PUBLIC

Code Assessment

of the SBC Deposit Smart Contracts

October 01, 2021

Produced for

POA

by

CHAINSECURITY

Contents

1 Executive Summary

Dear POA Network team,

First and foremost we would like to thank you for giving us the opportunity to assess the current state of their SBC Deposit system. This document outlines the findings, limitations, and methodology of our assessment.

We uncovered one critical issue that was fixed after our intermediate report. The remaining issues were of low severity and were fixed accordingly. The communication with the team was very responsive and good.

We hope that this assessment provides more insight into the current implementation and provides valuable findings. We are happy to receive questions and feedback to improve our service and are highly committed to further support your project.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	1
• Code Corrected	1
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	2
• Code Corrected	2

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the SBC Deposit repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	12 September 2021	31e14c00b056ad2170371f71ca62b2c333016e2a	Initial Version
2	28 September 2021	ac8cca4460d203037c043b4779d29ff80277f202	Second Version

For the solidity smart contracts, the compiler version `0.8.7` was chosen.

The following files were in the scope of the audit:

- interfaces/
  - IDepositContract.sol
  - IERC165.sol
  - IERC20.sol
  - IERC667.sol
  - IERC677Receiver.sol
- utils/
  - Claimable.sol
  - EIP1967Admin.sol
  - EIP1967Proxy.sol
- StakeDepositContract.sol
- StakeDepositContractProxy.sol

2.1.1 Excluded from scope

Excluded is the correctness of the incremental Merkle tree algorithm, which has been formally verified here.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview. Furthermore, in the findings section we have added a version icon (in case of different versions) to each of the findings to increase the readability of the report.

POA Network offers a Stake Beacon Chain (SBC) deposit contract that is supposed to be used by stakers in the context of a Proof-of-Stake consensus. Stakers will first come to an agreement with a

validator node about the amount to stake, then it will deposit the agreed-on stake amount to a deposit contract, such as the one proposed by POA Network.

The contract is based on the original Ethereum 2.0 deposit contract, but SBC Deposit adds extended functionality to it:

- *ERC20 deposits*: Stakers can deposit ERC20 STAKE tokens instead of native tokens
- *batch deposits on top of normal deposits*: batch deposits are fixed at 32 STAKE per deposit and normal deposits are floored to 1 STAKE
- *support for ERC677*: Adds a hook on ERC20 tokens transfer to trigger token receiver
- *upgradeability*: A proxy pattern is used to have the ability to upgrade the implementation contract
- *claimability*: An `admin` is able to withdraw any mistakenly sent non-STAKE tokens (ERC20 or native) in order to give them back to their owners
- *contract can be paused*: This functionality is only available for the `admin`

As the original contract, `StakeDepositContract` implements an incremental Merkle tree algorithm to keep track of the deposits' history. It can contain up to `2^32 - 1` deposit records and allows root computation in `O(log(n))`.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts associated with the items defined in the engagement letter on whether it is used in accordance with its specifications by the user meeting the criteria predefined in the business specification. We draw attention to the fact that due to inherent limitations in any software development process and software product an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account the facts that over the life cycle of any software product changes to the product itself or to its environment, in which it is operated, can have an impact leading to operational behaviors other than initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severities. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved, have been moved to the Resolved Findings section. All of the findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	1

- onTokenTransfer Wrong Accounting Code Corrected

High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	2

- EIP1967 Storage Addresses Code Corrected
- Interface Name Mismatch Code Corrected

6.1 onTokenTransfer Wrong Accounting

Correctness Critical Version 1 Code Corrected

When triggering `onTokenTransfer` with only one deposit, `stake_amount` is assumed to be 32 STAKE but is not checked. This allows a depositor to call `STAKE.transfer` with 1 STAKE but to be accounted for 32 in the Merkle tree.

Code corrected

In case of a single transfer the amount is set to the transferred amount specified. In batch transfers it is set to 32 Ether. This behavior is coherent with the behavior of single deposits and batch deposits.

6.2 EIP1967 Storage Addresses

Design Low Version 1 Code Corrected

Hard-coding long strings could be error prone. The storage addresses of `implementation` and `admin` are hardcoded as hex values in multiple places. This adds complexity and is even more error prone. A clean mitigation would be to set the storage addresses used for `implementation` and `admin` as constants once and use this constant later on.

Code corrected

The hard coded strings were replaced by constants.

6.3 Interface Name Mismatch

Design Low Version 1 Code Corrected

The name of the file and interface `IERC667` should be `IERC677` to be compliant with the transferAndCall Token Standard.

Code corrected

The file was renamed with the appropriate name.

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 Admin Power

Note Version 1

The system is administrated. The admin account has the complete power including withdrawing all funds by updating the implementation contract. Hence, the security of the admin key is of utmost importance as well as the trust in the key holder/s. POA network provided the information that they are aware of this note and will keep the admin contract under control by a multisig.

7.2 Locked Tokens in Implementation

Note Version 1

The `admin` can withdraw all tokens balances from the proxy, but funds sent to the implementation contract are stuck.

POA Network reasoned that from their past experience with user support requests, users tend to mistakenly send tokens only to the proxy contract, since only its address is being publicly advertised. Implementation contract is “hidden” from regular users, so they are unlikely to send tokens to that address.

7.3 Proxy/implementation Separation

Note Version 1

- The proxy takes care of the zero hashes initialization. To be consistent, this logic should belong into the implementation contract.
- The `admin` is able to act on both the proxy and implementation logics. It is usually a good practice to separate roles of proxy from roles of implementation.

POA network is aware of this note and explained that this is the intended behavior.