CertiK Audit Report for Benchmark Protocol

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 Benchmark Protocol (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. For more information: https://certik.io.

Executive Summary

This report has been prepared for **Benchmark Protocol** to discover issues and vulnerabilities in the source code of the smart contract that we reviewed. 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 structures and implementations against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Testing Summary

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

TYPE	**Smart Contract**
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	Aug 28, 2020
REVISION DATE	Sep 14, 2020
METHODS	A comprehensive examination has been performed using Dynamic Analysis, Static Analysis, and Manual Review.

Review Notes

Introduction

CertiK team was contracted by Benchmark Protocol to audit the design and implementation of Smart Contracts for security vulnerabilities and compliance with Solidity language standards under different perspectives and with different tools such as static analysis and manual reviews by smart contract experts to find specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole. The work will conduct and analyze under different perspectives and with different tools such as CertiK formal verification checking as well as manual reviews by smart contract experts. We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

The audited files and their associated sha256 hashes are:

1. uFragments.sol

`5275e3bef53b10ba1f9efe4089823a1a88ea0da4e177ccb85df9cef54950c465`

2. Molecules_0.2 (1).sol

`93d4c3664d44eeb3629880c317cfae3059c768c886deea1ad85b97af2d288a15`

Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Uncommon Naming Convention	Naming Conventions	Informational	uFragments.sol, L72-L73

[INFORMATIONAL] Description:

The linked variables were declared as constant yet did not conform to the uppercase format naming convention.

Recommendation:

We suggested that these variable names should be adjusted in order to conform to the Solidity style guide.

Alleviation:

The constant variable names were adjusted from mixed case to uppercase.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Pull-over-Push Pattern	Access Control	Minor	uFragments.sol, L113-L118

[MINOR] Description:

Owner management was not handled correctly through the pull over push pattern. It is a general practice to propose a new owner of a contract that proceeds to accept the invitation rather than directly overriding the current one as mishandling can forever lock the ownership of the contract.

Recommendation:

We advised that the Pull-over-Push pattern is applied here whereby a new owner is proposed and then the owner himself can accept the ownership in a separate function call.

Alleviation:

A newOwner state variable was added to the contract and management is correctly handled via the Pull-over-Push pattern.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Event Emittance	Language Specific	Minor	uFragments.sol, L130, L145

[MINOR] Description:

We pointed out that in the case that supplyDelta is equal to zero, the event LogRebase will be emitted twice incorrectly. Additionally, the if clause of L129-L131 is presumably meant to return after the event is emitted.

Recommendation:

We advised that the if clause should be refactored.

Alleviation:

A return statement was introduced in the case that supplyDelta is equal to zero in order to prevent the LogRebase event from being emitted twice.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Redundant Casting	Conversion	Informational	uFragments.sol, L134, L136

[INFORMATIONAL] Description:

We pointed out a case of the redundant casting of the supplyDelta variable to a uint256 whilst the variable itself was already of uint256 type.

Recommendation:

We advised that this casting should be removed.

Alleviation:

The redundant casting of the supplyDelta variable was removed.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Invalid Error Message/Check	Implementation	Informational	uFragments.sol, L165, L194

[INFORMATIONAL] Description:

We noticed that the linked require checks were either incorrect or contained an incorrect error message as they verified that the to address did not represent the address of the contract (this), whilst the error message stated that transfers cannot occur to the same address i.e. that msg.sender should not be equal to to.

Recommendation:

We advised that the linked code segments should be revised according to their desired functionality.

Alleviation:

The require message was refactored to more clearly convey its functionality.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Redundant require check	Implementation	Minor	uFragments.sol, L195

[MINOR] Description:

We noticed that the linked require check was internally checked in the SafeMath sub invocation of L197 and suggested that it could be safely omitted. If a custom error message is desired, a SafeMath alternative library exists where users can provide the error message as input.

Alleviation:

The redundant require statement was removed and is safely handled in the SafeMath functions.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Potential Incompatibility w/ ERC-20 Standard	Implementation	Minor	uFragments.sol, L214-L219

[MINOR] Description:

The measurement unit utilized in the approve function was different than the actual value utilized in transfer calls which caused an incompatibility according to the ERC-20 specification. The ramifications of this should be evaluated as on-chain contracts may rely on chained approve and transfer calls i.e. in the case of DeFi.

Alleviation:

No change was made and the issue still exists.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Literals to constant	Implementation	Minor	uFragments.sol, L87-L90

[MINOR] Description:

We suggested that the assignments of those lines could instead be directly made on the respective variable declarations to be able to declare them as constant thus greatly optimizing the gas cost involved in utilizing them, or that the decimals variable of L69 should be moved before the string declarations to ensure that it is tightly packed with the owner variable thus optimizing the storage space of the contract.

Alleviation:

No change was made and the issue still exists.