CERTIK

CertiK Audit Report for Sanu Gold

Contents

Contents **1**

Disclaimer **2**

About CertiK **2**

Executive Summary **3**

Testing Summary **4**

Review Notes **5**

- Introduction 5
- Documentation 6
- Summary 6
- Recommendations 7

Preliminary 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**

Exhibit 9 **16**

Exhibit 10 **17**

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 Sanu Gold (the “Company”), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the “Agreement”).

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 **Sanu Gold** to discover issues and vulnerabilities in the source code of their **SNG 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 Sanu Gold.
This audit was conducted to discover issues and vulnerabilities in the source code of Sanu Gold's SNG ERC token.

TYPE	Smart Contracts & Token Wrappers
SOURCE CODE	https://github.com/Kiansmithhq/sanu-gold-contract/tree/08b93804850de3a28e8467cf9db88c38750f401a
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	July 6, 2020
DELIVERY DATE	July 07, 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 Sanu Gold team to audit the design and implementation of their SNG ERC token smart contract with enhanced functionality such as address freezing and on-chain fee redemption.

The audited source code links are:

- SNG Contract Source Code: https://github.com/Kiansmithhq/sanu-gold-contract/tree/08b93804850de3a28e8467cf9db88c38750f401a

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 Sanu Gold team evaluated our exhibits and chose to apply them in an updated commit of the codebase that can be followed below:

- SNG Contract Source Code: https://github.com/Kiansmithhq/sanu-gold-contract/tree/db5e87d8a7e965ce8db8068ec11bc7c0d8728365

Documentation

The sources of truth regarding the operation of the contracts in scope were minimal and are **something we would advise to be expanded**. 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 Sanu Gold team or reported an issue.

Summary

The codebase of the project is relatively straightforward, however the additional functionality that was added with regards to freezing accounts contained exploits that should be remediated as soon as possible.

Some optimization steps were also **pinpointed but were of negligible importance** and mostly referred to coding standards and inefficiencies. **Medium severity flaws** were identified that should be remediated as soon as possible to ensure the contract of the Sanu Gold token is of the highest standard and quality.

The team **enforced a strict linting style and assimilated all the Exhibits we identified in our report** within a very short timeframe, indicating **the team's dedication to a secure and maintainable codebase**.

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 these steps were conducted, we believe that the Sanu Gold codebase has evolved into a secure and production-ready project with a capable team of developers that showcase affinity towards secure coding practices.

Preliminary Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Error Message	Ineffectual Code	Informational	SNG.sol: L57

[INFORMATIONAL] Description:

The error message of the “onlyOwner” modifier states “only FeeController”.

Recommendations:

We advise the error message be changed to properly reflect that the correct owner was not identified.

Alleviation:

This Exhibit was remediated according to our recommendation by adjusting the error message to “only Owner”.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Redundant Zero Assignment	Ineffectual Code	Informational	SNG.sol: L63

[INFORMATIONAL] Description:

The aforementioned line contains an assignment of the value literal “0” to the contract variable “feeRate”.

Recommendations:

All variables in Solidity are initialized at their default value, in this case zero, and as such this assignment is redundant. We advise it be removed.

Alleviation:

This Exhibit was remediated according to our recommendation by completely omitting the redundant assignment of zero to “feeRate”.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	SNG.sol: L83, L106

[INFORMATIONAL] Description:

Within the function, a greater-than “>” comparison is done between the unsigned integer “fee” 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:

This Exhibit was partially remediated according to our recommendation by converting the comparison of L83 to an inequality one and L106 remaining as is. We advise that L106 is adjusted to conform to the format of L83.

Alleviation v2:

Line 106 was adjusted to conform to the format of L83 according to our second suggestion, fully attending to this Exhibit.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Frozen Check	Logic Issue	Medium	SNG.sol: L98

[MEDIUM] Description:

The “transferFrom” function allows an authorized party to transfer funds between two addresses. This party can be unrelated to the “from” and “to” addresses of the transfer whilst the currently imposed check ensures that the “to” address and the “msg.sender” are not frozen.

Recommendations:

The current check allows one to transfer funds out of a “frozen” address by having set their allowance to the maximum prior to being frozen. As the “from” address is not checked, a frozen “from” address can still transfer funds out of its account via an allowance delegation. We advise that the “from” address is checked instead of the “msg.sender” on this line to alleviate the issue.

