CERTIK

CertiK Audit Report for Tellor

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 Tellor (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, in partnership with grants from IBM and the Ethereum Foundation, CertiK's mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that projects are checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and assessments to each 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 deliverable. For more information: https://certik.io.

Executive Summary

This report has been prepared for **Tellor** to discover issues and vulnerabilities in the source code of their **Smart Contracts** as well as any contract dependencies that were not part of an officially recognized library. 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:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Testing Summary

SECURITY LEVEL

TBA

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

TYPE	Smart Contracts
SOURCE CODE	https://rinkeby.etherscan.io/address/0x6cc73a7cb32b5978da026b8fb8a33c080a9a0fe4#code
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	June 8 2020
DELIVERY DATE	July 16, 2020
METHODS	A comprehensive examination has been performed using Dynamic Analysis, Static Analysis, and Manual Review.

Review Notes

Introduction

CertiK team was contracted by the Tellor team to audit the design and implementations of their Smart Contracts. The audited source code linkis:

- https://rinkeby.etherscan.io/address/0x6cc73a7cb32b5978da026b8fb8a33c080a9a0fe4#code

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

Documentation

The sources of truth regarding the operation of the codebase were extensive and well documented. Although the in-line comments were not as properly documented as the other resources, the documentation aided our understanding of the specification implementation and the functionality of the code.

Summary

Overall, the codebase of the project does not conform to the Solidity style guide as evident by certain very long lines as well as inconsistent spacing between the statements.

While most of the issues pinpointed were of negligible importance and mostly referred to coding standards and inefficiencies, **any minor (or above) flaws** that were identified, **should be remediated as soon as possible to ensure** the contracts of Tellor's team are of **the highest standard and quality.**

These inefficiencies and flaws can be swiftly dealt by the development team behind the Tellor project so that a second round of auditing can proceed. We will create and maintain a direct communication channel between us and the Tellor team to aid in amending the issues identified in the report.

Recommendations

With regard to the codebase, the main recommendation we can make is **to apply linters**. Although the documentation is thorough, the in-line comments should provide more descriptive information to the reader. Additionally, we advise that all our findings are carefully considered and assimilated in the codebase of the project to ensure the highest code standard is achieved.

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report, enforce linters and/or coding styles as well as correct any spelling errors and mistakes that appear throughout the code **to achieve a high standard of code quality and security.**

Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Unlocked Compiler Version Declaration & Different versions of Solidity used	Language Specific Issue	Informational	Line 5

[INFORMATIONAL] Description:

The compiler version utilized throughout the project uses the “>=” and “<” prefix specifiers, denoting that a compiler at or above the version included after the specifier should be used to compile the contracts. Also, the compiler version should be consistent throughout the codebase.

Recommendations:

It is a general practise to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Naming Convention Utilization	Coding Style	Informational	General

[INFORMATIONAL] Description:

Solidity defines a naming convention that should be followed. In general, the following naming conventions should be utilized in a Solidity file:

- Contracts should be in CapWords
- Functions and parameters should be in mixedCase
- Constants should be in UPPER_CASE_WITH_UNDERSCORES

Recommendations:

The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Conversion to Inequality Comparison	Optimization	Informational	Lines 79, 146, 208, 225, 298, 365, 637, 853, 1078, 1360, 1361, 1409, 1512

[INFORMATIONAL] Description:

When comparing unsigned integers (integers >= 0), it is a good practice to check for the edge case, meaning the instance where the value of the integer is not equal to zero, instead of checking all the values greater than zero.

Recommendations:

Convert to inequality comparisons as the variables they utilize are guaranteed to pass by the bound they are currently being compared against.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Code Redundancy	Optimization	Informational	Line 123

[INFORMATIONAL] Description:

In general, code redundancy should be avoided. In our case, the comparison in Line 123 is already checked in Line 107.

Recommendations:

Remove unnecessary code.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Search Algorithm Optimization	Optimization	Informational	Lines 129 - 133

[INFORMATIONAL] Description:

Although the binary search algorithm is a very powerful one, the implementation is not fully optimized, as the current iteration executes the worst-case scenario of binary search at all times.

Recommendations:

Restructure the “if-else” statement so that the "if" clauses should check if "fromBlock" is equal to "_block" and return it at that point directly.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Change Variable Type to Enum	Coding Style	Informational	Lines 146

[INFORMATIONAL] Description:

Enums are useful when you have a variable that can take one of a series of predefined values

Recommendations:

Consider changing the variable “currentStatus” to an enum.

Example:

enum MinerStatus { NOT_STAKED, STAKED, LOCKED_FOR_WITHDRAW, ON_DISPUTE }

MinerStatus currentStatus;

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Description in the Comments	Coding Style	Informational	Line 147

[INFORMATIONAL] Description:

In general, explaining the variable purpose or the function implementation with some comments is coding standard, although the comments should be very descriptive so that the reader should get the exact functionality without misconception.

Recommendations:

Change the comment to say “Subtract”.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
“allowedToTrade” Function Optimization	Optimization	Informational	Lines 146 - 154

[INFORMATIONAL] Description:

The code segment uses SafeMath subtractions ("sub" invocations) which guarantee that the value they return is above or equal to zeroif they succeed.

Recommendations:

Consider adjusting the checks to not make redundant subtractions internally and multiple checks.

Exhibit 9

TITLE	TYPE	SEVERITY	LOCATION
Conversion to Inequality Comparison	Optimization	Informational	Line 163

[INFORMATIONAL] Description:

It is a good practice to check for the edge case, meaning the instance where the value of the “fromBlock” is not equal to “block.number”, instead of checking all the former’s values lesser than the latter.

