CERTIK

CertiK Audit Report for Microtick

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 Microtick (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

Microtick is a decentralized price oracle built on the Cosmos SDK. Microtick offers market-based execution of shorter-term at-the-money options, and provides a unique solution to achieving price discovery on nearly any asset.

To that end, the sole objective of the audit is to verify Microtick's implementation of the Microtick module against the specifications, and discover potential issues and vulnerabilities in the codebase. The audit was conducted through a series of thorough security assessments, 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

Decentralized Oracle Audit

This report has been prepared as a product of the Decentralized Oracle Audit request by Microtick. This audit was conducted to discover issues and vulnerabilities in the source code of Microtick's blockchain implementation. Work was completed between 06/19/2020 — 7/09/2020

TYPE	Decentralized Price Oracle
SOURCE CODE	mjackson001/mtzone:master/13c5059c68a7322fa6da41d6031ebc8d3f9f575b
PLATFORM	Cosmos SDK v0.38.4
LANGUAGE	Golang \ go version go1.14 linux/amd64
SERVER, CLIENT, VERSION	mtd \ mtcli \ rc7
BUILD TAGS	build_host=manticore;build_date=Mon 13 Jul 2020 10:39:11 AM MDT
METHODS	Whitebox Analysis Dynamic Analysis, Static Analysis, and Manual Review

Review Notes

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: microtick module
- State transitions in each module were carefully verified against their specification
- 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 Microtick 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.

Key Checks

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

Code Structure	Application Module Interfaces	Messages and Queries	Handlers
Queriers	Keepers	Module Interfaces	Module Genesis
Errors	Invariants (if present)	BeginBlocker and EndBlocker (if present)	

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. 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

Audit Revision

The Microtick team took our Exhibits into account and decided to proceed optimizing their codebase according to our recommendations. A patch release was provided and accessed, the link of which can be found below:

- mjackson001/mtzone:master [13c5059c68a7322fa6da41d6031ebc8d3f9f575b]

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

Audit Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Code Structure	Coding Style	Informational	N/A

[INFORMATIONAL] Description:

We would like to make the following recommendations in regards to project structure:

- It's recommended to have the module's CLI and REST clients in their own directories under x/microtick/client/ as follows:
  - cli/
    - query.go
    - tx.go
  - rest/
    - rest.go
    - query.go
    - tx.go
- In continuation of the last point, we recommend the following:
  - Moving GetTxCmd from x/microtick/client/module_client.go to x/microtick/client/cli/tx.go
  - Moving GetQueryCmd from x/microtick/client/module_client.go to x/microtick/client/cli/query.go
  - Moving all query handlers from x/microtick/client/rest.go to x/microtick/client/cli/query.go
- x/microtick/handler.go should be named abci.go as it only implements EndBlocker at the moment. The actual handlers for the module are defined and registered in x/microtick/router.go
- The handler functions for all defined message types currently reside in their own files under x/microtick/msg, mixed in with the files where the querier functions are defined.

This causes a significant amount of confusion and doesn't quite follow the best practices of building custom modules using the Cosmos SDK. It is recommended to implement the handlers and their corresponding handler functions all in x/microtick/handler.go. Also, please make sure the keeper's setter functions are ONLY called if all checks (stateless and stateful) are successful.

- The querier functions for the querable data types currently reside in their own files under x/microtick/msg, mixed in with the files in which the handler functions are defined. It is recommended to implement the queriers and their corresponding querier functions all in x/microtick/keeper/querier.go
- As advised above, x/microtixk/router.go would be better off split into a x/microtixk/handler.go where the handlers are defined/registered, and a x/microtixk/keeper/querier.go where the queriers are defined/registered
- The query return types specifying the result type of each of the module's queries currently reside in their own files under x/microtick/msg, mixed in with the querier functions. It is recommended to define the query return types in x/module/types/querier.go instead
- We recommend implementing Invariants in x/microtick/keeper/invariants.go, one Invariant function per logical grouping of Invariants. This will help detect bugs early on and act upon them to mitigate the damage of potential consequences (e.g. a chain halt)

Revision:

✔️ folder structure under x/microtick/client/ restructured

✔️ `GetTxCmd` moved to `x/microtick/client/cli/tx.go`

✔️ `GetQueryCmd` moved to `x/microtick/client/cli/query.go`

✔️ `x/microtick/handler.go renamed to x/microtick/abdi.go`

This exhibit was partially addressed in the patch release. Considering those were recommendations and do not impose any substantial security threats, we conclude the exhibit has been fully attended to.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Missing Genesis Validity Check	Coding Style	Informational	Reference

[INFORMATIONAL] Description:

It is a general practice to perform validity checks on the Genesis files prior to booting the blockchain to ensure that the genesis files are structurally and functionally sound.

Recommendations:

ValidateGenesis is not currently implemented. It is strongly recommended to perform validity checks on each of the parameters listed in GenesisState. We also recommend placing genesis.go under x/microtick/types/.

