CERTIK

CertiK Audit Report for Social Good Token

Contents

Contents 2

Disclaimer 3

About CertiK 3

Executive Summary 4

Testing Summary 5

Review Notes 6

- Introduction 6
- Documentation 6
- Summary 7
- Recommendations 7

Findings 8

Exhibit 1 8

Exhibit 2 9

Exhibit 3 10

Exhibit 4 11

Exhibit 5 12

Exhibit 6 13

Exhibit 7 14

Exhibit 8 15

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 SocialGoodToken (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 **SocialGoodToken** to discover issues and vulnerabilities in the source code of their **ERC SG Token** 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 SocialGoodToken.

This audit was conducted to discover issues and vulnerabilities in the source code of SocialGoodToken's SG ERC token.

TYPE	Smart Contract
SOURCE CODE	https://bitbucket.org/socialgood-foundation/sg-token/src/develop/
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	July 10, 2020
DELIVERY DATE	July 13, 2020 July 17, 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 SocialGoodToken team to audit the design and implementation of their SG ERC token smart contract as well as the timelock mechanism of team tokens of the contract.

The audited source code link is:

- SG Source Code: https://bitbucket.org/socialgood-foundation/sg-token/src/d334beee91315f285da200435d3d97513a36f5b9/

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 SocialGoodToken swiftly dealt with the issues at hand and produced a new commit, the source code link of which is as follows:

- SG Source Code: https://bitbucket.org/socialgood-foundation/sg-token/src/02cb7f11faef23c8c008755bbcf5cacb23c46e8d/

After clarifying certain Exhibits that were unattended to in this version, a new version was produced by the SocialGoodToken team that can be deemed as the final version of this report:

- SG Source Code: https://bitbucket.org/socialgood-foundation/sg-token/src/7f54caa68fe7bc93cbbb6ddde6eceb0a09dcd79d/

Documentation

The sources of truth regarding the operation of the contracts in scope were lackluster and **are something we advise to be enriched to aid in the legibility of the codebase as well as project**. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions.

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

Summary

The codebase of the project is a typical ERC implementation and the locking mechanism of the token is derived from an officially recognized library, specifically TokenTimelock from OpenZeppelin.

Certain optimization steps that we pinpointed in the source code mostly referred to coding standards and inefficiencies, however **2 minor vulnerabilities were identified during our audit that solely concern the specification**. The codebase of the project strictly adheres to the standards and interfaces imposed by the OpenZeppelin open-source libraries and **can be deemed to be of high security and quality**.

Certain discrepancies between the expected specification and the implementation of it were identified and were relayed to the team, however they pose no type of vulnerability and concern an optional code path that was unaccounted for.

The SocialGoodToken team assimilated our preliminary findings in a very short timeframe and produced an updated version of the codebase that satisfied all the requirements they had laid out in the checklist document they had sent us.

In detail, the specifications they desired their token to conform to are as follows:

- Issue Limit: 210,000,000 SG
    - Airdrop Segment: 750,000 SG
    - Minting Segment: 209,250,000 SG
- Movement Limit: None
    - Airdrop Token Limit: Manual Lockup

- Contract Owner's Abilities
    - Burning of Personal Balance
    - Issuance of New Balance

Our initial findings found discrepancies with regards to the issuance limits and the airdrop lockup, however in the second version of the codebase satisfies all points of the aforementioned checklist except for the issuance limits.

In the final and third version of the codebase, the SocialGoodToken team fixed the issuance limit discrepancies by adjusting their "ERC20Capped" dependency as per our recommendation. During this process, certain minor optimizations were pointed out as well that were not included in the final report as they were deemed negligible.

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 crucial Exhibits were dealt with, we can safely say that the codebase is at a healthy state and conforms to the latest security standards of the Solidity community, thus safely labeling this project as secure.

Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Unlocked Compiler Version	Language Specific	Informational	SocialGoodToken.sol: L1

[INFORMATIONAL] Description:

The SG ERC token contract “pragma” statement regarding the compiler version indicates that version 0.5.0 or higher should be utilized.

Recommendations:

We advise that the compiler version is locked at version 0.5.0 or whichever Solidity version higher than that satisfies the requirements of the codebase as an unlocked compiler version can lead to discrepancies between compilations of the same source code due to compiler bugs.

Alleviation:

The compiler version was locked at 0.5.17 per our recommendation, ensuring that any future compiler bugs that may pop up can be pinpointed more easily.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Unchanged Contract Variables	Optimization	Informational	SocialGoodToken.sol: L21 - L24

[INFORMATIONAL] Description:

These variables are declared once and assigned to in their declaration with their values remaining unchanged throughout the lifetime of the contract.

Recommendations:

We advise these variables are instead changed to “constant” as they are not meant to change and would optimize the gas cost of all functions that relate to them. Additionally, the “MIXED_UPPERCASE” format could be followed for the “_hardCap” variable as per the Solidity style guide.

Alleviation:

The recommendation with regards to the naming convention of “_hardCap” was followed, although the contract members between L21 and L23 were not changed to “constant” per our recommendation.

Alleviation v2:

The variables were properly adjusted to be “constant” thus conforming to this Exhibit in full.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	SocialGoodToken.sol: L116, L152

[INFORMATIONAL] Description:

The lines above conduct a greater-than “>” comparison between unsigned integers and the value literal “0”.

