CERTIK

CertiK Audit Report for Alliance Block

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 Alliance Block (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 **Alliance Block** to discover issues and vulnerabilities in the source code of their **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 Alliance Block.
This audit was conducted to discover issues and vulnerabilities in the source code of Alliance Smart Contracts.

TYPE	Smart Contract
SOURCE CODE	https://github.com/Stichting-Alliance Block-Foundation/AllianceBlock-Contracts/
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	Sept 01, 2020
DELIVERY DATE	Sept 09, 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 Alliance Block team to audit the design and implementation of their smart contracts and its compliance with the EIPs it is meant to implement.

The audited source code link is:

- https://github.com/Stichting-AllianceBlock-Foundation/AllianceBlock-Contracts/tree/1c675368322bf7e96b4801131188f0a86e58888d

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.

Documentation

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

Summary

The project's codebase is a typical EIP20 token implementation, along with batch token transfer and vesting mechanisms.

Although **certain optimization steps** that we pinpointed in the source code mostly referred to coding standards and inefficiencies, **the minor flaw** that was identified **should be remediated as soon as possible to ensure the security of the contracts.**

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 secure**.

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
Unlocked Compiler Version	Language Specific Issue	Informational	AllianceBlockToken: L2 BatchTransfer: L2 PercentageCalculator: L2 Vesting: L2

[INFORMATIONAL] Description:

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.

Recommendations:

We advise that the compiler version is instead locked at the lowest version possible that the full project can be compiled at.

Alleviations:

The team opted to take our recommendation into account and locked the compiler to version `0.6.12`.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Unused Imported Contract	Optimization	Informational	BatchTransfer: L5

[INFORMATIONAL] Description:

The parameter `deployedTokenAddress` is not of type `AllianceBlock Token` (as it is casted to `IERC20` on L11), meaning that the import is unnecessary.

Recommendations:

We advise the team to remove unnecessary code.

Alleviations:

The team opted to take our recommendation into account and removed the `import` statement.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Unused `return` Value	Optimization	Informational	BatchTransfer: L14

[INFORMATIONAL] Description:

The function `sendTokens` ignores the value returned by the `transferFrom` function.

Recommendations:

We advise the team to check the returned `boolean` value, as it indicated whether the transfer was fulfilled.

Alleviations:

The team opted to take our recommendation into account and added a `require` statement with the returned `boolean` value of the `transferFrom` function.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Missing Variable Visibility	Optimization	Informational	Vesting: L10, L13

[INFORMATIONAL] Description:

The linked variables are missing the visibility specifier.

Recommendations:

We advise the team to add the visibility specifier for each of the linked variables.

Alleviations:

The team opted to take our recommendation into account and added the visibility specifiers.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Undocumented Magic Number	Optimization	Informational	Vesting: L101

[INFORMATIONAL] Description:

In the linked `require` statement, the conditional compares the length of the `recipients` array with a magic number (literals with no documentation).

Recommendations:

We advise the team to add documentation on the purpose of the magic number.

Alleviations:

The team opted to take our recommendation into account and added in-line comments describing the purpose of the magic number.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Unused `return` Value	Optimization	Informational	Vesting: L122

[INFORMATIONAL] Description:

The returned value of the `transfer` function remains unused in this code block.

Recommendations:

We advise the team to check this returned value, as it indicates whether the transfer was realised or not.

Alleviations:

The team opted to take our recommendation into account and added a `require` statement with the returned `boolean` value of the `transfer` function.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Unnecessary Named Variables	Optimization	Informational	Vesting: L130, L142

[INFORMATIONAL] Description:

The use of named variables in the linked functions is redundant, as the variables are never used by another function the way they are meant to be.

Recommendations:

We advise the team to remove the named variables from the linked functions.

Alleviations:

No alleviations.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Unused Local Variable	Optimization	Informational	Vesting: L135

[INFORMATIONAL] Description:

The local variable `calculatedAmount` remains unused after the function `calculateAmounts` assigns a value to it.

Recommendations:

We advise the team to use `_` instead of an actual variable, as the function `hasClaim` should not support functionality for the calculated amount.

Alleviations:

The team opted to take our recommendation into account and changed the codebase according to our references.

Exhibit 9

TITLE	TYPE	SEVERITY	LOCATION
Change To A Constant Variable	Optimization	Informational	Vesting: L10

[INFORMATIONAL] Description:

The contract-level variable `periodLength` should be declared as constant, as it indicates the periods of the vesting schedule.

Recommendations:

We advise the team to declare `periodLength` as a constant variable.

Alleviations:

The team opted to take our recommendation into account and changed the codebase according to our references.

Exhibit 10

TITLE	TYPE	SEVERITY	LOCATION
Add Existing Recipient as New	Volatile Code	Minor	Vesting: L70-L89

[MINOR] Description:

If you try to add an already existing recipient as a new recipient with the `addRecipient()` function, then the existing `withdrawnAmount` and `_withdrawPercentage` variables for that recipient will forever be lost.

Recommendations:

We advise the team to add a `require` statement to check whether the `recipients` mapping already has a `key-value` pair for the specific address.

Alleviations:

No alleviations.