CERTIK

Audit Report

Produced by CertiK

for Animoca Brands

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 Animoca Brands (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 **Animoca Brands** to discover issues and vulnerabilities in the source code of their **REVV token & ERC-20 Base Dependency 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 Animoca Brands.
This audit was conducted to discover issues and vulnerabilities in the source code of Animoca Brands's REVV Token & ERC Base Dependency Contracts.

TYPE	Smart Contracts & Token
SOURCE CODE	https://github.com/Animocabrands/ethereum-contracts-erc20_base https://github.com/Animocabrands/f1dt-ethereum-contracts
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	May 05, 2020
REVISION DATE	July 22, 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 Animoca Brands team to audit the design and implementations of their ERC-20 smart contract that is meant to be utilized as a dependency by other projects. Additionally, the F1 Delta Time token implementation that derives from these contracts called REVV was audited. The audited source code links are:

- ERC-20: https://github.com/Animocabrands/ethereum-contracts-erc20_base/tree/13377ec388ce74eff2ffd9ca618bfb820a6aac9b
- Core Library Dependencies: https://github.com/Animocabrands/ethereum-contracts-core_library/tree/d141ca5fb91db43f08cbd87f2a2fa454f9a328d5
- REVV Token: https://github.com/Animocabrands/f1dt-ethereum-contracts/tree/218a19b4867561cc6690f9c2401eaaafb6d1311a

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 development team of Animoca Brands dealt with the ERC-related Exhibits of the report in the latest releases of each of the two GitHub repositories. Each “Alleviation” chapter of the Exhibits is meant to describe the actions that were taken in the following versions:

- ERC-20: https://github.com/Animocabrands/ethereum-contracts-erc20_base/tree/0b8845f805c91166db8314e9467d9f5e95f20cb7
- Core Library Dependencies: https://github.com/Animocabrands/ethereum-contracts-core_library/tree/d54e07b66dd8478ae5d2af5e33a6a35c0b6401e7

After updates were pushed to the original ERC-20 and Core library dependencies. The final commit hashes evaluated are as follows:

- ERC-20: https://github.com/Animocabrands/ethereum-contracts-erc20_base/tree/fd81eb7610528e3576f4b3bd56eb282614c16c0e
- Core Library Dependencies: https://github.com/Animocabrands/ethereum-contracts-core_library/tree/64ac2fb6bf3d9ed0500228fd8b188df1e23f81c0

Documentation

The sources of truth regarding the operation of the contracts were minimal at first and are something we advised to be expanded. The Animoca Brands team quickly took note of our suggestion and expanded the documentation of the REVV token both in terms of in-line comments as well as a dedicated markdown file that describes its intended purpose and functionality, greatly aiding in our understanding of each contract's functionality.

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

Summary

The codebase of the ERC-20 libraries and subsequent implementation conforms to the latest standards in the Solidity community and after our findings were assimilated the codebase achieved **a high standard of security as well as code ethics**.

While **most of the issues pinpointed were of negligible importance** and mostly referred to coding standards and inefficiencies that were informational and remediated to ensure the contracts of the Animoca Brands team are of the highest standard and quality.

These inefficiencies and flaws were swiftly dealt by the development team behind the Animoca Brands project and **a second round of auditing proceeded**. A direct communication channel between us and the Animoca Brands team was created and maintained throughout the iteration process to aid in amending the issues identified in the report.

The team presented **excellent skill in managing the issues that were identified within the report**, showcasing a dedicated and knowledgeable team behind the Animoca Brands project.

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**.

As all our findings were dealt with by the team, our sole recommendation is to retain the same level of work across any updates that occur on the codebase and ensure that subsequent versions apply strict linting rules and conform to the latest standards imposed by the Solidity development community.

Findings (Round 2)

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Unlocked Compiler Version Declaration	Language Specific Issue	Informational	First SLoC of all Contracts

[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.

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.

Alleviation:

The Animoca Brands team took note of this Exhibit and opted not to apply it as they wish to utilize the latest release of Solidity, as evident by the update upgrading the version from 0.6.6 to 0.6.8. Our point still stands, however the event of a compiler bug being discovered past the contract's deployment is admittedly low and as such, we consider this Exhibit accounted for.

Alleviation v2:

The second iteration of their contracts were locked at version 0.6.8, heeding our advice and ensuring that compiler-related bugs can easily be narrowed down should they occur.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Underscore Naming Convention Utilization	Coding Style	Informational	PayoutWallet.sol Lines 14, 20

[INFORMATIONAL] Description:

In general, underscore variables are used to denote private members of a contract. In this instance, the underscore prefix is incorrectly utilized in a “public” variable to discern it from the inputs of the “setPayoutWallet” function.

Recommendations:

In such instances, the underscore is instead used on the inputs of the function rather than the members themselves. This is to conform to the camel-case function / variable naming convention and do not allude to a variable being “private” due to its underscore prefix. We advise that the underscore is swapped between the “address payoutWallet” input of “setPayoutWallet” on L20 and the “address payable public _payoutWallet” on L14 of the contract.

