CERTIK

CertiK Audit Report for Decentr

Contents

Contents **1**

Disclaimer **2**

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**

Exhibit 7 **13**

Exhibit 8 **14**

Exhibit 9 **15**

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 decentr (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 **decentr** to discover issues and vulnerabilities in the source code of their **decentrSaleContract ERC-20 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.

Testing Summary

SECURITY LEVEL

Smart Contract Audit

This report has been prepared as a product of the Smart Contract Audit request by decentr.

This audit was conducted to discover issues and vulnerabilities in the source code of the decentrSaleContract ERC-20 Smart Contract.

TYPE	Smart Contract
SOURCE CODE	https://etherscan.io/address/0xb76b2064b11a5fe03f60e77ce13ba8542fdd4b98#code
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	July 02, 2020
DELIVERY DATE	Aug 18, 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 decentr team to audit the design and implementation of their token smart contract and its compliance with the EIPs it is meant to implement.

The audited source code link is:

- https://etherscan.io/address/0xb76b2064b11a5fe03f60e77ce13ba8542fdd4b98#code

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.

Documentation

The sources of truth regarding the operation of the contracts in scope were minimal although the token fulfilled a simple use case we were able to fully assimilate. To help aid our understanding of each contract's functionality we referred to the thorough in-line comments and naming conventions.

Summary

The codebase of the project is a typical EIP20 implementation with additional support for pausing and locking mechanisms.

Although **certain optimization steps** that we pinpointed in the source code mostly referred to coding standards and inefficiencies, **any minor (or above) flaws** that were identified, **should be remediated as soon as possible to ensure the security of the contracts** of Decentr's team.

The codebase of the project strictly adheres to the standards and interfaces imposed by the OpenZeppelin open-source libraries and as such its typical ERC-20 functions **can be deemed to be of high security and quality, however the custom functionality built on top of it possessed flaws** we identified.

Decentr's team considered our references and provided feedback based on our findings, but opted not to change their current codebase. **We approve of this stance**, as Decentr's team showed awareness and maturity towards the edge cases of their codebase.

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
Unlocked Compiler Version Declaration	Language Specific Issue	Informational	Lines 10, 37, 117, 279, 423, 732, 802, 892

[INFORMATIONAL] Description:

The compiler version utilized throughout the project uses the “^” prefix specifier, denoting that a compiler at or above the version included after the specifier should be used to compile the contracts. Also, the compiler version should be consistent throughout the codebase.

Recommendations:

It is a general practise 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.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Unsigned Integer Handling	Optimization	Informational	Lines 234, 967, 1203

[INFORMATIONAL] Description:

When dealing with unsigned integers, the condition should better check for the edge case, i.e. equality/inequality with zero, as the value will always be greater than or equal to zero.

Recommendations:

We advise the team to change the condition of the “require” statements to check for zero inequality.

Example:

require(x != 0, “error message”)

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
State Variable that could be declared Constant	Optimization	Informational	Line 914

[INFORMATIONAL] Description:

The variable representing the maximum amount of mintable tokens, i.e. "_maxMintable", could be changed to a constant.

Recommendations:

We advise the team to consider changing the state variable "_maxMintable" to constant.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Unused State Variable	Optimization	Informational	Line 931

[INFORMATIONAL] Description:

The state variable “tokenInitialized" is never used in the contracts.

Recommendations:

We advise the team to consider removing the state variable "tokenInitialized".

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Potential Overflow/Underflow	Volatile Code	Minor	Lines 979, 1058, 1206, 1209, 1215

[MINOR] Description:

In general, direct mathematical operations should be avoided in fear of a potential overflow/underflow.

Recommendations:

We advise the team to consider using the SafeMath library to narrow down the possibility of overflow/underflow.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Require Function Error Messages	Coding Style	Informational	Lines 1032, 1203

[INFORMATIONAL] Description:

The “require” statements provide no “reason” string for when they fail. A corresponding error code should be provided, as done with all other “require” statements, to ease the debugging of smart contracts.

Recommendations:

We advise that the team adds error messages for all "require" statements.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Unused/Ignored Return Value	Optimization	Informational	Lines 1035, 1221

[INFORMATIONAL] Description:

The returned boolean value of the function “mintToken()” is ignored when called by the functions “_releaseTimeLockedTokensForAddress” and “_buyToken”.

Recommendations:

We advise the team to consider checking if the value of the returned variable after the function call.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Potentially Incorrect Logic Implementation	Procedure Logic	Major	Lines 341, 381

[MAJOR] Description:

The variable "_totalMinted" is being incremented at an exponential rate, meaning that locked up tokens will not be redeemable past their expiration period due to the fact that the total supply of the contract will never be minted.

Recommendations:

We advise the team to revise the relevant code and also to consider changing to "_totalMinted += amount".

Exhibit 9

TITLE	TYPE	SEVERITY	LOCATION
Special Variable Update	Language Specific Issue	Informational	Lines 968, 977, 1006, 1032, 1070

[INFORMATIONAL] Description:

The "now" keyword, as newer versions of Solidity (namely 0.7.0 onwards) deprecated the "now" keyword.

Recommendations:

We advise the team to consider changing "now" with "block.timestamp".