Recommendations:

As unsigned integers are restricted to the positive range, it is possible to convert this check to an inequality “!=” reducing the gas cost of the function.

Alleviation:

Our recommendation was followed to the letter by changing the “greater-than” comparisons with “inequality” comparisons.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Inconsistent Implementation w/ Spec	Ineffectual Code	Minor	SocialGoodToken.sol: L92, L100 - L104

[MINOR] Description:

The function “airdropMint” is meant to mint locked tokens for a specific airdrop address. While this is possible with the current implementation, a separate code path exists should the input variable “isLock” be false whereby the tokens are directly minted to the airdrop address.

Recommendations:

As the specification provided to us by the SocialGoodToken team states that all airdrops should be locked up, we advise that the input variable is omitted and the “if” clause of L100 - L104 is replaced by L101.

Alleviation:

The input variable was indeed omitted per our recommendation and the code refactored to not rely on it and instead execute a single code path.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Inconsistent Airdrop Cap	Ineffectual Code	Minor	SocialGoodToken.sol: L108 - L121

[INFORMATIONAL] Description:

The "_preValidateAirdrop" function which validates an airdrop before it occurs does not guarantee any specific cap on the total amount of tokens minted via airdrop.

Recommendations:

As the specification provided to us by the SocialGoodToken team states that an airdrop limit of 750000 tokens should be imposed, we advise that a separate "airdropCap" variable is retained within the contract and added to to ensure that the cap of 750000 is never exceeded.

Additionally, the "ERC20Capped" contract inheritance may not be suitable for the current implementation as it only supports a single cap. This means that the tokens minted for the airdrop can overlap with the tokens minted via the normal way, thus causing an issue and surplus locked up tokens. To solve this, a custom implementation of how the "ERC20Capped" function needs to be made that checks the type of token being minted (airdrop or normal) and tests it against each respective limit by overriding the "_mint" function.

Alleviation:

An additional check was added to "_preValidateAirdrop" that ensures consecutive airdrops will not surpass the airdrop cap limit. However, the current implementation does not solve the latter paragraph of the two described in Recommendations.

Specifically, even if the airdrop tokens are “airdropped”, the minting cap is not affected. This means that it is possible to airdrop the full cap amount as well as mint the full limit, ultimately having “minted” a total of tokens that surpasses the actual limit.

Two ways can be used to tackle this issue. The first way is to override the “_mint” function to also verify the minting cap of both airdropped tokens and minted tokens. The second way is to instead directly mint the tokens of the airdrop to an address and revamp the lockup system whereby a mapping from an address to a boolean is used instead that prevents transfer of tokens.

The former way is more gas efficient yet slightly more complex and we leave the remediation action up to the team to choose.

Alleviation v2:

The “ERC20Capped.sol” dependency was adjusted to accept two independent caps instead of one and instead of relying on the “totalSupply” variable retain two separate variables postfixed with the word “used” to indicate how much of a particular cap has been utilized. The dependency also exposes an “unguarded” minting method that does not verify the cap and is available via “_airdropMint” to allow any airdropped tokens to be minted as the cap of the airdrop is tracked at the locking process rather than the minting process.

Factoring the above into account, we can conclude that the caps of the project were properly set and this Exhibit has been accounted for.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Lower-Than Comparison	Coding Style	Informational	SocialGoodToken.sol: L98, L141

[INFORMATIONAL] Description:

These two comparisons concern the “for” loops of “airdropMint” and “unlocks” and check whether the current iteration variable “i” is less than the “length” of the total accounts that are meant to be iterated in each respective function.

Recommendations:

We advise this comparison to be converted to an inequality comparison as the unsigned integer “i” is sequentially increased and the inequality and less-than comparison are logically equal yet the former is more optimal gas-wise.

Alleviation:

The conditions of the “for” loops remained unchanged.

Alleviation v2:

The conditions of the “for” loops were adjusted to utilize an inequality comparison for efficiency.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Redundant “require” Check	Ineffectual Code	Informational	SocialGoodToken.sol: L147 - L150

[INFORMATIONAL] Description:

The “_unlock” function internally checks whether the address meant to be unlocked is equal to the zero address and reverts if so to prevent unlocking tokens for the zero address.

Recommendations:

The function “_preValidateAirdrop” conducts the exact same “require” statements and as such guarantees that no tokens will ever be minted for the zero address. As such, the “require” check between L151 - L154 covers the case of the zero address and thus the “require” check of L147 to L150 should be omitted.

Alleviation:

The “require” check was properly omitted from the “_unlock” function optimizing its gas cost.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Potential Misleading “burnFrom” Functionality	Ineffectual Code	Informational	SocialGoodToken.sol: L81 - L83

[INFORMATIONAL] Description:

The function “burnFrom” is only invoke-able by the owner of the contract and allows the owner to burn a specific amount of tokens from an address.

Recommendations:

The function “burnFrom” internally ensures that the “owner” has enough “allowance” from the address the tokens will be burned from before actually burning the tokens. As this is meant to act as an administrative function, perhaps its execution flow should be re-evaluated as it will not allow the owner to burn tokens from an arbitrary address at its current state as it would need to call “_approve” with the correct function arguments.

Alleviation:

After consultation with the SocialGoodToken team, they stated that this is intended behavior and that the owner is not meant to burn tokens from an arbitrary address and is instead meant to burn tokens from addresses that have allowed it to do so, conforming to the specification of the current codebase.