CertiK Audit Report for Fetch.ai

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 Fetch.ai (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 **Fetch.ai** to discover issues and vulnerabilities in the source code of their **MetalX** smart contracts. 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 structures and implementations 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

Smart Contract Audit

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

This audit was conducted to discover issues and vulnerabilities in the source code of **Fetch.ai's MetalX** Smart Contracts.

TYPE	**Smart Contract**
SOURCE CODE	https://github.com/orgbitsapphire/metalx-token/tree/52a5a6281196a5750cab54245622d2e613e4727c
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	Aug 17, 2020
REVISION DATE	Sep 09, 2020
METHODS	A comprehensive examination has been performed using Dynamic Analysis, Static Analysis, and Manual Review.

Review Notes

Introduction

CertiK team was contracted by Fetch.ai to audit the design and implementation of their MetalX Smart Contracts for security vulnerabilities and compliance with Solidity language standards under different perspectives and with different tools such as static analysis and manual reviews by smart contract experts to find specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

The audited commit hash is:

`52a5a6281196a5750cab54245622d2e613e4727c`

The audited files and their associated sha256 hashes are:

1. interfaces/IERC20.sol

`78d45685e5f888c15af7f2123053c4705bd6f6d632787bf3883c3a9e577a25f2`

2. interfaces/IMetalX.sol

`cf1009985d61b44989d6e862710241dbd09c2abbd743e53767f8d0172284a59e`

3. interfaces/IStaking.sol

`074b6a62ba8f147458746cc82c975e61270a833e3d3a5ec4c8d45cc5143f61f1`

4. contracts/MetalX.sol

`42d4f7ff49093af6fdba84a99713f3089073623e5dac5bb26fb25d303b197796`

5. contracts/Staking.sol

`eee16539ed69e46585f5bb38bdd00d52e76f549bd81286ede4a826b742bb6480`

Summary

Fetch.ai has designed and implemented a staking smart contract for their MTLX ERC-20 token. The Staking contract allows any user to deposit an amount of FET tokens up until the "frozen window" defined by the owner of the Staking contract. During the frozen window, deposited FET tokens cannot be deposited or withdrawn and all deposited FET tokens begin to accumulate by the drip amount per-block as set by the Staking contract owner. Users can withdraw any amount of their deposited FET tokens before the frozen window has been reached without any MTLX token accumulation taking effect. Redeeming accumulated MTLX tokens during the frozen window resets the accumulation factor for a particular user. Users can redeem accumulated MTLX tokens at any time, but cannot withdraw their staked FET tokens until the frozen window has passed.

The team statically analyzed and manually reviewed the smart contracts of the provided MetalX and Staking implementations for security vulnerabilities and compliance with Solidity language preferred practices and standards. The codebase was found to be well-written and has a sound implementation, but ignored the Check Effects Interactions pattern and did not adhere to typical Solidity naming conventions. 2 minor and 7 informational findings were found. While these issues did not compromise the system, we recommended that they should be addressed in order to clearly convey and verify that the system functions as intended, which Fetch.ai addressed in the following commits:

1. `06678dd6489c2e51bee5a39cdd26afc74216c529`
2. `b3c6cf62a9dc21634436fd04a9a63d8c74827f11`
3. `e9e87c74e1cc9eb49f369637587fbb0c56d0658b`

Fetch.ai also made an additional commit to the MetalX repository that introduced a cap to the MetalX token contract, which was verified to not introduce any new issues. The commit introducing the token cap is `6dfe71d8128966f519d5ef558b19c7e61be02c6a`.

Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Unused return value	State Change	Minor	Staking.sol, L136-L138
Insufficient internal implementation	Implementation	Minor	Staking.sol, L125-140

[MINOR] Description:

The Staking._redeem function was used by other functions in the contract in a way that prevented it from safely considering the Solidity Check Effects Interactions pattern. An external call was also made in the internal Staking._redeem function to the issuingToken state variable's IERC20.transfer function without taking the result of the token transfer into account.

