PUBLIC

Code Assessment

of the HTX Smart Contract

January 15, 2024

Produced for

HTX DAO

by

CHAINSECURITY

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help HTX DAO with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of HTX according to Scope to support you in forming an opinion on their security risks.

HTX DAO implements the HTX token following the TRC-20 standard with immutable parameters set at deployment, including its name, symbol, decimals, and an initial fixed supply minted to the deployer. The token enables standard functionalities such as transfer and approval. Importantly, it does not allow for additional token minting, ensuring a fixed supply. It operates without any roles endowed with special privileges.

The most critical subjects covered in our audit are functional correctness, security of the assets and adherence to the TRC-20 specification. Security regarding all the aforementioned subjects is high.

The general subjects covered are energy efficiency and usability. The code is derived from a legacy OpenZeppelin implementation originally written for Solidity version 0.4.24. While it has been adapted for compilation with Solidity 0.8.x, it does not utilize newer Solidity features, such as built-in SafeMath or immutables. Consequently, the code is not optimal, particularly in terms of energy consumption.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code hash (md5sum) which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code file. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Source Code Hash	Note
1	02 Jan 2024	da3096efbc63148e0023f68f09ad47e9	Initial Version

For the solidity smart contract, the compiler version `0.8.20`, adapted for Tron was chosen.

2.1.1 Excluded from scope

The Solidity compiler, particularly the adapted version for Tron, is assumed to function correctly and is out of scope.

Furthermore, the source code reviewed was retrieved from the chain explorer Tronscan; this version is assumed to be the correct, unaltered contract for our review. While the verification of the contract's source code integrity is out of the scope of our analysis, we have checked that the actual bytecode deployed on the network matches the bytecode resulting from compiling this source code.

2.2 System Overview

HTX is a simple TRC20 token with basic functionality (transfer, allowances). The token parameters (name, symbol, decimals) and the initial amount minted to the deployer are set as deployment parameters. No additional tokens can be minted afterwards, and more generally there is no role at the smart contract level that has special privileges.

At the time of this review the HTX token is already deployed to the TRON blockchain at address `TUPM7K8REVzD2UdV4R5fe5M8XbnR2DdoJ6`. The configuration of this token is as follows: name: `HTX`, symbol: `HTX`, decimals: `18`, initial amount (immutable total supply) `999_990_000_000_000 * 10**18`. It is important to note that the decimals are used solely for visualization purposes. All token amounts are handled in their 'raw' representation.

The following public functions are present:

- `transfer()`: Allows `msg.sender` to transfer tokens to a specific address.
- `transferFrom()`: Transfers tokens from one address to another. `msg.sender` requires sufficient allowance from the `from` address. HTX does not emit `Approval` event from this function.
- `approve()`: Enables `msg.sender` to authorize a given address to spend up to a specified amount of tokens on their behalf. However, changing an allowance with this method brings the risk that someone may use both the old and the new allowance due to transaction ordering. This is a known "double spend" issue in approvals of ERC20/TRC20 tokens.
- `increaseAllowance()`: Allows `msg.sender` to increase the amount of tokens that the given address can spend on behalf. This function can help mitigate the double spend issue in the `approve()` function by avoiding the need to set the allowance to zero before increasing it.

- `decreaseAllowance()`: Enables `msg.sender` to decrease the amount of tokens that an address is authorized to spend.

Public view functions:

- `totalSupply()`: Returns the total number of tokens in existence.
- `balanceOf()`: Given an address, it returns the balance of tokens held by that address, allowing users to check their token balance or the balance of others.
- `allowance()`: Reports the amount of tokens that an owner has allowed a spender to use.

In addition there are getter functions exposed for `name`, `symbol` and `decimals`.

The implementation is based on a legacy OpenZeppelin implementation (#83bc045), and adapted for Solidity 0.8.x.

2.3 Trust Model & Roles

Deployer:

- Role: Initially owns all minted tokens. The deployer maintains the `origin_address` role for the smart contract on the Tron blockchain, as detailed in the Tron developers' guide. The `origin_address` has limited capabilities: it can clear the ABI, set the `origin_energy_limit`, and adjust the `consume_user_resource_percent`.
- Trust Level: Trusted entity.
- Responsibilities: Responsible for deploying the contract with the correct constructor parameters and the initial distribution of tokens. This initial setup is crucial for defining the token's characteristics (name, symbol, decimals, amount).

Token Holder:

- Role: Addresses holding HTX tokens.
- Trust Level: Untrusted. Any entity or individual who possesses HTX tokens becomes a token holder, regardless of their intentions or behavior.
- Responsibilities: Can engage in transactions using HTX tokens, including transferring tokens to others and approving spenders.

Approved Spender:

- Role: An address that has been given an allowance by a token holder to use a specified number of their tokens.
- Trust Level: Semi-trusted. They are trusted by the token holder to a certain extent, as defined by the set allowance.
- Responsibilities: Can use up to the allowed number of tokens on behalf of the token holder.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe our findings. The findings are split into these different categories:

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

6.1 Floating Pragma

Informational **Version 1**

CS-HTX-001

The contract HTX uses the floating pragma `^0.8.0`. Although this contract has been compiled with Tron's Solidity version `0.8.20` and is already deployed, we would like to note that contracts should always be deployed with the compiler version and flags that were used during testing and auditing. Locking the pragma helps to ensure that contracts are not accidentally deployed using a different compiler version and help ensure a reproducible deployment.

6.2 Optimizations

Informational **Version 1**

CS-HTX-002

The codebase could be more efficient in terms of energy usage. Reducing the energy costs may improve user experience. Below is a list of potential inefficiencies:

1. The contract HTX uses Solidity version `0.8.x` which, by default, implements overflow and underflow checks. Therefore, the use of library SafeMath is redundant and could be avoided.
2. Furthermore, the functions `mul()`, `div()` and `mod()` of the library SafeMath remain unused.
3. The internal functions `_burn()` and `_burnFrom()` are unused in the codebase.
4. The storage variable `_decimals` could be immutable as it is only set in the constructor. This would reduce the number of storage operations made when `decimals()` gets called.
5. The functions `increaseAllowance()` and `decreaseAllowance()` perform a redundant SLOAD operation when emitting the `Approval` event.

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 Locked Assets

Note Version 1

Tokens (TRX, TRC-20 or similar) could be accidentally/intentionally sent to the HTX contract. In that case the tokens will be locked, with no way to recover them. Incidents in the past showed this is a real issue as there always will be users sending tokens to the token contract.

Note that TRX and tokens can be forced into any contract and get locked if there is no "recover" function.