Chain Integrity

The Web 3.0 Ethical Hacking Company

Libera Trade

Smart Contract Audit Deliverable

Date: Apr 23, 2022

Version: 1

Table of Contents

1.1 Summary

Project

Name	Libera Trade
Description	Supply Chain, AI, Tokenization
Platform	Polygon Network
Codebase	*
Commit	*

Engagement

Delivered	Apr 23, 2022
Methods	Static Analysis, Manual Review, TM, RA.
Consultants	2
Timeline	2 Days

Observations

Total	1	**Status**
Critical	0	
High	0	
Medium	0	
Low	1	Pending
Undetermined	0	

Executive

This document has been prepared for Libera Trade (Client) to discover and analyze the smart contract provided by the team for security vulnerabilities, code correctness, and risks. The smart contract has been comprehensively examined using structural analysis, behavioral analysis, and manual review techniques.

Throughout the audit, caution was taken to ensure that the smart contract(s):

- Implements robust functions which are safe from well-known attack vectors.
- The logic and behavior adheres to the associated documentation and code comments.
- Transfer flows are designed in a sustainable way, safeguarded from i.e. infinite loops.
- Does not hide potential back doors and implements sanity checks where suitable.

1.2 Scope

File	Fingerprint (SHA-256 Checksum)
LIBE.sol	2491a5c06271c6d581bfb5fcd8350a46578d339a9cbe823ae69673513b86796e

1.3 Documentation

The smart contract in scope is documented partially but the styling guide and code layout have a quality decreasing effect on the readability and clarity of the code. Next, some of the code comments do not align with the actual implementation causing confusion and incorrectness. Finally, all observations are explicitly based on the information available in the provided codebase and whitepaper.

1.4 Review Notes

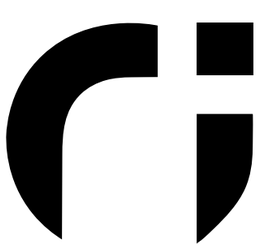
0 security issues were observed during the audit of the smart contract in scope.
The smart contract is inheriting industry-standard smart contracts.

1.5 Recommendations

The inheritance graph should use the same compiler version as the main smart contract to prevent unexpected behavior and undisclosed bugs. Make sure that the code is present with the latest updates.

1.6 Disclaimer

It should be noted that this document is not an endorsement of the effectiveness of the smart contracts, rather limited to an assessment of the logic and implementation. This audit should be seen as an informative practice with the intent of raising awareness on the due diligence involved in secure development and make no material statements or guarantees in regards to the operational state of the smart contract(s) post-deployment. Chain Integrity (Consultant) do not undertake responsibility for potential consequences of the deployment or use of the smart contract(s) related to the audit.



For clarity of understanding, observations are arranged from critical to informational. The severity of each issue is evaluated based on the risk of exploitation or other unexpected behavior.

Critical

An issue flagged as critical means that it can affect the smart contract in a way that can cause serious financial implications, catastrophic impact on reputation, or disruption of core functionality.

High

An Issue flagged as high means that it can affect the ability of the smart contract to function in a significant way i.e. lead to broken execution flows or cause financial implicants.

Medium

An Issue flagged as medium means that the risk is relatively small and that the issue can not be exploited to disrupt execution flows or lead to unexpected financial implications.

Undetermined

An issue flagged as undetermined means that the impact of the discovered issue is uncertain and needs to be studied further.

Low

An Issue flagged as informational does not pose an immediate threat to disruption of functionality, however, it should be considered for security best practices or code integrity.

2.2 Observations List

ID	Title	Type	Severity
OBSN-01	**Inconsistent Compiler Versioning**	Code Correctness	

Unchanged Improved Fixed

OBSN-01

Location: LIBE.sol

Explanation:

The inheritance graph of smart contracts should be using the same compiler version and fixed to prevent unexpected behavior from undiscovered bugs in other compiler versions.

Recommendation:

Lock the pragmas to the version which has been used for the development and testing of the smart contract. Some of the inherited contracts are outdated make sure you are using the latest versions of the inherited smart contracts.

No security issues were found during the assessment and the smart contracts in scope pass our auditing process.

Smart contracts will offer greater visibility and tighter integrations enabling a more efficient and leaner supply chain with strong safety monitoring and compliance.

The smart contracts in scope adhere to best practices in terms of security and it is primarily influenced by code publicized by OpenZeppelin.

A. Classification List

TYPE	DESCRIPTION
Gas Optimization	Gas Optimization findings refer to code improvements that do not affect the functionality of the code but execute more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Mathematical Operations	Mathematical Operations details findings related to mishandling of mathematical formulas such as integer overflows, an incorrect order of operations, or percentage precisioning et al.
Logical Issue	Logical Issue findings defines faults in the logic of chained functions or particular expressions, such as an incorrect implementation of incentive designs, vesting schemes, and similar.
Control Flow	Control Flow findings concern the access control imposed on functions, such as overpowered access functions being executable by anyone under certain circumstances or control flow hijackings like reentrancy attacks.
Volatile Code	Volatile Code findings refer to code implementations that behave unexpectedly on particular edge cases that may result in exploitability or sensitive and unreliable code behavior.
Overpowered Design	Owerpowered Design findings describe code that entails a certain amount of trust in centralized entities such as an owner not behaving maliciously and to maintain code integrity.
Language Specific	Language Specific findings are issues that are related to the Solidity programming language such as incorrect usage of the delete keyword or conformity with language limitations.
Coding Style	Coding Style findings are primarily informational and they help to increase the quality of the codebase and easier maintainable by following a consistent styling guide.
Code Correctness	Code Correctness findings refer to functions that should seemingly behave similarly yet contain different code, legacy inheritance graph versioning, explicit visibility markings, etc.

A. Classification List

TYPE	DESCRIPTION
Magical Numbers	Magic Number findings refer to numerical values that are defined in the codebase in their raw format and should otherwise be specified as contract variables to increase code legibility and maintainability.
Compiler Error	Compiler Error findings refer to an issue in the implementation of a segment of the code that renders it impossible to compile using the specified version of the codebase.
Dead Code	Code that otherwise does not affect the functionality of the codebase and can be safely omitted to avoid code bloat and to increase the overall quality of the codebase.

B. Source Code Fingerprints

FILE	FINGERPRINT (SHA-256 Checksum)
LIBE.sol	2491a5c06271c6d581bfb5fcd8350a46578d339a9cbe823ae69673513b86796e