Recommendation:

We recommended refactoring the internal Staking._redeem function to return a bool and to take an explicit amount parameter, which can be safely calculated ahead of time by other functions in the contract. We also recommended that the call to the IERC20.transfer function should be required to succeed.

Alleviation:

The recommendations were applied in commit `b3c6cf62a9dc21634436fd04a9a63d8c74827f11`.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Potential for re-entrancy; Out-of-order events	Control Flow	Informational	Staking.sol, L94-L97

[INFORMATIONAL] Description:

The Staking.deposit function had the potential for re-entrancy due to ignoring the Solidity Check Effects Interactions pattern, which can lead to emitting events out of order and possibly cause issues for third parties.

Recommendation:

We recommended to follow the Solidity Check Effects Interactions pattern and that all events be emitted before making external calls.

Alleviation:

The recommendations were applied in commit `e9e87c74e1cc9eb49f369637587fbb0c56d0658b`.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Potential for re-entrancy; Out-of-order events	Control Flow	Informational	Staking.sol, L101-L117

[INFORMATIONAL] Description:

The Staking.withdrawAndRedeem function had the potential for re-entrancy due to ignoring the Solidity Check Effects Interactions pattern, which can lead to emitting events out of order and possibly cause issues for third parties.

Recommendation:

We recommended calculating the accumulated MTLX tokens before redeeming, applying changes to all state variables, emitting all events, calling the internal Staking._redeem function with the already-calculated accumulated amount and requiring it to succeed, then finally withdrawing the staked FET tokens.

Alleviation:

The recommendations were applied in commit `e9e87c74e1cc9eb49f369637587fbb0c56d0658b`.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Potential for re-entrancy; Out-of-order events	Control Flow	Informational	Staking.sol, L119-L123

[INFORMATIONAL] Description:

The Staking.redeem function had the potential for re-entrancy due to ignoring the Solidity Check Effects Interactions pattern, which can lead to emitting events out of order and possibly cause issues for third parties.

Recommendation:

We recommended calculating the accumulated MTLX tokens before redeeming, emitting all events, calling the internal Staking._redeem function with the already-calculated accumulated amount and requiring it to succeed.

Alleviation:

The recommendations were applied in commit `e9e87c74e1cc9eb49f369637587fbb0c56d0658b`.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Redundant interface definition	Implementation	Informational	IERC20.sol

[INFORMATIONAL] Description:

There is an unused IERC20 interface included in the codebase inside interfaces/IERC20.sol which is a direct copy of the previously-verified OpenZeppelin implementation of the IERC20 interface.

Recommendation:

Since the codebase already depends on OpenZeppelin contracts and uses its IERC20 interface definition in place of the redundant IERC20 interface definition, consider removing the interfaces/IERC20.sol file.

Alleviation:

No alleviation was made, which is acceptable as the interface is unused.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Function names not in mixed-case	Naming Conventions	Informational	IStaking.sol

[MINOR] Description:

The IStaking interface had functions which were not named using mixed-case.

Recommendation:

We recommended renaming the functions in the IStaking contract using mixed-case instead of snake-case.

Alleviation:

The recommendations were applied in commit `b3c6cf62a9dc21634436fd04a9a63d8c74827f11`.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Function names not in mixed-case	Naming Conventions	Informational	Staking.sol

[INFORMATIONAL] Description:

The Staking contract had functions which were not named using mixed-case.

Recommendation:

We recommended renaming the functions in the Staking contract using mixed-case instead of snake-case.

Alleviation:

The recommendations were applied in commit `b3c6cf62a9dc21634436fd04a9a63d8c74827f11`.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
State variable names not in mixed-case	Naming Conventions	Informational	Staking.sol

[INFORMATIONAL] Description:

The Staking contract had state variables which were not named using mixed-case.

Recommendation:

We recommended renaming the state variables in the Staking contract using mixed-case instead of snake-case.

Alleviation:

The recommendations were applied in commit `b3c6cf62a9dc21634436fd04a9a63d8c74827f11`.