CERTIK

CertiK Audit Report for MOSS

Contents

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 MOSS (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 **MOSS** to discover issues and vulnerabilities in the source code of their **ERC-20 Smart Contracts** 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 MOSS.

This audit was conducted to discover issues and vulnerabilities in the source code of MOSS' ERC-20 Smart Contracts.

TYPE	Smart Contract
SOURCE CODE	https://bitbucket.org/onepercent_devs/oken-contracts/src/master/
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	Aug 10, 2020
DELIVERY DATE	Sept 1, 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 MOSS team to audit the design and implementation of their ERC-20 token smart contracts and its compliance with the EIPs it is meant to implement.

The audited source code link is:

- https://bitbucket.org/onepercent_devs/oken-contracts/src/b935a2c42ceb450769406e9e86abf4980b910435/

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 in-line comments and naming conventions.

Summary

The codebase of the project is a typical EIP20 implementation with additional support for a burning mechanism.

Certain optimization steps that we pinpointed in the source code mostly referred to coding standards and inefficiencies, although **two minor flaws** were identified that should be remediated as soon as possible to ensure the contracts of the MOSS team are of the highest standard and quality.

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**.

Recommendations

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report **to achieve a high standard of code quality and security**.

Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Redundant `event` Variables	Optimization	Informational	TokenFactory: L12-L13

[INFORMATIONAL] Description:

The `blockNumber` variables that are passed to the events represent the `block.number` at the time the transaction is executed. As this information already exists as metadata of the emitted event, it can be omitted and instead retrieved from there.

Recommendations:

We advise the removal of redundant code.

Alleviations:

No alleviations.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Asynchronous Arrays	Coding Style & Optimization	Informational	TokenFactory: L9-L10

[INFORMATIONAL] Description:

These arrays may lose sync if transactions are emitted in a disorderly fashion which is expected as multiple token deployments and access list deployments are meant to occur throughout the lifetime of the contract and at separate times.

Recommendations:

We advise that the concept described should be revamped, and a synchronous addition should be introduced.

Alleviations:

No alleviations.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Enable / Disable Function to Setter	Optimization	Informational	Authorizable: L17-L25

[INFORMATIONAL] Description:

The two linked functions can instead be set to a single “Setter” function whereby the true / false variable is accepted as an argument.

Recommendations:

We advise the implementation of a single “Setter” function, as described above.

Alleviations:

No alleviations.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Duplicate Getter	Optimization	Informational	Authorizable: L7, L27-L29

[INFORMATIONAL] Description:

The variable `authorized` is defined as `public` yet a user defined “Getter” function exists called `isAuthorized`.

Recommendations:

We advise that one of the two getters is omitted to avoid duplicate bytecode, the latter of which we advise is avoided as compiler-generated getters are less prone to error than user-defined ones.

Alleviations:

The team opted to define the variable `authorized` as `private` and keep the user-defined “Getter” function.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Redundant User-Defined Getter	Optimization	Informational	TokenAccessList: L7, L71-L76

[INFORMATIONAL] Description:

As the user-defined “Getter” possesses the same name as the underlying variable that is retrieved barring for the underscores, it is possible to instead rename the variable to the Getter’s name and set its visibility to `public`, rendering the linked B (L71-L76) Getter redundant.

Recommendations:

We advise the removal of redundant code.

Alleviations:

The team opted to take our recommendation into account and removed the redundant "getter" function for a `public` variable.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Enable / Disable Function to Setter	Optimization	Informational	TokenAccessList: L17-L48

[INFORMATIONAL] Description:

The four linked functions can instead be set to two “Setter” functions whereby the true / false variable is accepted as an argument.

Recommendations:

We advise the implementation of the two “Setter” functions, as described above.

Alleviations:

No alleviations.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Function Overloading	Optimization	Informational	TokenAccessList: L50-L69

[INFORMATIONAL] Description:

As each function is meant to check multiple addresses and their status, it is wiser to create instead a single-address check as well as an array-based address check whereby if one address evaluates false, the loop is terminated early, and the value is returned.

Recommendations:

We advise the implementation of the concept described above in favor of the existing one.

Alleviations:

No alleviations.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Non-Standard Naming Convention	Coding Style	Informational	BurnerRole: L11

[INFORMATIONAL] Description:

Contract-level declarations should always begin with a lowercase letter preceded at most by a single underscore unless they represent `constant` variables as defined by the Solidity style guide.

Recommendations:

We advise the linked variable declaration(s) to be adjusted to reflect that.

Alleviations:

The team opted to take our recommendation into account and adhered to the standard Solidity style patterns.

Exhibit 9

TITLE	TYPE	SEVERITY	LOCATION
Invalid Event Emittence	Volatile Code	Minor	BurnerRole: L30-L32

[MINOR] Description:

As the `renounceBurner` function is unguarded, it is possible for an address that was never a burner to be emitted as “BurnerRemoved” thus potentially messing with external systems.

Recommendations:

The `onlyBurner` modifier may be introduced here to alleviate this issue.

Alleviations:

The team opted to take our recommendation into account, implemented the `onlyBurner` modifier and then introduced it to the `renounceBurner` function.

