CERTIK

Fetch.ai Fetch Token

Security Assessment

October 5th, 2020

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Fetch.ai Fetch Token
Description	The contract implements standard ERC-20 token with mintable and pausable functionalities.
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commit	46bfb327d694f65e75b78c52da80aeb02d4235c1

Audit Summary

Delivery Date	Oct. 05, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Oct. 02, 2020 - Oct. 05, 2020

Vulnerability Summary

Total Issues	5
Total Critical	0
Total Major	0
Total Minor	0
Total Informational	5

Executive Summary

The file `FetToken.sol` contains flattened `FetchToken` contract which makes use of `OpenZeppelin`'s contracts to implement standard `ERC20` interface with `Mintable` and `Pausable` functionalities. The `OpenZeppelin`'s contracts are well tested and officially recognized.
The compiler version of the contract can be locked to a specific `solidity` version.
The `require` calls in the functions of `FetchToken` contract can be converted to modifiers. The `virtual` modifier can be removed from all the functions of `FetchToken` contract and the functions with `public` visibilities can have their visibility changed to `external`.

Findings

ID	Title	Type	Severity
FET-01	Unlocked Compiler Version	Compiler Version	Informational
FET-02	Substitution of require call with Modifier	Coding Style	Informational
FET-03	Substitution of require calls with Modifier	Coding Style	Informational
FET-04	`virtual` modifier can be removed	Language Specific	Informational
FET-05	Function visibility can be `external`	Function Visibility	Informational

FET-01: Unlocked Compiler Version

Type	Severity	Location
Compiler Version	Informational	FetToken.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at f.e. the contract can be safely locked at `v0.6.2`.

```
pragma solidity 0.6.2;
```

Alleviation:

Alleviations were applied as advised and the compiler version was locked to `v0.6.2`.

FET-02: Substitution of require call with Modifier

Type	Severity	Location
Coding Style	Informational	FetToken.sol L1216

Description:

The aformentioned `require` call can be converted into a modifier to increase the legibility of the code.

Recommendation:

We recommend to convert the `require` call to a modifier.

```
modifier onlyMinter {
    require(
        hasRole(MINTER_ROLE, _msgSender()),
        "signer must have minter role to mint"
    );
    _;
}
```

The usage of modifier in the function would be as followed.

```
function mint(address to, uint256 amount) public virtual onlyMinter {...}
```

Alleviation:

Alleviations were applied as advised with introduction of `onlyMinter` modifier.

FET-03: Substitution of require calls with Modifier

Type	Severity	Location
Coding Style	Informational	FetToken.sol L1230, L1244

Description:

The `require` calls on the aforementioned lines can be converted into a modifier to aid readability and maintainability of the code by avoiding code duplication.

Recommendation:

We recommend to convert the `require` calls to a modifier.

```
modifier onlyPauser {
    require(
        hasRole(PAUSER_ROLE, _msgSender()),
        "signer must have pauser role to pause"
    );
    _;
}
```

The usage of modifier in the functions would be as followed.

```
function pause() public virtual onlyPauser {...}
function unpause() public virtual onlyPauser {...}
```

Alleviation:

Alleviations were applied as advised with the introduction of `onlyPauser` modifier.

FET-04: `virtual` modifier can be removed

Type	Severity	Location
Language Specific	Informational	FetToken.sol L1216, L1230, L1244, L1249

Description:

The functions on the aforementioned lines have `virtual` modifiers and yet they are the final implementations in the contract's inheritance chain as there are no functions overriding any of them. As these functions are not being overriden, the `virtual` modifiers can be safely removed from each of the functions.

Recommendation:

We recommend to remove `virtual` modifiers from the functions on aforementioned lines.

Alleviation:

Alleviations were applied as advised and the modifier `virtual` was removed from the aforementioned functions.

FET-05: Function visibility can be `external`

Type	Severity	Location
Function Visibility	Informational	FetToken.sol L1216, L1230, L1244

Description:

The functions which are never called internally from within the contract should have `external` visibility. The functions on the aforementioned lines have `public` visibility which can be safely changed to `external`.

Recommendation:

We recommend to change the functions' visibilities from `public` to `external`.

Alleviation:

Alleviations were applied as advised and the visibilities of the aforementioned functions were changed to `external`.