblocked. As a result, Line 144 will try to remove an already unblocked address from the blocked list.

Recommendation:

Remove the `if` clause on Line 139.

Alleviation:

The recommendation was applied in commit 118942cca602c6848212d3b84e641a5d1101cc23.

KAV-04: Inefficient use of `append`

Type	Severity	Location
Language Usage	Minor	issuance.go L113

Description:

During a mutation `append` first makes a copy of the origin slice and then compares the length of the copied slice with its capacity. When equal, a new backing array with doubled capacity will be allocated on the heap, resulting a costly allocation. As a general note, avoid using `append` unless in a decoding or unmarshaling function.

Recommendation:

As a mitigation method, we recommend replacing `append` entirely with the following:

```
asset.BlockedAddresses[len(asset.BlockedAddresses)] = blockedAddress
```

Alleviation:

This exhibit was discussed in length with the Kava team. Though not addressed, we consider the exhibit fully attended to as it doesn't impose any meaningful security concerns.