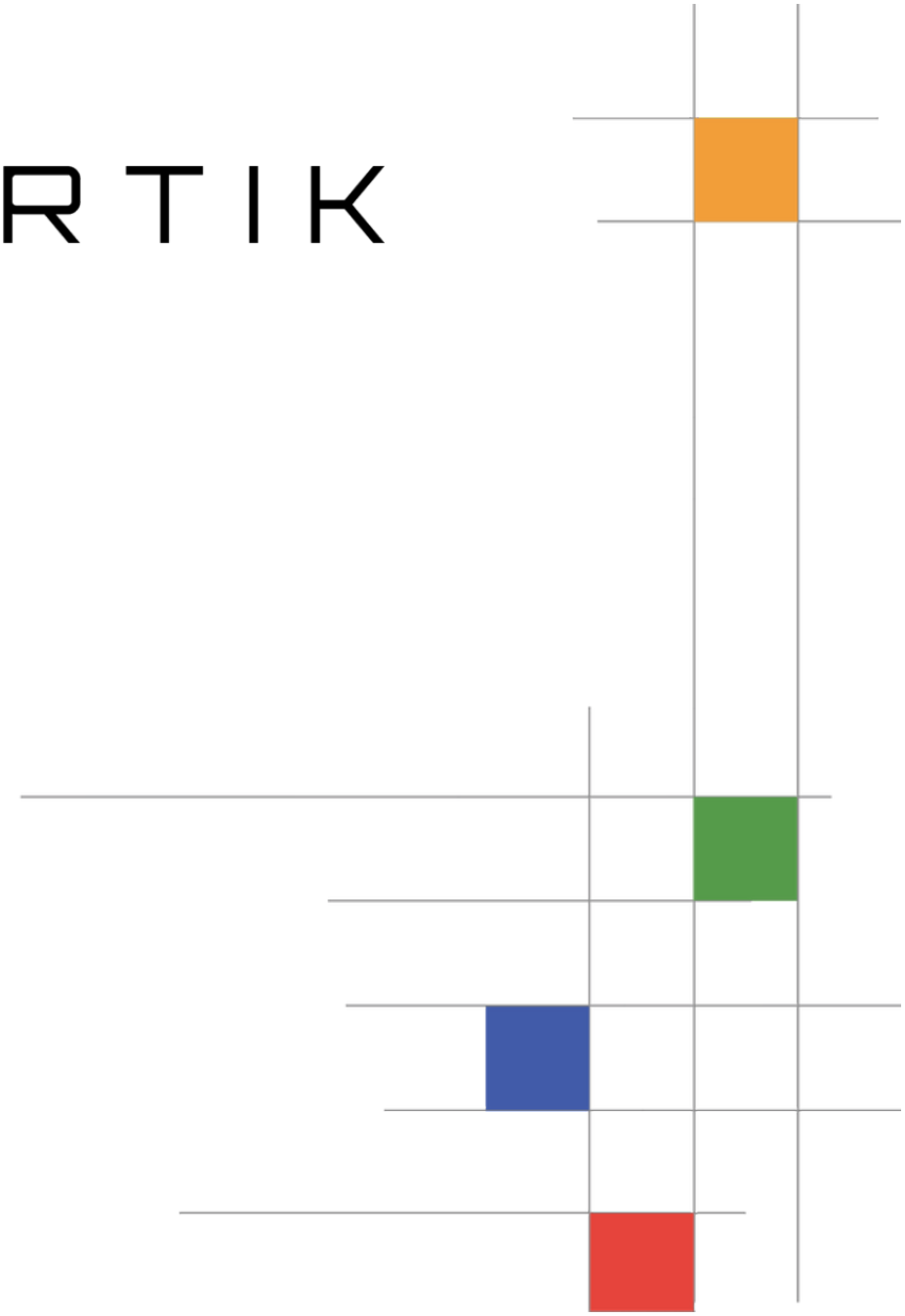
Liquid Chain

Security Assessment

May 13th, 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	Liquid Chain
Description	Replicated State Machine
Platform	Native
Codebase	GitHub Repository
Commits	1. 23e744792565aa45ff6b23f32dc6a77685fcb963

Audit Summary

Delivery Date	Apr 9th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Mar 12th, 2021 - Apr 9th, 2021

