CERTIK

CertiK Audit Report for NIAX

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 NIAX (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 **NIAX** to discover issues and vulnerabilities in the source code of their **NIAX ERC-20 Smart Contract** 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

TBD

Smart Contract Audit

This report has been prepared as a product of the Smart Contract Audit request by NIAX.

This audit was conducted to discover issues and vulnerabilities in the source code of the NIAX ERC-20 Smart Contract.

TYPE	Smart Contract
SOURCE CODE	https://etherscan.io/address/0xf71982762D141f8679Eb944fAec8cEC415fB5E23#code
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	July 24, 2020
DELIVERY DATE	July 26, 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 NIAX team to audit the design and implementation of their token smart contract and its compliance with the EIPs it is meant to implement.

The audited source code link is:

- Token Source Code: https://etherscan.io/address/0xf71982762D141f8679Eb944fAec8cEC415fB5E23#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.

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 contracts in scope were minimal although the token fulfilled a simple use case we were able to fully assimilate. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions.

These were considered the specification, and when discrepancies arose with the actual code behaviour, we consulted with the NIAX team or reported an issue.

Summary

The codebase of the project is a typical EIP20 implementation with additional support for a freezing mechanism and a time lock mechanism.

Certain optimization steps that we pinpointed in the source code mostly referred to coding standards and inefficiencies and no vulnerabilities or attack vectors were identified during our audit.

The codebase of the project strictly adheres to the standards and interfaces imposed by the OpenZeppelin open-source libraries and as such its typical ERC-20 functions **can be deemed to be of high security and quality, however the custom functionality built on top of it possessed flaws** we identified.

Recommendations

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	Language Specific	Informational	All “pragma” statements

[INFORMATIONAL] Description:

The smart contract “pragma” statements regarding the compiler version indicate that version 0.5.17 or higher should be utilized.

Recommendations:

We advise that the compiler version is locked at version 0.5.17 or whichever Solidity version higher than that satisfies the requirements of the codebase as an unlocked compiler version can lead to discrepancies between compilations of the same source code due to compiler bugs and differences.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Usage of Magic Numbers	Mathematical	Informational	NIAX: L614

[INFORMATIONAL] Description:

The specified line is meant to mint the total supply of the token to the “msg.sender”, equal to 800 million, as a hard-coded numeric literal.

Recommendations:

We advise that a “constant” contract variable is utilized instead that is more representative of its purpose.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	NIAX: L621, L631, L638

[INFORMATIONAL] Description:

The lines above conduct a greater-than “>” comparison between unsigned integers and the value literal “0”.

Recommendations:

As unsigned integers are restricted to the positive range, it is possible to convert this check to an inequality “!=” reducing the gas cost of the functions.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Redundant Duplicate Lookups	Optimization	Informational	NIAX: L621 - L625, L696 - L701, L710 - L717

[INFORMATIONAL] Description:

The aforementioned code-blocks conduct a lookup on the “timelockList” mapping multiple times for the same variable redundantly.

Recommendations:

It is possible to instead store the result of the lookup, in this case of type “LockInfo[] storage”, to an in-memory variable that is subsequently accessed to reduce the gas cost of the functions as they would not require to conduct the hashing operations involved in the lookup duplicate times.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Precondition to Modifier	Coding Style	Informational	NIAX: L631 - L633, L638 - L640

[INFORMATIONAL] Description:

The aforementioned “if” blocks ensure that whenever an outgoing transaction occurs from a specific account, the funds that have been time locked up to that point are released.

Recommendations:

We advise that the code block is instead converted to a “autoUnlock” modifier that accepts an “address” argument and subsequently checks and unlocks any time locked funds to increase the legibility of the codebase and limit code duplication.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Toggle to Setter Function	Optimization	Informational	NIAX: L644 - L656

[INFORMATIONAL] Description:

The functions “freezeAccount” and “unfreezeAccount” set and unset the frozen status of an account respectively.

Recommendations:

We advise that they are instead combined into a single setter function that accepts the value that should be assigned to “frozenAccount” as an argument to reduce the bytecode of the contract and code duplication.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Inexistent Error Messages	Coding Style	Informational	NIAX: L608, L645, L652, L683, L736

[INFORMATIONAL] Description:

The aforementioned “require” statements do not contain an accompanying error message.

Recommendations:

It is a generally accepted coding practice to include an error message whenever a condition is checked via a “require” invocation.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Array Swap Operation	Language Specific	Informational	NIAX: L699 - L701

[INFORMATIONAL] Description:

The aforementioned code segment firstly deletes the element to be swapped, overrides its position with the last element of the array and consequently reduces the length of the array by 1.

Recommendations:

Within Solidity, it costs less gas to affect an already initialized storage space and as such, the “delete” operation on L699 is inefficient as new data will be written to that storage space on the immediate next statement.

We advise the removal of the “delete” statement as well as the replacement of the statement in L701 with a “pop” invocation on the array as it will internally “delete” the last element and ensure that the “length” does not underflow under any circumstance.