CERTIK

Cook Token

Security Assessment

February 10th, 2021

For :
Cook Finance

By :
Angelos Apostolidis @ CertiK
angelos.apostolidis@certik.org

Minzhi He @ CertiK
minzhi.he@certik.org

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	Cook Token
Description	A typical ERC20 implementation with enhanced features.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. 26f94b2ac1771f7d5d3347a744dc072cb37f8280 2. 3548841944cc17d2be976ecdb19c45424745f5aa

Audit Summary

Delivery Date	February 10th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	February 3rd, 2021 - February 10th, 2021

Vulnerability Summary

Total Issues	7
Total Critical	0
Total Major	0
Total Medium	0
Total Minor	0
Total Informational	7

Executive Summary

This report represents the results of CertiK's engagement with Cook Finance on their implementation of the Cook Token smart contract.

Our findings mainly refer to optimizations and Solidity coding standards, hence the issues identified pose no threat to the contract deployment's safety.

Files In Scope

ID	Contract	Location
CTN	CookToken.sol	contracts/token/CookToken.sol

Findings

ID	Title	Type	Severity	Resolved
CTN-01	Unlocked Compiler Version	Language Specific	Informational	✓
CTN-02	Order of Layout	Coding Style	Informational	✓
CTN-03	Function Optimization	Logical Issue	Informational	✓
CTN-04	Partial NatSpec Comments	Coding Style	Informational	✓
CTN-05	Unused Function Parameter	Gas Optimization	Informational	✓
CTN-06	`external` Over `public` Function	Gas Optimization	Informational	✓
CTN-07	Ambiguous Use of `virtual`	Volatile Code	Informational	✓

CTN-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	CookToken.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. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.6.2`.

CTN-02: Order of Layout

Type	Severity	Location
Coding Style	Informational	CookToken.sol L27-L32

Description:

The layout of the contract is out of order.

Recommendation:

We advise to closely follow the Solidity style guide.

Alleviation:

The development team opted to consider our references and changed the order of layout, closely following the Solidity conventions.

CTN-03: Function Optimization

Type	Severity	Location
Logical Issue	Informational	CookToken.sol L29

Description:

The `initialize()` function redundantly calls the `mint()` one, as the function call in L28 will ensure that the initializer will have minter privileges.

Recommendation:

We advise to change to a `_mint()` call, as the `require` statement in the `mint()` function will always be fulfilled. This will allow for the `mint()` function to be granted the `external` attribute.

Alleviation:

The development team opted to consider our references and changed the linked statement as proposed.

CTN-04: Partial NatSpec Comments

Type	Severity	Location
Coding Style	Informational	CookToken.sol L34-L39

Description:

The linked NatSpec comments omit the token creation information.

Recommendation:

We advise to update the linked NatSpec comments.

Alleviation:

The development team opted to consider our references and updated the NatSpec comment.

CTN-05: Unused Function Parameter

Type	Severity	Location
Gas Optimization	Informational	CookToken.sol L50

Description:

The `__ERC20PresetMinterPauser_init_unchained()` functionality does not utilize the function parameters `name` and `symbol`.

Recommendation:

We advise to remove the linked parameters.

Alleviation:

The development team opted to consider our references and removed the redundant code.

CTN-06: `external` Over `public` Function

Type	Severity	Location
Gas Optimization	Informational	CookToken.sol L27, L80, L94

Description:

The linked functions remain unused by the contract.

Recommendation:

We advise that the linked functions have their visilibity changed to `external` to save gas.

Alleviation:

The development team opted to consider our references and used the `external` attribute for the linked functions.

CTN-07: Ambiguous Use of `virtual`

Type	Severity	Location
Volatile Code	Informational	CookToken.sol L27, L66, L80, L94, L99

Description:

The linked functions ambiguously use the keyword `virtual`, as they are not expected to be overriden.

Recommendation:

We advise to remove the keyword `virtual` from the linked functions.

Alleviation:

The development team opted to consider our references and removed the redundant code.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.