CERTIK

CertiK Audit Report for HUMAN Protocol

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 HUMAN Protocol (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 **HUMAN Protocol** to discover issues and vulnerabilities in the source code of their **HMT Solidity 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

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

TYPE	Smart Contract
SOURCE CODE	https://github.com/hCaptcha/hmt-escrow
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	Jun 26, 2020
DELIVERY DATE	Sept 15, 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 HUMAN Protocol's team to audit the design and implementations of HUMAN Protocol's HM Token Smart Contract and Escrow Smart Contracts.

The audited source code link:

- https://github.com/hCaptcha/hmt-escrow/tree/4b0838173e2a3d61d51433b8ead8e618d4d58712

The remediated source code link:

- https://github.com/hCaptcha/hmt-escrow/tree/be76032e496eae79c40c8678cd9ef94beb169a5c

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.

The findings of the initial audit have been conveyed to the team behind the contract implementations and the source code is expected to be re-evaluated before another round of auditing has been carried out.

Documentation

The sources of truth regarding the operation of the codebase were extensive and well documented. The in-line comments were also richly documented and together with the other resources, the documentation greatly aided our understanding of the specification implementation and the functionality of the code.

Summary

The project's codebase is a typical EIP20 token implementation, along with an Escrow mechanism.

Although **certain optimization steps** that we pinpointed in the source code mostly referred to coding standards and inefficiencies, **any minor flaws** that were identified **should be remediated as soon as possible to ensure** the contracts of HUMAN Protocol's team are of **the highest standard and quality.**

Overall, the codebase has been refactored to assimilate this report's findings in order **to achieve a high standard of code quality and security**.

Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Missing Visibility Modifier	Coding Style	Informational	Escrow: L16

[INFORMATIONAL] Description:

It is generally a bad coding style not to add a visibility modifier for a variable. In this case, the variable `canceler` is missing.

Recommendations:

Add a visibility modifier for this variable.

Alleviations:

The team opted to consider our references and added a visibility modifier to the variable `canceler`.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Incorrect `struct` Utilization	Optimization	Informational	Escrow: L34-L37

[INFORMATIONAL] Description:

Consider changing the struct `TrustedHandler`, as the address `handlerAddress` of the struct is never used (because of mapping in line 39).

Recommendations:

Consider removing the struct `TrustedHandler` and only use the `TrustedHandlers` mapping for this procedure, as seen below:

mapping(address => bool) public areTrustedHandlers;

Alleviations:

The team opted to consider our references, removed the `TrustedHandler` struct and used the `TrustedHandlers` mapping to match the `Trust` factor of a specific address.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Conversion to Inequality Comparison	Optimization	Informational	Escrow: L184, L227 EscrowFactory: L47 HMToken: L56

[INFORMATIONAL] Description:

When comparing numbers, it is a good practice to check for the edge case. An example would be when comparing unsigned integers that are not equal to zero, instead of checking all the values greater than zero. Another example would be when comparing an integer to its max value. One should be checking the inequality to the max value, instead of checking all the values lower than the max value.

Recommendations:

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

Alleviations:

The team opted to consider our references and changed each linked conditional with an inequality comparison.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Code Redundancy	Optimization	Informational	Escrow: L158, L234 HMToken: L116, L138, L156

[INFORMATIONAL] Description:

In general, code redundancy should be avoided.

On Lines 158 & 234 of the “Escrow” file, the assignment of false is redundant.

On Lines 116 & 138 of the “HMToken” file, the assignment of zero is redundant.

On Line 156 of the “HMToken” file, the use of SafeMath’s subtractions ("sub" invocations) is redundant due to the conditional on Line 154.

Recommendations:

Remove or change unnecessary code.

Alleviations:

The team opted to consider our references and removed redundant code.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Unclear Variable Naming	Coding Style	Informational	Escrow: L44

[INFORMATIONAL] Description:

The variable `_expiration` could better be called `duration` as it is not exactly referring to expiration.

Recommendations:

Consider changing the name of the variable `_expiration` to `duration`.

Alleviations:

The team opted to consider our references and changed the name of the variable to `duration`.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Change Visibility Modifier	Coding Style	Informational	Escrow: L57-L71, L81-L108 EscrowFactory: L24-L39

[INFORMATIONAL] Description:

In general, we prefer compiler generated getters over user defined ones as they are less error prone.

Recommendations:

Consider changing the visibility modifiers of the variables from `private` to `public` and removing the “getter” functions.

Alleviations:

The team opted to consider our references, changing the visibility modifiers of the variables from `private` to `public` and removed the “getter” functions.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Unconventional Error Messages	Coding Style	Informational	Escrow: L135, L139

[INFORMATIONAL] Description:

We prefer avoiding misleading and/or incorrect error messages, as they are meant to lead the reader in the correct direction as to why this error occurred.

Recommendations:

Consider changing the error messages.

Alleviations:

The team opted to consider our references and added descriptive error messages.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Code Optimization	Optimization	Informational	Escrow: L142, L143

[INFORMATIONAL] Description:

There is a double use of SafeMath's addition ("add" invocations).

Recommendations:

Consider storing the duplicate addition to `memory` variable for a more efficient `storage` management.

Alleviations:

The team opted to consider our references, declared the `totalStake` variable, assigned the value of the `_reputationOracleStake.add(_recordingOracleStake)` operation to this variable and then also used this variable to the following `require` statement.

