

Produced for



by



Contents

1	Executive Summary	3
2	Assessment Overview	4
3	Limitations and use of report	6
4	Terminology	7
5	Findings	8
6	Resolved Findings	9

1 Executive Summary

Dear Sir or Madam,

First and foremost we would like to thank POA Network 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.

The contracts make correct use of battle proof libraries and are based on established contracts. We did not find any severe issues but three low design issues which are described in our issue section. All issues have been fixed accordingly. We value the good and professional communication with your team.

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.

-Severity Findings	0
-Severity Findings	0
-Severity Findings	0
-Severity Findings	3
•	3

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. The issues pointed out by the previous report from 01 October 2021 are still to be taken into account.

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	27 October 2021	a1c7e9ec971b4e1207f742046bebd2a35d4625ba	Initial Version
2	04 November 2021	7f2508f41881c468bdd26c60551b9d7e1404bb25	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
 - IERC667.sol
 - IERC677Receiver.sol
 - IPermittableToken.sol
- utils/
 - Claimable.sol
 - EIP1967Admin.sol
 - EIP1967Proxy.sol
 - PausableEIP1967Admin.sol
- SBCDepositContract.sol
- SBCDepositContractProxy.sol
- SBCToken.sol
- SBCTokenProxy.sol
- SBCWrapper.sol
- SBCWrapperProxy.sol

2.1.1 Excluded from scope

Excluded is the correctness of the incremental Merkle tree algorithm, which has been formally verified [here](#).

We assume the admin/governance and all connected token contracts to be non-malicious.

2.2 System Overview

This system overview describes the initially received version () 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.

2.2.1 SBC Deposit contract :

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 SBC tokens instead of native tokens
- *batch deposits on top of normal deposits*: batch deposits are fixed at 32 SBC per deposit and normal deposits are floored to 1 SBC
- *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-SBC 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`
- *validator unique withdrawal credentials*: Each validator is allowed a unique withdrawal credentials to mitigate the following front-running attack <https://medium.com/immunefi/rocketpool-lido-frontrunning-bug-fix-postmortem-e701f26d7971>

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 $O(\log(n))$.

2.2.2 SBC Token contract :

This ERC20 token is used for staking on the beacon chain. Only the `admin` (set to the wrapper contract) can mint and the token cannot be burned.

2.2.3 SBC Wrapper contract :

This contract is used to wrap authorized tokens into SBC token. The word wrapping might be misleading. Generally, it is a one way swap because no unwarped or swapping back exists. Users can either swap their tokens with the ERC677 hook `onTokenTransfer` or with the `swap` function, which they need to provide some permission data along.

All three contracts are pausable, claimable (possibility to recover funds in a contract that should not be in it) and are operated through EIP1967-based proxies.

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 behaviours 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			
Medium			
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:

- : Architectural shortcomings and design inefficiencies

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

-Severity Findings	0
-Severity Findings	0
-Severity Findings	0
-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.

-Severity Findings	0
-Severity Findings	0
-Severity Findings	0
-Severity Findings	3

- [Event Issues](#)
- [Irrational State Transition](#)
- [Outdated Compiler Version](#)

6.1 Event Issues

Important action does not emit event : An event could be added when token is paused in `SBCWrapper.pauseToken`

Misleading event : When `pauseToken` was called and a token is enabled and the token is later unpaused via `enableToken`, a `UpdateSwapRate` event is emitted which could be misleading (because the swap rate might not have changed at all but the token only unpaused). Two actions are possible : update rate or enable token, but only one is registered through an event.

Code corrected :

Event is emitted when a token is paused.

When a token is enabled or reenabled after being paused `TokenSwapEnabled` event is emitted. Every time the swap rate changes `SwapRateUpdated` event is emitted.

6.2 Irrational State Transition

The wrapper contract admin is allowed to call `pauseToken` on tokens that were never enabled. This state transition seems unintended.

Code corrected :

Only an `ENABLED` token can be paused.

6.3 Outdated Compiler Version

In the code the following pragma directives are used:

```
pragma solidity 0.8.7;
```

Known bugs in version 0.8.7 are:

https://github.com/ethereum/solidity/blob/develop/docs/bugs_by_version.json#L1580

More information about these bugs can be found here:

<https://docs.soliditylang.org/en/latest/bugs.html>

At the time of writing the most recent Solidity release is version 0.8.9 which contains some bugfixes.

Code corrected :

Compiler version has been updated to 0.8.9.