Vulnerability Summary

Total Issues	24
Total Critical	0
Total Major	0
Total Medium	1
Total Minor	2
Total Informational	21

Executive Summary

Liquid Chain is a high-performance, language-agnostic blockchain built on top of the Tendermint consensus engine. With Tendermint being the consensus and P2P layers, the Liquid Chain implements its own replicated state machine consisting of:

- Virtual Machine
- Blockchain Core
- Storage

At the core of the Liquid Chain, the Virtual Machine (VM) enables deterministic smart contract execution at low latency, supporting a wide range of development languages. With the Blockchain Core coordinating cross-component communications, execution events in the Liquid Chain can include multiple values and are able to return complex data structures. As transactions are made and put into blocks, the Storage packs them into what's called the Block DB, separated from the State DB where smart contract is stored.

Liquid Chain has appointed CertiK to review and verify the implementation of the Liquid Chain poject against its specifications. A series of thorough security assessments have been 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.

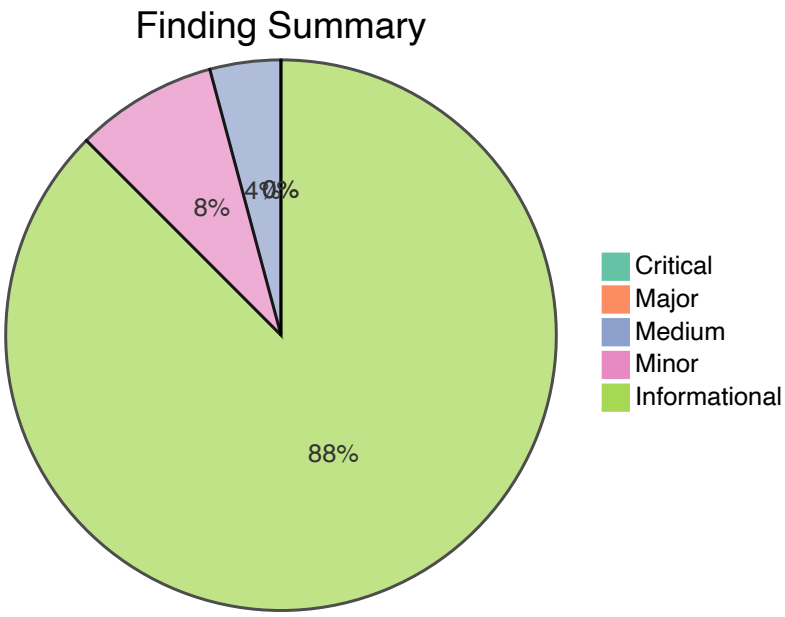
Review Notes

Our audit approach primarily revolves around a multi-round manual review of the codebase, and largely favors modularity and encapsulation in code design. At a high level we analyze each object (or module) 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. The primary focus for the audit is to have a thorough look into the following components of the Liquid Chain application:

- Virtual Machine
- Blockchain Core
- Storage
- Fee Model

Specifically we analyze how the state machines are defined and how state transitions are triggered, the goal of which is to check the implementation against the specs (if provided) and hence minimize the possibilities of unintentional state behaviors taking place.

Findings



Status Icon Definitions

✓ Resolved	In Progress	Ignored (pro)
✗ Not Resolved	? Incorrect	Ignored (con)

Findings Overview

ID	Title	Type	Severity	Resolved
LIQ-153	Discrepancy between Code and Comment	Language Usage	Informational	✓
LIQ-140	Discrepancy between Code and Comment	Language Usage	Informational	✓
LIQ-236	Unnecessary Decoupling	Language Usage	Informational	✓
LIQ-180	Possible Method ID Collision	Implementation	Minor	✓
LIQ-146	Direct use of key seed for tx signing	Implementation	Medium	✓
LIQ-195	Inappropriate Command Naming	Language Usage	Informational	✓
LIQ-152	Discrepancy between Code and Comment	Language Usage	Informational	✓
LIQ-147	Discrepancy between Code and Comment	Language Usage	Informational	✓
LIQ-221	Redundant else-if Clause	Implementation	Informational	✓
LIQ-232	Ambiguous Function Name	Language Usage	Informational	✓
LIQ-182	Unsafe uint64 Multiplication	Implementation	Informational	✓
LIQ-329	Ambiguous Function Name	Language Usage	Informational	✓
LIQ-233	Ambiguous Function Name	Language Usage	Informational	✓
LIQ-224	Missing Error Handling	Implementation	Minor	✓
LIQ-339	Duplicated Code	Implementation	Informational	✓
LIQ-272	Redundant Function	Language Usage	Informational	✓
LIQ-321	Redundant Return	Language Usage	Informational	✓
LIQ-254	Redundant Use of len()	Language Usage	Informational	✓
LIQ-226	Redundant Code in Adding Commands	Language Usage	Informational	
LIQ-226	Ambiguous Data Semantics	Language Usage	Informational	✓