Exhibit 9

TITLE	TYPE	SEVERITY	LOCATION
Potential Loss of Tokens	Volatile Code	Informational	Escrow: L166-L174

[INFORMATIONAL] Description:

If the status of an “Escrow” is partial, then tokens can be permanently lost.

Recommendations:

No recommendation.

Alleviations:

No alleviations.

Exhibit 10

TITLE	TYPE	SEVERITY	LOCATION
Potential Use of “require” Function	Optimization & Coding Style	Informational	Escrow: L200-L202, L265-L267

[INFORMATIONAL] Description:

We prefer using the `require` function when possible for security.

Recommendations:

Consider using the `require` function instead of the `if` block for the status of an “Escrow”.

Alleviations:

The team opted to consider our references and used a `require` statement instead of the `if` block for the sub-Exhibit A.

Exhibit 11

TITLE	TYPE	SEVERITY	LOCATION
Code Optimization	Optimization & Coding Style	Informational	Escrow: L259-L260

[INFORMATIONAL] Description:

We prefer to avoid multiple consecutive assignments to a variable. In this case, it is more efficient to assign a value to the variable `bulkPaid` once and properly format it for code readability.

Recommendations:

Consider using the "AND" operator to assign a value to the boolean variable `bulkPaid` as follows:

```
bulkPaid   =    token.transfer(reputationOracle, reputationOracleFee) &&
                token.transfer(recordingOracle, recordingOracleFee);
```

Alleviations:

The team opted to consider our references and assigned a value to the boolean variable `bulkPaid` as described.

Exhibit 12

TITLE	TYPE	SEVERITY	LOCATION
Redundancy & Code Optimization	Optimization	Informational	Escrow: L268-L272

[INFORMATIONAL] Description:

It is not recommended to check for the same condition multiple times, if possible. In this case, the first term of the "OR" operator is already being checked by the previous \`if\` block.

Recommendations:

Consider changing the \`if\` statement as follows:

if (balance == 0 && status == EscrowStatuses.Partial)

as the lines 265-267 check the first term of the "OR" operator.

Alleviations:

The team opted to consider our references and changed the \`if\` statement as described.

Exhibit 13

TITLE	TYPE	SEVERITY	LOCATION
Potential Fee Trim	Coding Style	Informational	Escrow: L285, L288

[INFORMATIONAL] Description:

Potential trimming of the fee, as there is a use of SafeMath’s division ("div" invocation) right after SafeMath’s multiplication ("mul" invocation).

Recommendations:

No recommendation.

Alleviations:

No alleviations.

Exhibit 14

TITLE	TYPE	SEVERITY	LOCATION
Missing Return Value	Coding Style	Informational	EscrowFactory: L16, L22

[INFORMATIONAL] Description:

It is not recommended to be vague about the return value of a function, as this will make things more difficult for the reader to understand.

Recommendations:

Consider specifying the return value of the function `createEscrow()`.

Alleviations:

The team opted to consider our references and specified the return value of the `createEscrow()` to be the `lastEscrow` variable.

Exhibit 15

TITLE	TYPE	SEVERITY	LOCATION
Use of Magic Number	Coding Style	Informational	EscrowFactory: L17

[INFORMATIONAL] Description:

We prefer to avoid the usage of magic numbers (literals with no explanation).

Recommendations:

Magic number ‘8640000’ should be declared constant and documented.

Alleviations:

The team opted to consider our references and added in-line documentation for the linked magic number.

Exhibit 16

TITLE	TYPE	SEVERITY	LOCATION
Code Optimization	Optimization	Informational	EscrowFactory: L33-L34

[INFORMATIONAL] Description:

We prefer to avoid unnecessary code. In this case, a value is assigned to a variable and then the variable is returned, instead of simply returning the value in the first place.

Recommendations:

Consider skipping Line 33 and just use:

return escrowCounters[address]

Alleviations:

No alleviations, as the team opted to entirely remove the “getter” function.

Exhibit 17

TITLE	TYPE	SEVERITY	LOCATION
Unconventional Code	Optimization & Coding Style	Informational	HMToken: L20

[INFORMATIONAL] Description:

This line of code can be considered as an illegal statement in conventional solidity code.

Recommendations:

Consider utilizing `~uint256(0)`.

Alleviations:

The team opted to consider our references and opted to change the code block as described.

Exhibit 18

TITLE	TYPE	SEVERITY	LOCATION
Redundancy & Code Optimization	Optimization & Coding Style	Informational	HMToken: L50, L53, L57

[INFORMATIONAL] Description:

The former part of the "AND" condition in the `require` statement on Line 50 will also be checked by SafeMath's subtractions ("sub" invocation) on Lines 53 and 57. Also, some versions of SafeMath implement a subtraction ("sub" function) with an extra parameter for the error message.

Recommendations:

Consider either implementing a wrapper "sub" function that takes an extra parameter for the error message or use a version of SafeMath that has already implemented such function.

Alleviations:

The team opted to consider our references, removing the former part of the "AND" condition in the `require` statement and using SafeMath's "sub" invocation with error messages.

Exhibit 19

TITLE	TYPE	SEVERITY	LOCATION
Potential High Gas Operation	Coding Style	Informational	HMToken: L68-L74

[INFORMATIONAL] Description:

This block of code is prone to race conditions of approval, which could lead to high gas transactions.

Recommendations:

No recommendation.

Alleviations:

No alleviations.