Security Assessment

DOTC

May 18th, 2021

Summary

This report has been prepared for DOTC 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 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	DOTC
Description	ERC20 Token for Decentralized digital asset OTC trading platform
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/DOTCPro/Contracts
Commits	576a168519ccb7518204979294d68f292e698e5a

Audit Summary

Delivery Date	May 18, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

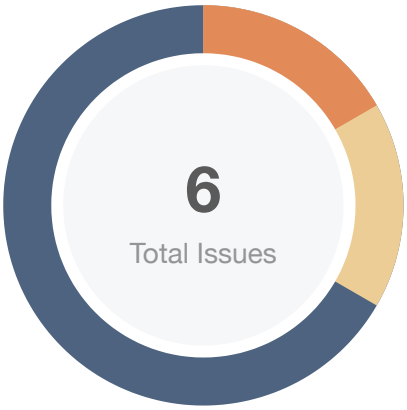
Vulnerability Summary

Total Issues	6
Critical	0
Major	1
Medium	0
Minor	1
Informational	4
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
DOT	token/DOTCToken.sol	9f0d98780e395e4c7f9a1c9e8ea0f4a4d225d49600b3dd204973823680fba992

Findings



Critical	0 (0.00%)
Major	1 (16.67%)
Medium	0 (0.00%)
Minor	1 (16.67%)
Informational	4 (66.67%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
DOT-01	Unlocked Compiler Version	Language Specific	Informational	Acknowledged
DOT-02	Set `constant` to Variables	Logical Issue	Informational	Resolved
DOT-03	Lack of Input Validation	Volatile Code	Minor	Resolved
DOT-04	Proper Usage of `require` and `assert` Functions	Coding Style	Informational	Acknowledged
DOT-05	Unused Return Value	Coding Style	Informational	Resolved
DOT-06	Incorrect ERC20 Interface	Logical Issue	Major	Resolved

DOT-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	token/DOTCToken.sol: 2	Acknowledged

Description

The contract has unlocked compiler versions. 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 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

It is a general practice to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

Alleviation

The client changed `^0.7.0` to `>=0.7.0` and the contract still has unlocked compiler versions in commit `8b0297a94cc4c7651ab7b87842fc10f61dabccb2`.

DOT-02 | Set `constant` to Variables

Category	Severity	Location	Status
Logical Issue	Informational	token/DOTCToken.sol: 11	Resolved

Description

The variable `decimals` is not changed throughout the smart contract.

Recommendation

We advise the client to set `decimals` as a constant variable.

Alleviation

The `decimals` variable is now declared as a `constant` and the issue is fixed in commit `8b0297a94cc4c7651ab7b87842fc10f61dabccb2`.

DOT-03 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	Minor	token/DOTCToken.sol: 38	Resolved

Description

The assigned values to `_to` should be verified as non-zero values to prevent being mistakenly assigned as `address(0)` in the `_transfer()` function.

Recommendation

Check that the addresses are not zero by adding the following checks in the `_transfer()` function.

```
require(_to != address(0),"Zero address");
```

Alleviation

The client informed us that transferring to zero address is for burning tokens and didn't add the check in commit `8b0297a94cc4c7651ab7b87842fc10f61dabccb2`.

DOT-04 | Proper Usage of `require` and `assert` Functions

Category	Severity	Location	Status
Coding Style	Informational	token/DOTCToken.sol: 47	Acknowledged

Description

The `assert` function should only be used to test for internal errors, and to check invariants. The `require` function should be used to ensure valid conditions, such as inputs, or contract state variables are met, or to validate return values from calls to external contracts.

Recommendation

Consider using the `require` function, along with a custom error message when the condition fails, instead of the `assert` function on the lines showcased above.

Alleviation

The client prefers `assert` over `require` on this and did not change the code in commit `8b0297a94cc4c7651ab7b87842fc10f61dabccb2`.

DOT-05 | Unused Return Value

Category	Severity	Location	Status
Coding Style	Informational	token/DOTCToken.sol: 85	Resolved

Description

The return value `success` is declared but never used in the function body.

Recommendation

Remove or comment out the return value.

Alleviation

The client removed `success` and fixed the issue in commit `8b0297a94cc4c7651ab7b87842fc10f61dabccb2`.

DOT-06 | Incorrect ERC20 Interface

Category	Severity	Location	Status
Logical Issue	Major	token/DOTCToken.sol: 58	Resolved

Description

Incorrect return values for ERC20 function `transfer()`. A contract compiled with Solidity > 0.4.22 interacting with these functions will fail to execute them, as the return value is missing.

Recommendation

Set the appropriate return values and types for the defined ERC20 function `transfer()`.

Alleviation

The `transfer()` function now returns a boolean in accordance with the ERC-20 and the issue is fixed in commit `8b0297a94cc4c7651ab7b87842fc10f61dabccb2`.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

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.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

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.