Revision:

✔️ `ValidateGenesis` implemented in `/x/microtick/types/genesis.go`

This exhibit was fixed in the patch release and hence we consider the issue raised originally was addressed in full.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Incomplete Type Definition in REST Requests	Coding Style	Informational	Reference

[INFORMATIONAL] Description:

The type definition of REST requests in the REST server implementation lack proper attribute definition.

Recommendations:

Each request type, suffixed Req, should include a base request baseReq, the name of the transaction, and all arguments that must be provided for the transaction.

Revision:

- ✔ Incomplete Type Definition in REST Request (`signedReq`) removed

This exhibit was fixed in the patch release and hence we consider the issue raised originally was addressed in full.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Duplicate Function Names	Coding Style	Informational	Reference

[INFORMATIONAL] Description:

Duplicate function names increase the illegibility of the codebase and render it harder to consume by external readers.

Recommendations:

InitGenesis on L124 is defined in x/microtick/genesis.go, and called here in a method with the same name, causing the aforementioned confusion in regards to the normal course of execution.

Revision:

This exhibit was not addressed in the patch release. Considering it was a recommendation and does not impose any substantial security threats, we conclude the exhibit has been fully attended to.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Redundant else Clause	Ineffectual Code	Informational	Reference

[INFORMATIONAL] Description:

The else clause in the aforementioned code segment contains a break statement within.

Recommendations:

The break statement could be moved outside the else block and after the execution of the if block as it is the last statement of the for loop.

Revision:

This exhibit was not addressed in the patch release. Considering it was a recommendation and does not impose any substantial security threats, we conclude the exhibit has been fully attended to.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Redundant Variable Declarations	Ineffectual Code	Informational	Reference I, II

[INFORMATIONAL] Description:

These variable declarations are followed by “declare and assign” Golang statements “:=”.

Recommendations:

Thus, these variable declarations can be completely omitted.

Revision:

✔️ Redundant declarations removed

This exhibit was fixed in the patch release and hence we consider the issue raised originally was addressed in full.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Improper Error Handling	Ineffectual Code	Minor	Reference I, II, III, IV

[MINOR] Description:

The GetDataMarket call of these lines can return an error which is otherwise unchecked in the statements mentioned above. This will lead to an empty struct being returned and subsequently utilized.

Recommendations:

We advise that proper error handling is introduced at these points in the codebase.

Revision:

✔️ Proper error handling of GetDataMarket added

This exhibit was fixed in the patch release and hence we consider the issue raised originally was addressed in full.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Redundant else Return Blocks	Ineffectual Code	Informational	Reference I, II, III

[INFORMATIONAL] Description:

The aforementioned else blocks all contain a direct return statement.

Recommendations:

As return statements halt the execution of the function and the if clause they are contained within is the last statement of the code block they are included in, it is possible to completely omit the else block and move the return statements at the end of the preceding if clause.

Revision:

✔️ Redundant else return blocks removed

This exhibit was fixed in the patch release and hence we consider the issue raised originally was addressed in full.

Exhibit 9

TITLE	TYPE	SEVERITY	LOCATION
Heavy Pass-By-Value Functions	Ineffectual Code	Informational	Reference I, II

[INFORMATIONAL] Description:
Overall, the application is meant to handle complex structs that are heavy memory-wise and occupy a lot of space. Wherever they are used, they are declared as in-memory variables and as such are passed by-value rather than by-referenced causing a non-negligible operational overhead.

Recommendations:
We recommend that the whole codebase is examined to ensure that wherever copies of data are passed are done so with sense. Automated tools can assist in this endeavour as our tools indicated over 198 instances of this issue occurring.

Revision:
This exhibit was not addressed in the patch release. Considering it was a recommendation and does not impose any substantial security threats, we conclude the exhibit has been fully attended to.

Exhibit 10

TITLE	TYPE	SEVERITY	LOCATION
Ineffectual Empty String Comparison	Ineffectual Code	Informational	Reference I, II, III

[INFORMATIONAL] Description:

The statements above contain a comparison of the length (len) of a string with the number zero (0) to check whether the string is an empty string.

Recommendations:

This is inefficient as it is possible to directly compare the string with an empty one, instead replace len(msg.Market) == 0 with msg.Market == “”.

Revision:

✔️ len(msg.Market) == 0 replaced with msg.Market ==””

This exhibit was fixed in the patch release and hence we consider the issue raised originally was addressed in full.

Exhibit 11

TITLE	TYPE	SEVERITY	LOCATION
Builtin Variable Shadowing	Coding Style	Informational	Reference

[INFORMATIONAL] Description:

The variable len is a utility built-in function of Golang, however the statement above declares a local variable called len.

Recommendations:

This type of variable shadowing is ill-advised as it makes the codebase harder to comprehend and is misleading with regards to how the variable len actually behaves in the code. We advise the renaming of the variable to something more sensible.

Revision:

✔️ len renamed

This exhibit was fixed in the patch release and hence we consider the issue raised originally was addressed in full.