Exhibit 10

TITLE	TYPE	SEVERITY	LOCATION
Contract Implementation	Coding Style	Informational	CreatorRole.sol

[INFORMATIONAL] Description:

No findings were found at this stage of the audit. However, we find that BurnerRole.sol differs in implementation with this “Role” suffixed contract and as such whether the difference should exist should be evaluated.

Recommendations:

We advise the team to revise the contract implementation.

Alleviations:

The team opted to take our recommendation into account and restructured all “Role” suffixed contracts to follow the same coding pattern.

Exhibit 11

TITLE	TYPE	SEVERITY	LOCATION
Non-Standard Naming Convention	Coding Style	Informational	OperatorRole: L11

[INFORMATIONAL] Description:

Contract-level declarations should always begin with a lowercase letter preceded at most by a single underscore unless they represent `constant` variables as defined by the Solidity style guide.

Recommendations:

We advise the linked variable declaration(s) to be adjusted to reflect that.

Alleviations:

The team opted to take our recommendation into account and adhered to the standard Solidity style patterns.

Exhibit 12

TITLE	TYPE	SEVERITY	LOCATION
Invalid Event Emittence	Volatile Code	Minor	OperatorRole: L30-L32

[MINOR] Description:

As the `renounceOperator` function is unguarded, it is possible for an address that was never an operator to be emitted as “OperatorRemoved” thus potentially messing with external systems.

Recommendations:

The `onlyOperator` modifier may be introduced here to alleviate this issue.

Alleviations:

The team opted to take our recommendation into account, implemented the `onlyOperator` modifier and then introduced it to the `renounceOperator` function.

Exhibit 13

TITLE	TYPE	SEVERITY	LOCATION
Misleading Function Name	Coding Style	Informational	ControlledToken: L43-L96

[INFORMATIONAL] Description:

The function name alludes that a certain type of role being revoked yet internally the function also assigns that type of role to the caller of the function, potentially redundantly if called multiple times.

Recommendations:

We advise the team to revise the function implementation and change its name to a more descriptive one.

Alleviations:

No alleviations.

Exhibit 14

TITLE	TYPE	SEVERITY	LOCATION
Potentially Incorrect `modifier`	Coding Style	Informational	ControlledToken: L108-L115

[INFORMATIONAL] Description:

Even though multiple creators may be set, only the owner of the contract is able to renounce their creator status as the `renounceCreator` function is guarded by the `onlyOwner` modifier.

Recommendations:

We advise the team to revise the functionality implementation.

Alleviations:

No alleviations.

Exhibit 15

TITLE	TYPE	SEVERITY	LOCATION
`if` and `revert` to `require`	Coding Style & Optimization	Informational	ERC20AccessList: L40

[INFORMATIONAL] Description:

The linked `if` clause containing a `revert` statement should be changed to a single `require` statement for legibility purposes.

Recommendations:

We advise the implementation of the described concept.

Alleviations:

The team opted to take our recommendations into account, removed the linked `if` clause containing a `revert` statement and introduced a single `require` statement.

Exhibit 16

TITLE	TYPE	SEVERITY	LOCATION
Checks to `modifier`	Coding Style & Optimization	Informational	ERC20AccessList: L58-L173

[INFORMATIONAL] Description:

The `if` conditionals checking the access list on each linked function could instead be replaced with a modifier that accepts a set of arguments.

Recommendations:

We advise the implementation of a modifier, as described above.

Alleviations:

The team opted to take our recommendations into account and implemented the `hasAccess` modifier.

Exhibit 17

TITLE	TYPE	SEVERITY	LOCATION
Contract Implementation	Coding Style	Informational	ERC20Operator.sol

[INFORMATIONAL] Description:

No findings were identified at this stage of the audit. We should note that an operator has full control over the funds that exist all over the network, thus bringing a high centralization aspect to the cryptocurrency.

Recommendations:

No recommendation.

Alleviations:

No alleviations.

Exhibit 18

TITLE	TYPE	SEVERITY	LOCATION
Redundant User-Defined Getter	Optimization	Informational	ERC20CapEnabler: L12 & L19-L27, L13 & L29-L37

[INFORMATIONAL] Description:

As the user-defined “Getters” possess the same name as the underlying variables that are retrieved barring for the underscores, it is possible to instead rename the variables to the Getters’ name and set their visibility to `public`, rendering the linked B (L19-L27) and D (L29-L37) Getters redundant.

Recommendations:

We advise the team to remove redundant code.

Alleviations:

The team opted to take our recommendations into account, set the visibility to `public` for the linked variables and removed the user-defined “Getter” functions.

Exhibit 19

TITLE	TYPE	SEVERITY	LOCATION
Contract Implementation	Coding Style	Informational	ERC20BurnableAdmin.sol

[INFORMATIONAL] Description:

No findings were identified at this stage of the audit. We should note that a burner has delete control over the funds that exist all over the network, thus bringing a high centralization aspect to the cryptocurrency and enabling burners to easily manipulate the market.

Recommendations:

No recommendation.

Alleviations:

No alleviations.