CERTIK

Preliminary Comments

ExzoCoin Token

May 25th, 2021

Summary

This report has been prepared for ExzoCoin Token smart contracts, to discover issues and vulnerabilities in the source code of their 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.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	ExzoCoin Token
Description	A SafeMoon fork with additional functionality
Platform	BSC
Language	Solidity
Codebase	https://bscscan.com/address/0xa678d1785ce8ace00137f7200dd74288da082bea#code
Commits	a7723f8d042d6fc0d11f803cc76546f4852c013e

Audit Summary

Delivery Date	May 25, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	CoinToken

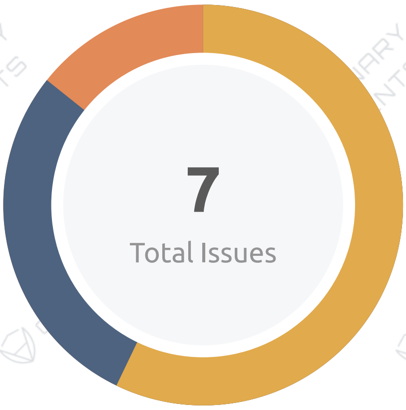
Vulnerability Summary

Total Issues	7
Critical	0
Major	1
Minor	4
Informational	2
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
ECE	ExzoCoin.sol	bbade0dcf99dbd3d0a3b0ed871dd858822997d185f64837c4aecb835e21de0c8

Findings



Critical	**0** (0.00%)
Major	**1** (14.29%)
Minor	**4** (57.14%)
Informational	**2** (28.57%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
ECE-01	Unlocked Compiler Version	Language Specific	Informational	Pending
ECE-02	User-Defined Getters	Gas Optimization	Informational	Pending
ECE-03	ERC-20 Incompatibility	Volatile Code	Major	Pending
ECE-04	Missing `event` Emission	Logical Issue	Minor	Pending
ECE-05	Potential Overflow	Mathematical Operations	Minor	Pending
ECE-06	Usage of `transfer()` for sending Ether	Volatile Code	Minor	Pending
ECE-07	Potential Over-centralization of Functionality	**Centralization / Privilege**	**Minor**	**Pending**

ECE-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	ExzoCoin.sol: 5	Pending

Description

The contract specifies an 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;
```

ECE-02 | User-Defined Getters

Category	Severity	Location	Status
Gas Optimization	Informational	ExzoCoin.sol: 398	Pending

Description

The linked variable contains a user-defined getter function that are equivalent to their name barring for an underscore (_) prefix / suffix.

Recommendation

We advise that the linked variable is renamed to its respective getter's name as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

ECE-03 | ERC-20 Incompatibility

Category	Severity	Location	Status
Volatile Code	Major	ExzoCoin.sol: 697	Pending

Description

The data type of the `_decimals` state variable should be `uint8` to conform to the EIP-20 standard, as every smart contract interacting with ERC-20 tokens will result in a fail.

Recommendation

We advise to change the data type of `_decimals` to `uint8`.

ECE-04 | Missing `event` Emission

Category	Severity	Location	Status
Logical Issue	Minor	ExzoCoin.sol: 756	Pending

Description

The constructor function of the `CoinToken` contract changes the `_owner` state variable, yet it omits the `OwnershipTransferred` event emission.

Recommendation

We advise to emit an `OwnershipTransferred` event from the zero address to the `tokenOwner`.

ECE-05 | Potential Overflow

Category	Severity	Location	Status
Mathematical Operations	Minor	ExzoCoin.sol: 896, 900	Pending

Description

Although the linked functions are only invocable by the owner of the contract, the linked statements can lead to an integer overflow.

Recommendation

We advise to utilize the `SafeMath` library for the linked arithmetic operations.

ECE-06 | Usage of `transfer()` for sending Ether

Category	Severity	Location	Status
Volatile Code	Minor	ExzoCoin.sol: 964	Pending

Description

After EIP-1884 was included in the Istanbul hard fork, it is not recommended to use `.transfer()` or `.send()` for transferring ether as these functions have a hard-coded value for gas costs making them obsolete as they are forwarding a fixed amount of gas, specifically `2300`. This can cause issues in case the linked statements are meant to be able to transfer funds to other contracts instead of EOAs.

Recommendation

We advise that the linked `.transfer()` and `.send()` calls are substituted with the utilization of the `sendValue()` function from the `Address.sol` implementation of OpenZeppelin either by directly importing the library or copying the linked code.

ECE-07 | Potential Over-centralization of Functionality

Category	Severity	Location	Status
Centralization / Privilege	Minor	ExzoCoin.sol: 963~965	Pending

Description

The linked function is meant to be used in an edge-case situation whereby the `owner` of the contract can claim the contract's remaining Ether.

Recommendation

We advise this functionality to be guarded by either a time delay to ensure that the normal course of operation of the contract has progressed.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings 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 findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings 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 but rather 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.

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 Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with 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.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is 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 assessment. This report does 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.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

CERTIK