LIQ-200	Unidiomatic Function Naming	Language Usage	Informational	✓
LIQ-188	Unsafe uint64 Multiplication	Implementation	Informational	✓
LIQ-181	Unsafe uint64 Multiplication	Implementation	Informational	✓
LIQ-257	Unnecessary Type Declaration	Language Usage	Informational	✓

LIQ-153: Discrepancy between Code and Comment

Type	Severity	Location
Language Usage	Informational	chain.go L135

Description:

The comment suggests that `GetBlockReceipts` returns transactions of a given block while in reality it handles the receipts.

Recommendation:

Correct comment as suggested.

Alleviation:

The recommendation was applied in commit 99e78302a9aad3e0007676e5d5bbcefcb8bd5643.

LIQ-140: Discrepancy between Code and Comment

Type	Severity	Location
Language Usage	Informational	contract.go L20

Description:

The comment suggests that `GetContract` delivers transactions to blockchain while in reality it fetches contracts from the state.

Recommendation:

Correct the comment as suggested.

Alleviation:

The recommendation was applied in commit b6fd1b11fca43500a79cae422adb2f399b030421.

LIQ-236: Unnecessary Decoupling

Type	Severity	Location
Language Usage	Informational	parser.go L16-L78

Description:

The two highlighted methods are unnecessarily decoupled from the concrete data they are meant to handle. Methods do give a piece of data behavior, but data having behavior should be an exception, not a rule. Decoupling a piece of concrete data comes at the cost of indirection and allocation. In case of a double-indirection when accessing the data, since escape analysis can't track whether the data should stay on the stack or not, the data will most likely end up on the heap, which is a very costly during runtime.

Recommendation:

Make `parseParam` and `parseFunction` functions, instead of methods.

Alleviation:

The recommendation was applied in commit 25a4d9cfc50cafc8c5c79f45c7d98f673c173945.

LIQ-180: Possible Method ID Collision

Type	Severity	Location
Implementation	Minor	method_signature.go#L5 -

Description:

The hash byte length is rather short, and therefore can be prone to method ID collision. During fuzzing we tested with random strings for possible collisions, and it took around 100k signatures to get 1 collision. The chances of a signature collision between two random strings came out to be roughly `1/4,000,000,000` which we consider nonnegligible.

Recommendation:

Increase the hash byte length.

Alleviation:

In commit 655e0a6c8a824f06841caf5352d4e103b5635aff and fb5d75a950d5454e550d95dd5d82be809aebe733 the Liquid team added proper error handling for collisions. We consider this exhibit addressed in full.

LIQ-146: Direct use of key seed for tx signing

Type	Severity	Location
Implementation	Medium	cli/cli.go L161

Description:

The highlighted code unsafely loads the raw seeds of a key from the command line.

Recommendation:

Instead of requiring raw seeds to be passed in, encrypt the key before loading it in.

Alleviation:

The file was removed in commit e2dfd0cb4d99320b4e8572ae4f108daf171d5cda and we hence consider the exhibit addressed in full.

LIQ-195: Inappropriate Command Naming

Type	Severity	Location
Language Usage	Informational	cli/cli.go L150

Description:

Read-only functions in smart contracts are normally referred to as `View` or `query` functions.

Recommendation:

Refine the terminologies following the `Deploy → Invoke → Query` pattern.

Alleviation:

The file was removed in commit e2dfd0cb4d99320b4e8572ae4f108daf171d5cda and we hence consider the exhibit addressed in full.

LIQ-152: Discrepancy between Code and Comment

Type	Severity	Location
Language Usage	Informational	block.go L42

Description:

The comment suggests that the `Receipts` method returns transactions of a block while in reality, as the name would suggest, it returns the receipts of a block.

Recommendation:

Correct the comment as suggested.

