Audit Report

Snowbridge Updates 2

v1.0

August 10, 2024

Table of Contents

License

THIS WORK IS LICENSED UNDER A CREATIVE COMMONS ATTRIBUTION-NODERIVATIVES 4.0 INTERNATIONAL LICENSE.

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

COPYRIGHT OF THIS REPORT REMAINS WITH THE AUTHOR.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Snowfork to perform a security audit of updates to Snowbridge.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/paritytech/polkadot-sdk
Commit	`5d9826c2620aff205811edf0e6a07b55a52cbf50`
Scope	The scope of this audit was restricted to the changes in the commit stated above, which were introduced in pull request https://github.com/paritytech/polkadot-sdk/pull/3761.
Fixes verified at commit	`94147079c3d5a8d4e2334b346eb29850ec21e917` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Snowbridge is a general-purpose, trustless, and decentralized bridge between Polkadot and Ethereum. This is achieved by using light clients. The protocol uses a BEEFY light client implemented in Solidity smart contracts to track the Polkadot chain, and an Altair-compliant light client to keep track of the Ethereum Beacon Chain implemented in a Substrate pallet.

The audit scope is restricted to the modifications introduced in the https://github.com/paritytech/polkadot-sdk/pull/3761 pull request.

Previously, execution headers were required to be imported into the pallet and each received message was verified against them, which was inefficient. This update allows execution headers to be sent contextually with the message, eliminating this inefficiency. This improvement was facilitated by the Deneb Ethereum upgrade, which made it easier to locate the relevant beacon header from an execution header.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	**Description**
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending, Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	**Status**	**Comment**
Code complexity	**High**	The code implements complex operations and makes use of the latest features coming from Substrate and Cumulus. It also uses the latest XCM specification. The bridge uses/integrates with low-level functionality from different ecosystems.
Code readability and clarity	**Medium**	-
Level of documentation	**High**	The protocol and the modifications applied are well documented.
Test coverage	**Medium**	Test coverage for `pallets`, `primitives`, and `runtime-common` reported by `cargo tarpaulin` is `75.70%`.

Summary of Findings

No	Description	Severity	Status
1	Inefficiency in `verify_execution_proof` function	Informational	Resolved
2	Incorrect error message	Informational	Acknowledged
3	Outdated comment	Informational	Resolved

Detailed Findings

1. Inefficiency in `verify_execution_proof` function

Severity: Informational

In the `ethereum-client` pallet, the function `verify_execution_proof` is defined in `bridges/snowbridge/parachain/pallets/ethereum-client/src/impls.rs:73-129`.

During the refactoring, the data flow of the function slightly changed compared to the older definition of the `process_execution_header_update` function located in `bridges/snowbridge/parachain/pallets/ethereum-client/src/lib.rs`. Specifically, `bridges/snowbridge/parachain/pallets/ethereum-client/src/impls.rs:116-129` handles a particular case of the verification when the ancestry proof is not provided.

Considering the present data flow, only `beacon_block_root` needs to be computed to perform this verification.

Consequently, a single hash tree root computation is required, and if the slot numbers do not match, the control flow can immediately return.

This optimization eliminates the need for one Merkle proof verification if the ancestry proof is missing and the slot numbers do not match.

Recommendation

We recommend restructuring the code of the `verify_execution_proof` function to improve its efficiency. This should be done by first computing the `beacon_block_root` and then checking for the presence of the ancestry proof and the equality of slot numbers, as outlined in lines `116-124`.

Status: Resolved

2. Incorrect error message

Severity: Informational

In the `ethereum-client` pallet, specifically in the `bridges/snowbridge/parachain/pallets/ethereum-client/src/impls.rs:143`, the `HeaderNotFinalized` error is triggered when `block_slot` is greater than or equal to `state.slot`.

However, when the slot numbers are equal, a different error message should be used.

Consequently, the error message is incorrect since the header is already finalized in this case.

Recommendation

We recommend ensuring that the case of equal slot numbers is rejected with the correct error message.

Status: Acknowledged

3. Outdated comment

Severity: Informational

In the `ethereum-client` pallet, `bridges/snowbridge/parachain/pallets/ethereum-client/src/impls.rs:14-17` contains commentary for the `verify` function, which states:

“The execution header containing the log should be in the beacon client storage.”

However, since the recent changes in the audited codebase remove storage for execution headers, the comment is misleading.

Recommendation

We recommend updating this comment.

Status: Resolved