Recommendations:

Consider converting the second comparison of "if" clause to an inequality comparison instead.

Exhibit 10

TITLE	TYPE	SEVERITY	LOCATION
Greater-Than Comparison with Zero	Coding Style	Informational	Line 164

[INFORMATIONAL] Description:

It is recommended not to change read-only variables like “checkpoints.length”.

Recommendations:

Consider using the "push" pattern here.

Exhibit 11

TITLE	TYPE	SEVERITY	LOCATION
“Require” Function Error Message	Coding Style	Informational	General

[INFORMATIONAL] Description:

“require” can be used to check for conditions and throw an exception if the condition is not met, in which case the error message provided by the developer will appear. This is why a very descriptive error message is needed.

Recommendations:

Consider changing the error message string to be more detailed.

Consider adding an error message in Line 1650

Exhibit 12

TITLE	TYPE	SEVERITY	LOCATION
Format Long Lines	Optimization & Coding Style	Informational	General

[INFORMATIONAL] Description:

It is not recommended to have long lines into your codebase, as they can get confusing and reduce code readability.

Recommendations:

Split long lines to multiple steps with proper formatting.

Exhibit 13

TITLE	TYPE	SEVERITY	LOCATION
Event Restructure	Coding Style	Informational	Line 318

[INFORMATIONAL] Description:

Solidity events give an abstraction on top of the EVM's logging functionality. Applications can subscribe and listen to these events or search for them. When you call Events, they cause the arguments to be stored in the transaction's log - a special data structure in the blockchain.

Recommendations:

Restructure the "Voted" event to contain the "voteWeight" data.

Exhibit 14

TITLE	TYPE	SEVERITY	LOCATION
Misleading/Unconventional Assignment	Coding Style	Informational	Line 425

[INFORMATIONAL] Description:

It is unclear to the reader why the variable "disputeRound" is increased to one in this scenario. It is recommended that any reader should be able to understand the reason for specific implementations.

Recommendations:

Add a description explaining the reasoning behind this assignment.

Exhibit 15

TITLE	TYPE	SEVERITY	LOCATION
Edge Case Identification	Volatile Code	Informational	Line 569

[INFORMATIONAL] Description:

If no stakes exist, then the code in Line 574 will fail.

Recommendations:

No recommendation.

Exhibit 16

TITLE	TYPE	SEVERITY	LOCATION
Missing Variable Description	Coding Style	Informational	Line 743

[INFORMATIONAL] Description:

Missing description of the variable “currentStatus” value equal to four (4).

Recommendations:

Fully describe the predefined values of the variable “currentStatus”.

Exhibit 17

TITLE	TYPE	SEVERITY	LOCATION
Description Correction	Optimization	Informational	Lines 822

[INFORMATIONAL] Description:

Function “getMax” returns the maximum value in an array. It is incorrectly described in the comments that it returns the minimum value.

Recommendations:

Change the word “minimum” with “maximum”.

Exhibit 18

TITLE	TYPE	SEVERITY	LOCATION
Variable/Function Naming Convention	Coding Style	Informational	Lines 1040, 1455, 1938

[INFORMATIONAL] Description:

In general, it is a convention when naming variables or functions to be in singular form if the value of “single” types (like integers, addresses, booleans or unsigned integers) and in plural form when it contains “multiple” values (like arrays or mappings).

Recommendations:

Consider changing the name of the function from “getDisputeUnitVars” to “getDisputeUnitVar”, as it returns a “uint256” value.

Consider changing the name of the variable from “_topId” to “_topIds”, as it returns an array of “uint256” values.

Consider changing the name of the function parameter from “_requestId” to “_requestIds”, as it returns an array of “uint256” values.

Exhibit 19

TITLE	TYPE	SEVERITY	LOCATION
Variable/Function Naming Convention	Coding Style	Informational	Lines 1110, 1669

[INFORMATIONAL] Description:

Naming the variables or the function should describe its purpose or its functionality.

Recommendations:

Consider changing the name of the mapping from “minersByValue” to “minersByTimestamp”.

Consider changing the name of the mapping from “valuesByTimestamp” to “valuesByMinerIndex”.

Exhibit 20

TITLE	TYPE	SEVERITY	LOCATION
Comparison Optimization	Optimization	Informational	Line 1363

[INFORMATIONAL] Description:

It is very efficient, both for the readers and the developers, to check equality or inequality in comparisons, as it provides a better code readability and quality, by targeting edge cases.

Recommendations:

Consider changing the condition comparison to:

“_requestId == self.uintVars[keccak256("requestCount")] + 1”

Exhibit 21

TITLE	TYPE	SEVERITY	LOCATION

Update “updateOnDeck” Function	Coding Style	Informational	Line 1710

[INFORMATIONAL] Description:

Function “updateOnDeck” neither returns a value nor emits an event.

Recommendations:

Consider updating the “updateOnDeck” function to either return a value or emit an event.

Exhibit 22

TITLE	TYPE	SEVERITY	LOCATION
“updateOnDeck” Function Optimization	Optimization	Informational	Line 1714

[INFORMATIONAL] Description:

The existing code can be optimized, as the line is long.

Recommendations:

Consider using a “for” loop outside of the “if” statement to check for the value of “_requestId”.

Exhibit 23

TITLE	TYPE	SEVERITY	LOCATION
Change of Ownership Procedure	Coding Style	Informational	Line 1702

[INFORMATIONAL] Description:

Change of ownership could be restructured to change the address of the “pending_owner” after he changes roles.

Recommendations:

Consider resetting the “pending_owner” to “address(0)” immediately after the “pending_owner” becomes an “owner” .