Alleviation:

The update referenced to by this report reflects the aforementioned changes albeit in an unconventional fashion. Instead of using the underscore “_” as a prefix for arguments variables with a colliding name, it is used as a postfix. Additionally, the changelog of the difference states “Applied naming convention where missing: name must starting with `_` for `private` and `internal` variables and functions *only*” which is not true, as evident by multiple `internal` functions not conforming to the convention.

Alleviation v2:

N/A

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Underscore Naming Convention Utilization	Coding Style	Informational	ERC20Fees.sol Lines 20 - 21, 33, 36 - 37

[INFORMATIONAL] Description:

See Exhibit 2.

Recommendations:

See Exhibit 2.

Alleviation:

See Exhibit 2. Additionally, constants even when declared as internal should not have an underscore prefix as denoted by the Solidity styling guide.

Alleviation v2:

N/A

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Potential Overflow	Mathematical Operations	Informational	ERC20Fees.sol Lines 68, 85, 102

[INFORMATIONAL] Description:

The specified lines contain a multiplication of a “uint256” value with the contract member “_gasPriceScaling” followed by a division with the constant value “GAS_PRICE_SCALING_SCALE”.

Recommendations:

It is possible that the multiplication will overflow, causing incorrect values to be passed as the value to charge as a fee for the transaction since the variable “_gasPriceScaling” relies on user input. We advise the utilization of the already-imported “SafeMath” library to conduct the multiplication to ensure no overflow occurs however unlikely.

Alleviation:

The Exhibit was fully dealt with by introducing the “SafeMath” library and utilizing its functions for the multiplication operations.

Alleviation v2:

N/A

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
“super” Keyword Ambiguous Specifier	Coding Style	Informational	ERC20Fees.sol Lines 112, 122

[INFORMATIONAL] Description:

The specified lines utilize the “super” specifier to invoke a contract, however the “ERC20Fees” utilizes multiple inheritance.

Recommendations:

In Solidity you read class-name declarations right-to-left from lowest-to-highest in contrast to other programming languages such as Python where the opposite holds true. Thus, the “super” keyword may be ambiguous / confusing to a reader of the codebase. Although the code compiles correctly, the “super” keyword should be replaced with the name of the contract the function belongs to to aid in the legibility of the codebase.

Alleviation:

The “super” keyword was replaced by the explicit contract name “GSNRecipient” as per our suggestion.

Alleviation v2:

N/A

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Usage of Compiler-Swapped Value	Coding Style	Informational	ERC20Full.sol Line 8

[INFORMATIONAL] Description:

Line 8 of ERC20Full.sol contains the expression “2**256 - 1” which is equivalent to the maximum value a “uint256” type can hold.

Recommendations:

This value would otherwise overflow as “2**256” cannot be represented in the “uint256” limit. The compiler simply swaps “2**256 - 1” with the evaluated result, however this may lead readers of the codebase to assume it is possible to conduct such operations. We advise the utilization of a proper Solidity expression such as “~uint256(0)”, which inverts all the bits of the number zero represented in the form of a “uint256”.

Alleviation:

This Exhibit was attended to by completely removing the “UINT256_MAX” variable from the contract and instead replacing its instances with the “type” operator introduced in 0.6.0 and specifically the “max” member introduced in 0.6.8 applied on the “uint256” type.

Alleviation v2:

N/A

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Authoritative Access to Token Balances	Volatile Code	Informational	ERC20Full.sol Lines 44 - 46

[INFORMATIONAL] Description:

The “transferFrom” function contains statements that “bypass the internal allowance manipulation and checks for the whitelisted operator (i.e. _msgSender())” as stated in the comments.

Recommendations:

The “if” clause at L44 checks whether the caller of the “transferFrom” function is the “Operator”. If it is, the function proceeds by transferring the funds between the two accounts provided as input to the function regardless of whether the accounts have approved the “Operator”. Although this is a side-effect of what the contract is meant to achieve, utmost care should be taken when handling the private keys of the “Operator” as it allows one who holds the private key of an “Operator” to drain any account of token funds at will.

Alleviation:

An informative notice was included above the “transferFrom” function that warns users of the implications of maintaining a list of Operators, ensuring that users are fully aware of any potential unwanted consequences that may occur when the Operators are mishandled.

Alleviation v2:

N/A

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Superfluous “isOperator” Checks	Ineffectual Code	Informational	ERC20Full.sol Lines 66 - 74, 78 - 86

[INFORMATIONAL] Description:

The functions “increaseAllowance” and “decreaseAllowance” internally check whether the “spender” is an “Operator” and if so, return true as if they have succeeded.

Recommendations:

These checks are unnecessary as “increaseAllowance” and “decreaseAllowance” internally invoke “_approve” which is already overridden to simply “return” if the “spender” is an “Operator”. We advise the removal of the overridden functions “increaseAllowance” and “decreaseAllowance”.

Alleviation:

After collaboration with Animoca Brands, we deduced that overriding “increaseAllowance” and “decreaseAllowance” is necessary as the way the system works would lead to underflows occurring in their original implementation. Although they do internally invoke the “_approve” functions, the additional operations would throw an error because of the SafeMath library. This is a side-effect of the system returning the maximum “uint256” for “allowance” invocations but actually possessing the number “0” in the actual mapping. As a result of the aforementioned, this Exhibit is rendered null.

Alleviation v2:

N/A