Alleviation:

This exhibit was fully attended to in commit a88716e6b44dee9dcc3133ca419914cd4344e337.

LIQ-147: Discrepancy between Code and Comment

Type	Severity	Location
Language Usage	Informational	receipt.go L27-L32

Description:

The comment suggests that the `encode` method returns a byte representation of transaction while the receiver of `encode` is actually `receipt`.

Recommendation:

Correct the comment as suggested.

Alleviation:

This exhibit was fully attended to in commit 5db43bbac77df9148af1fac6f109263669f20108.

LIQ-221: Redundant else-if Clause

Type	Severity	Location
Implementation	Informational	util.go L61-L69

Description:

The `err` returned on `L61` is either going to be `nil` or `function ... not found`, thus the else-if clause on `L69` is redundant and can be safely removed.

Recommendation:

Remove the else-if clause on `L69`.

Alleviation:

This exhibit was fully attended to in commit 445582a7ec094bf2807e6147960a7dc06e369b34.

LIQ-232: Ambiguous Function Name

Type	Severity	Location
Language Usage	Informational	station.go L12-L13

Description:

In the highlighted functions, the return variables don't carry over the intention displayed by the function names.

Recommendation:

Add comments explaining what the function is intended to do and what is expected to be returned.

Alleviation:

This exhibit was fully attended to in commit 99cf98cb2921583f014f69344fcb71cb52d01c76.

LIQ-182: Unsafe uint64 Multiplication

Type	Severity	Location
Implementation	Informational	execution.go L62

Description:

The highlighted operation does not check for possible arithmetic overflow.

Recommendation:

Check the product of multiplication against possible overflow.

Alleviation:

This exhibit was acknowledged by the Liquid team and ultimately discarded, with the comment that the highlighted operation couldn't possibly result in ovrflow as both `GasUsed` and `GasPrice` were declared as `uint32`. We consider the exhibit fully attended to as it doesn't impose any meaningful security concerns.

LIQ-329: Ambiguous Function Name

Type	Severity	Location
Language Usage	Informational	token.go L59-L61

Description:

The function name would've suggested that `GetContract` returns the contracts owned by the provided account. In actuality `GetContract` returns the account address of the contract.

Recommendation:

Correct the function name as suggested.

Alleviation:

This exhibit was fully attended to in commit db9cdded72002b609281a3b688d2150d5885f53c and d93fe83b4765217d998f41ca152869eaeedacf23.

LIQ-233: Ambiguous Function Name

Type	Severity	Location
Language Usage	Informational	trie.go L139-L139

Description:

Generally in `trie` an update is perceived as updating the value associated with a provided key. In the highlighted function however, `Update` modifies the key based on the passed in value.

Recommendation:

Divide up `Update` into two separate functions for inserting and deleting into the `trie`, respectively.

Alleviation:

This exhibit was acknowledged by the Liquid team and ultimately discarded, with the comment that were `Update` split into three separate functions, the trie interface would be complex to use. We consider the exhibit fully attended to as it doesn't impose any meaningful security concerns.

LIQ-224: Missing Error Handling

Type	Severity	Location
Implementation	Minor	execution.go L146-L152

Description:

`increaseNonce` silently ignores the error when incrementing the nonce on an account that does not yet exist in the state.

Recommendation:

Return an error to the caller if `` and the caller can decide how to handle it.

Alleviation:

This exhibit was acknowledged by the Liquid team and ultimately discarded. As remediation a comment was added in commit 74ba5723b07a82a023c1cbf86663145a4cb1b652, providing context as to why the use of `account == nil` is valid. We consider the exhibit fully attended to as it doesn't impose any meaningful security concerns.

LIQ-339: Duplicated Code

Type	Severity	Location
Implementation	Informational	execution.go L19-L22

Description:

`deployContract` and `invokeContract` share much of the same functionalities and would be better off merged into one.

Recommendation:

Merge said functions and mark them in events or parameters.

Alleviation:

This exhibit was acknowledged by the Liquid team and ultimately discarded, with the comment that by keeping `deployContract` and `invokeContract` separate not only does it keep function signatures simple, but also allows for better maintainability and upgradability. We consider the exhibit fully attended to as it doesn't impose any meaningful security concerns.

LIQ-272: Redundant Function

Type	Severity	Location
Language Usage	Informational	wasi_adapter.go L22-L24

Description:

`wasiDefaultHandler` isn't warranted to be a standalone function.

