SantaCash

Security Assessment

February 8th, 2021

[Preliminary Report]

For :
SantaCash

By :
Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org

Angelos Apostolidis @ CertiK
angelos.apostolidis@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	SantaCash
Description	A typical ERC20 implementation with enhanced features.
Platform	Ethereum; Solidity, Yul
Codebase	EtherScan code

Audit Summary

Delivery Date	February 8th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	February 5th, 2021 - February 8th, 2021

Vulnerability Summary

Total Issues	6
Total Critical	0
Total Major	0
Total Medium	0
Total Minor	0
Total Informational	6

Executive Summary

This section will represent the summary of the whole audit process once it has concluded.

Findings

ID	Title	Type	Severity	Resolved
STC-01	Library Naming	Coding Style	Informational	!
STC-02	`external` Over `public` Function	Gas Optimization	Informational	!
STC-03	User-Defined Getters	Gas Optimization	Informational	!
STC-04	Variable Mutability Optimization	Gas Optimization	Informational	!
STC-05	Redundant Function	Gas Optimization	Informational	!
STC-06	Ambiguous Use of `virtual`	Volatile Code	Informational	!

STC-01: Library Naming

Type	Severity	Location
Coding Style	Informational	L19

Description:

The `safeMath` library name is not in CapWords.

Recommendation:

We advise to closely follow the Solidity naming conventions.

STC-02: `external` Over `public` Function

Type	Severity	Location
Gas Optimization	Informational	L79 , L82 , L85 , L88 , L91 , L94 , L101 , L105 , L110, L114

Description:

The linked functions remain unused by the contract.

Recommendation:

We advise that the linked functions have their visilibity changed to `external` to save gas.

STC-03: User-Defined Getters

Type	Severity	Location
Gas Optimization	Informational	L79-L90

Description:

The linked variables contain user-defined getter functions that are equivalent to their name barring for an underscore (`_`) prefix / suffix.

Recommendation:

We advise that the linked variables are instead declared as `public` and that they are renamed to their respective getter's name as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

STC-04: Variable Mutability Optimization

Type	Severity	Location
Gas Optimization	Informational	L63-L65

Description:

The linked state variables are assigned to a literal in the constructor and are not updated afterwards.

Recommendation:

We advise to change the mutability of the linked variables to `immutable` to save gas.

STC-05: Redundant Function

Type	Severity	Location
Gas Optimization	Informational	L146-L148

Description:

The use of the `_setupDecimals()` function is to change the value of the `_decimals` state variable in the constructor of an inhereting contract.

Recommendation:

We advise to remove the linked function.

STC-06: Ambiguous Use of `virtual`

Type	Severity	Location
Volatile Code	Informational	General

Description:

The functions in the `STC` contract ambiguously use the keyword `virtual`, as they are not expected to be overriden.

Recommendation:

We advise to remove the keyword `virtual` from the linked functions.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.