Alleviation:

This Exhibit was fully remediated according to our recommendation by adding an additional check that ensures the recipient of the funds is not a “frozen” address. The line right now verifies that the “from”, “to” as well as “msg.sender” variables are not “frozen”. As “msg.sender” here is the caller of the function and will not possess any tokens at any point of the function, it is possible to remove the check of whether “msg.sender” is “frozen”. However, this may be desirable functionality and we leave that up to the Sanu Gold team’s discretion.

Alleviation v2:

The team decided to remove the extraneous check for “msg.sender” after we detailed its purpose. We would like to state that the current checks verify tokens will never be transmitted “from” and “to” a frozen address, however it permits “allowed” addresses to move funds between non-frozen addresses according to their “allowance”. As per our closing point on the “Alleviation” paragraph, the Sanu Gold team evaluated this and decided it is desirable functionality.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Code Duplication	Ineffectual Code	Informational	SNG.sol: L118 - L122, L128 - L132

[INFORMATIONAL] Description:

The functions “freeze” and “unfreeze” are identical in nature differing in the literal they assign and the event they emit.

Recommendations:

We advise that both functions are merged into a single function called “toggleFreeze” that simply negates the currently held value of “frozen[_addr]” and emits the corresponding event based on an “if-else” statement.

Alleviation:

This Exhibit was fully remediated according to our recommendation by combining those two functions into a single “toggleFreeze” function. An additional optimization step is possible whereby a local “bool” variable is declared equal to “!frozen[_addr]” and subsequently assigned to “frozen[_addr]” and checked in the “if” clause. This would reduce the number of storage reads from 2 to 1.

Alleviation v2:

The additional optimization step we proposed was fully assimilate in the codebase to reduce the number of storage reads.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Invalid Wiping Logic	Logic Issue	Medium	SNG.sol: L139 - L144

[MEDIUM] Description:

The ERC standard dictates that a user who invokes the “burnFrom” function needs to have the necessary “allowance” from the owner of the funds that are being burned. In the case of “wipeFrozenAddress”, the “burnFrom” function is called without any type of “allowance” change between the “msg.sender”, a.k.a. the “owner”, and the “_addr” that will be wiped. This leads to the function always failing and being impossible to wipe the balance of a frozen address.

Recommendations:

The correct approach would be to directly set the allowance to the balance of the user prior to the execution of the “burnFrom” function of “super” to make sure the function call goes through and the address is properly wiped.

Alleviation:

This Exhibit was fully remediated according to our recommendation by properly invoking “_approve” with the correct arguments before the “burnFrom” function occurs.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Redundant Zero Equality	Optimization	Informational	SNG.sol: L196 - L198

[INFORMATIONAL] Description:

The lines of code mentioned above contain a logic check for the state variable “feeRate”, checking whether it is zero and returning the numeric literal “0” if that’s the case.

Recommendations:

This logical check is internally conducted in the “mul” function of the “SafeMath” library and as such can be safely removed here.

Alleviation:

This Exhibit was fully remediated according to our recommendation by omitting the “if” clause completely.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Invalid Function Overriding	Compilation Error	Informational	SNG.sol: L62

[INFORMATIONAL] Description:

The function “initialize” with the signature of “(string memory name, string memory symbol)” is labelled as “virtual” and “override” whereas the contracts fail to compile since the “initialize” function of the derived contracts is not “virtual”.

Recommendations:

We advise that either a “constructor” or an “initialize” function with a different function signature is utilized to instantiate the contract state variables.

Alleviation:

This Exhibit was fully remediated by the Sanu Gold team by relocating the OpenZeppelin contracts and fixing the compilation issues that were previously prevalent.

Exhibit 9

TITLE	TYPE	SEVERITY	LOCATION
Illegible Code Layout	Coding Style	Informational	Full Contract

[INFORMATIONAL] Description:

The codebase of the contract indicates no use of any type of linting tools.

Recommendations:

We advise that linting tools are used to enforce a strict style guide and enhance the readability of the codebase.

Alleviation:

This Exhibit was fully remediated according to our recommendation by enforcing a linting style throughout the project via Ethlint / Solium, increasing the legibility of the codebase.

Exhibit 10

TITLE	TYPE	SEVERITY	LOCATION
Event Inputs Optimization	Optimization	Informational	SNG.sol: L160, L171, L183

[INFORMATIONAL] Description:

These lines emit an event with the latter of the two input parameters being read from storage.

Recommendations:

As this value is equal to the input variable of the function, we advise that the input variable of the function is utilized instead as it is stored in memory rather than storage and would reduce the gas cost of these functions.

Alleviation:

The emittance of the events was adjusted to utilize in-memory variables rather than storage variables, attending to this Exhibit in full.