CERTIK

CertiK Audit Report for Orion

Contents

Contents 1

Disclaimer 1

About CertiK 2

Executive Summary 3

Testing Summary 4

Review Notes 5

- Introduction 5
- Documentation 5
- Summary 6
- Recommendations 6

Findings 7

Exhibit 1 7

Exhibit 2 8

Exhibit 3 9

Exhibit 4 10

Exhibit 5 11

Exhibit 6 12

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 Verification Services Agreement between CertiK and Orion (the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, 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.

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, CertiK's mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that projects are checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and assessments to each project, in turn creating a more secure and robust software system.

CertiK has served more than 100 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance's BGBP and Paxos Gold to decentralized oracles

such as Band Protocol and Tellor. CertiK customizes its engineering tool kits, while applying cutting-edge research on smart contracts, for each client on its project to offer a high quality deliverable. For more information: https://certik.io.

Executive Summary

This report has been prepared for **Orion** to discover issues and vulnerabilities in the source code of their **Orion Token and the token's crowdsale** 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.

Testing Summary

SECURITY LEVEL

TBA

Smart Contract Audit
This report has been prepared as a product of the Smart Contract Audit request by Orion.
This audit was conducted to discover issues and vulnerabilities in the source code of Orion's ORN ERC token and crowdsale mechanism.

TYPE	Smart Contract & Crowdsale Mechanism
SOURCE CODE	https://github.com/orionprotocol/sale-contract/tree/ebf0afdd46abe2b22651c7cd000a5716b0f94839
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	July 10, 2020
DELIVERY DATE	July 11, 2020
METHODS	A comprehensive examination has been performed using Dynamic Analysis, Static Analysis, and Manual Review.

Review Notes

Introduction

CertiK team was contracted by the Orion team to audit the design and implementation of their ORN ERC token smart contract as well as the crowdsale mechanism of the said token.

The audited source code link is:

- ORN & Crowdsale Source Code: https://github.com/orionprotocol/sale-contract/tree/ebf0afdd46abe2b22651c7cd000a5716b0f94839

The goal of this audit was to review the Solidity implementation for its business model, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

The findings of the initial audit have been conveyed to the team behind the contract implementations and the source code is expected to be re-evaluated before another round of auditing has been carried out.

The Orion team evaluated our exhibits and chose to apply them in an updated commit of the codebase that can be followed below:

- ORN & Crowdsale Source Code: https://github.com/orionprotocol/sale-contract/tree/53725ed347c76a60d8c106d2856d40f50cc64c5a

Documentation

The sources of truth regarding the operation of the contracts in scope were lackluster and **are something we advise to be enriched to aid in the legibility of the codebase as well as project**. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions.

These were considered the specification, and when discrepancies arose with the actual code behaviour, we consulted with the Orion team or reported an issue.

Summary

The codebase of the project is relatively straightforward and the crowdsale of the token follows the conventional model of an Ethereum-based crowdsale using an ETH to ORN ratio.

A **minor vulnerability** was identified in the way the ETH to ORN ratio calculation is carried out as well as **certain optimization steps** that we pinpointed in the source code mostly referred to coding standards and inefficiencies.

Recommendations

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report, enforce linters and / or coding styles as well as correct any spelling errors and mistakes that appear throughout the code **to achieve a high standard of code quality and security**.

Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Implementation Inconsistency	Coding Style	Informational	USDC.sol & OrionToken.sol

[INFORMATIONAL] Description:

The coding style of the USDC contract implementation contains hard-coded values for the initialization of the ERC20Detailed OpenZepplin library, however the Orion Token contract implementation accepts these values as constructor arguments.

Recommendations:

We advise that either the hard-coded style of USDC or the argument-based style of OrionToken is followed by both contracts to ensure consistency in the codebase, the former of which we would advise to ensure that the comments between L6 and L20 of OrionToken.sol hold true regardless of how the contract is deployed. Additionally, this style would prevent L30 from potentially overflowing if the constructor arguments are malformed i.e. “cap” is set to a very high value.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Misleading Comments	Code Legibility	Informational	OrionSale.sol: L21, L24

[INFORMATIONAL] Description:

The aforementioned lines concern comments made regarding the variable declarations that follow them. These comments do not accurately reflect the functionality of the variables, f.e. L24 states that the variable “rate” represents the USDC to token being sold ratio, however L90 utilizes it as an ETH to token being sold ratio.

Recommendations:

We advise that these comments are adjusted to properly reflect the functionality and / or purpose of the variables that follow them.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	OrionSale.sol: L139, L155, L156, L172

[INFORMATIONAL] Description:

The lines above conduct a greater-than “>” comparison between unsigned integers and the value literal “0”.

Recommendations:

As unsigned integers are restricted to the positive range, it is possible to convert this check to an inequality “!=” reducing the gas cost of the function.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison	Optimization	Informational	OrionSale.sol: L58

[INFORMATIONAL] Description:

The lines above conduct a greater-than “>” comparison between the variables “tokensSold” and “cap” whereas the function “_getTokenAmount” guarantees that the tokens sold will never exceed the cap.

Recommendations:

We advise this comparison to be converted to an equality comparison to reduce the gas cost necessary for executing the function.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Loss of Ether Precision	Ineffectual Code	Minor	OrionSale.sol: L94 - L101

[MINOR] Description:

The “if-else” statements contained in the aforementioned lines ensure that the tokens that have been calculated for the sale do not exceed the cap of the crowdsale. However, the returned “realAmount” does not accurately represent the number of tokens being rewarded for the sale as the statement of L90 truncates the values.

Recommendations:

We advise that the code block is refactored to instead only assign either “tokensToBuy” or “remaining” to the variable “allowed” as is the current state and an additional statement is created outside that assigns the result “allowed” multiplied by “1 ether” and divided by the “rate” to properly reflect the amount of funds necessary for the sale.

Additionally, we would advise that the surrounding codebase is refactored so that the function “_getTokenAmount” returns a remainder instead that, if not equal to zero, is transmitted to the original buyer of the sale.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Usage of “transfer” to Wallet	Language Specific	Informational	OrionSale.sol: L182, L186

[INFORMATIONAL] Description:

The “transfer” statement of Solidity forwards a gas stipend of 2300 which cannot be altered.

Recommendations:

Should the recipient of the funds be a wallet implementation, it may conduct additional statements in its fallback function and thus be unable to receive funds transmitted by this contract. Utmost care should be taken when considering what type of address the variable “wallet” will hold.