CertiK Audit Report for MAINSTON

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 Validity Labs (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 **Validity Labs** to discover issues and vulnerabilities in the source code of their **StonToken** smart contract. 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 contract 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 **Validity Labs**. This audit was conducted to discover issues and vulnerabilities in the source code of **Validity Labs' StonToken** Smart Contract.

TYPE	**Smart Contract**
SOURCE CODE	https://gitlab.com/validitylabs/mainston/smart-contract-audit/-/blob/master/contracts/6/v2/StonToken.sol
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	Aug 05, 2020
REVISION DATE	Aug 12, 2020
METHODS	A comprehensive examination has been performed using Dynamic Analysis, Static Analysis, and Manual Review.

Review Notes

Introduction

CertiK team was contracted by Validity Labs to audit the design and implementation of their STON v2 Token Smart Contract and its compliance with the ERC-20 Standard.

The audited source code link is:

- Token Source Code: https://gitlab.com/validitylabs/mainston/smart-contract-audit/-/blob/master/contracts/6/v2/StonToken.sol

The goal of this audit is 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.

Summary

We used the following source of truth about how the **STON v2** token should work: https://gitlab.com/validitylabs/mainston/smart-contract-audit/-/tree/master#ston-v2

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.

Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Incorrect or redundant implementation of overridden function.	Language Specific	Minor	StonToken, L169-175

[MINOR] Description:

Due to C3 linearization, the right-most contract with the function in the inheritance list is chosen, which is “ERC20Capped” in this case. This means that “super._beforeTokenTransfer” evaluates to “ERC20Capped._beforeTokenTransfer”, which is the original functionality without overriding the function, making the override unnecessary.

Recommendation:

Reference the Solidity 0.6 documentation for inheritance:

https://solidity.readthedocs.io/en/v0.6.0/contracts.html#inheritance

If the intention is to bypass the check created in “ERC20Capped._beforeTokenTransfer”, explicitly specify “ERC20._beforeTokenTransfer”:

```
function _beforeTokenTransfer(
    address from,
    address to,
    uint256 amount
) internal virtual override(ERC20, ERC20Capped) {
    ERC20._beforeTokenTransfer(from, to, amount);
}
```

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Improper guarding against multiple calls to “removeUpgradeEngine”.	Access Controls	Minor	StonToken, L73-76

[MINOR] Description:

While this issue does not compromise the system, the owner of the token has the ability to call “removeUpgradeEngine” multiple times after minting due to the lack of requiring the swap engine address to be non-zero or setting an explicit “upgradeFinished” state.

Recommendations:

Either change the requirements of “removeUpgradeEngine” to include that the swap engine address should be non-zero, or add an “upgradeFinished” state variable to the requirements of “removeUpgradeand verify that it is set after removing the swap engine:

```
require(
    upgradeStarted,
    "Upgrade has not started yet. Cannot reset the engine address!"
);
require(
    upgradeEngine != address(0),
    "Upgrade engine address has already been reset!"
);
```

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Incorrect spelling in comment.	Grammar	Informational	StonToken, L134

[INFORMATIONAL] Description:

While this issue does not compromise the system, fixing it may lead to an improvement in the overall code/syntax or more clearly convey the intended functionality.

Recommendation:

Change “forbide” to “forbid”.