CERTIK

Security Assessment

DinoX

Jun 18th, 2021

Table of Contents

Summary

This report has been prepared for DinoX 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.

No notable vulnerabilities were identified in the codebase and it makes use of the latest security principles and style guidelines. There were certain optimizations observed as well as security principles that can optionally be applied to the codebase to fortify the codebase to a greater extent.

Overview

Project Summary

Project Name	DinoX
Description	A typical ERC-20 implementation plus an ERC-721 with additional features.
Platform	Ethereum
Language	Solidity
Codebase	DinoX
Commit	db77db27e929de55f3b8e0062a0377fbb6949fed

Audit Summary

Delivery Date	Jun 18, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	ERC-20, ERC-721

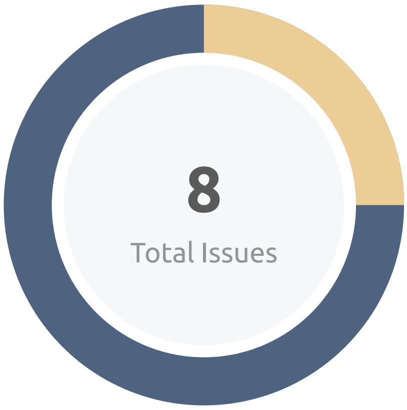
Vulnerability Summary

Total Issues	8
Critical	0
Major	0
Medium	0
Minor	2
Informational	6
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
DDN	DNX/Dinox.sol	455059c60f1549924105b71fc3db85ab391dfb6a8478f009205ba118304c7b05
DXC	DNXC/DinoXCoin.sol	d9ec3611c9f5f286f4f61fcf157399f76257dc70ac3c50bd8b90ce7666f8ce64

Findings



Critical	**0** (0.00%)
Major	**0** (0.00%)
Medium	**0** (0.00%)
Minor	**2** (25.00%)
Informational	**6** (75.00%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
DDN-01	Unlocked Compiler Version	Language Specific	Informational	Acknowledged
DDN-02	Redundant Variable Initialization	Coding Style	Informational	Acknowledged
DDN-03	Ambiguous Use of `virtual`	Language Specific	Informational	Acknowledged
DDN-04	Usage of `send()` for sending Ether	Volatile Code	Minor	Acknowledged
DDN-05	Conditional Optimization	Gas Optimization	Informational	Acknowledged
DDN-06	Ambiguous Use of `payable`	Language Specific	Minor	Acknowledged
DDN-07	Inefficient `storage` Read	Gas Optimization	Informational	Acknowledged
DXC-01	Unlocked Compiler Version	Language Specific	Informational	Acknowledged

DDN-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	DNX/Dinox.sol: 2	Acknowledged

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;
```

Alleviation

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

DDN-02 | Redundant Variable Initialization

Category	Severity	Location	Status
Coding Style	Informational	DNX/Dinox.sol: 27~28, 80~81, 90~91	Acknowledged

Description

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint` / `int`: All `uint` and `int` variable types are initialized at `0`
- address`: All `address`types are initialized to`address(0)`
- `byte`: All `byte` types are initialized to their `byte(0)` representation
- `bool`: All `bool` types are initialized to `false`
- `ContractType`: All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)
- `struct`: All `struct` types are initialized with all their members zeroed out according to this table

Recommendation

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Alleviation

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

DDN-03 | Ambiguous Use of `virtual`

Category	Severity	Location	Status
Language Specific	Informational	DNX/Dinox.sol: 218	Acknowledged

Description

The linked functions are not expected to be overridden, hence rendering the use of the keyword `virtual` redundant.

Recommendation

We advise to remove redundant code.

Alleviation

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

DDN-04 | Usage of `send()` for sending Ether

Category	Severity	Location	Status
Volatile Code	Minor	DNX/Dinox.sol: 185	Acknowledged

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.

Alleviation

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

DDN-05 | Conditional Optimization

Category	Severity	Location	Status
Gas Optimization	● Informational	DNX/Dinox.sol: 190	ⓘ Acknowledged

Description

The linked code segment can be omitted with the use of a return variable utilization.

Recommendation

We advise to utilize a return variable and invert the linked conditional.

Alleviation

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

DDN-06 | Ambiguous Use of `payable`

Category	Severity	Location	Status
Language Specific	Minor	DNX/Dinox.sol: 184	Acknowledged

Description

The `rawAll()

Recommendation

We advise to remove the `payable` keyword from the `withdrawAll()` function.

Alleviation

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

DDN-07 | Inefficient `storage` Read

Category	Severity	Location	Status
Gas Optimization	● Informational	DNX/Dinox.sol: 35, 39, 78, 94, 50, 51, 52, 55, 107	ⓘ Acknowledged

Description

Inefficient storage reads represent the redundant storage reads where gas can be saved by storing storage variables in local variable.

Recommendation

We advise to introduce a local variable instead.

Alleviation

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

DXC-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	DNXC/DinoXCoin.sol: 2	Acknowledged

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;
```

Alleviation

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

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.

Volatile Code

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

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.

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.

CERTIK