Recommendation:

Remove `L22-L24` and replace `L17` with

```
return 0, fmt.Errorf("unsupported func call %s", name)
```

Alleviation:

This exhibit was acknowledged by the Liquid team and ultimately discarded, commenting that shall there be additional logics to be added necessary measures will be taken. We consider the exhibit fully attended to as it doesn't impose any meaningful security concerns.

LIQ-321: Redundant Return

Type	Severity	Location
Language Usage	Informational	adapter.go L117-L122

Description:

The error type in return is redundant as it will be always `nil`.

Recommendation:

Remove error from return.

Alleviation:

This exhibit was acknowledged by the Liquid team and ultimately discarded, with the comment that the `vm` interface requires `(uint64,error)` response. We consider the exhibit fully attended to as it doesn't impose any meaningful security concerns.

LIQ-254: Redundant Use of len()

Type	Severity	Location
Language Usage	Informational	node.go L146

Description:

The use of `len()` is redundant as the expected length of the hash is already declared in `common/types.go`.

Recommendation:

Replace `len(common.Hash{})` with `common.HashLength`.

Alleviation:

The recommendation was applied in commit 5eb23bff95694c07347f76fb9fda99cc0e1e2d7a.

LIQ-226: Redundant Code in Adding Commands

Type	Severity	Location
Language Usage	Informational	node.go L26-L29

Description:

`addDefaultCommands` on `L29` is partially redundant as root commands would've been added by `commands.RootCmd` on `L26` already.

Recommendation:

Remove `L26` and replace `L35` with `node.commands = commands.RootCmd.AddCommand(...)`.

Alleviation:

LIQ-226: Ambiguous Data Semantics

Type	Severity	Location
Language Usage	Informational	call.go L16

Description:

Field `Height` is declared using pointer semantic, yet used using value semantic throughout the codebase. It's technically ok to pass in a pointer when the function specifically asks for copy, but it's not idiomatic and against Golang's best practices.

Recommendation:

Declare field `Height` using value semantics instead.

Alleviation:

This exhibit was acknowledged by the Liquid team and ultimately discarded, with the comment that the `Height` field is intentionally declared with an nullable value in anticipation of an edge case where a nil `Height` is possible. We consider the exhibit fully attended to as it doesn't impose any meaningful security concerns.

LIQ-200: Unidiomatic Function Naming

Type	Severity	Location
Language Usage	Informational	policy.go L10-L12

Description:

Per Golang best practices, it's neither idiomatic nor necessary to put `Get` into the getter's name.

Recommendation:

Remove the `Get` prefix from function name.

Alleviation:

This exhibit was acknowledged by the Liquid team and ultimately discarded. We consider the exhibit fully attended to as it doesn't impose any meaningful security concerns.

LIQ-188: Unsafe uint64 Multiplication

Type	Severity	Location
Implementation	Informational	execution.go L78

Description:

The highlighted operation does not check for possible arithmetic overflow.

Recommendation:

Check the product of multiplication against possible overflow.

Alleviation:

This exhibit was acknowledged by the Liquid team and ultimately discarded, with the comment that the recommended check wouldn't be necessary for two numbers of a multiplication that are declared in `uint32`. We consider the exhibit fully attended to as it doesn't impose any meaningful security concerns.

LIQ-181: Unsafe uint64 Multiplication

Type	Severity	Location
Implementation	Informational	execution.go L119

Description:

The highlighted operation does not check for possible arithmetic overflow.

Recommendation:

Check the product of multiplication against possible overflow.

Alleviation:

This exhibit was acknowledged by the Liquid team and ultimately discarded, with the comment that the recommended check wouldn't be necessary for two numbers of a multiplication that are declared in `uint32`. We consider the exhibit fully attended to as it doesn't impose any meaningful security concerns.

LIQ-257: Unnecessary Type Declaration

Type	Severity	Location
Language Usage	Informational	meta_key.go L37-L39

Description:

Type `metaKeyPrefix` does not serve any real purposes. In Golang you are only really encouraged to declare new custom types if they are new and unique.

Recommendation:

Instead of

```
type num int

func Foo(n number) {...}
```

Do

```
func Bar(num int) {...}
```

Alleviation:

The recommendation was applied in commit 18f466cf74f1ef79c4e3b266b2aff0b7eff80379.