Audit Report

Snowbridge Updates 3

v1.0

January 7, 2025

Table of Contents

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.

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	`87971b3e92721bdf10bf40b410eaae779d494ca0`

Scope	The scope was restricted to the changes compared to commit `5d9826c2620aff205811edf0e6a07b55a52cbf50` of the https://github.com/Snowfork/snowbridge repository for the following directories and files:
	
	`.`
	`└── bridges`
	`    └── snowbridge`

Fixes verified at commit	`5682f9a273309c7160722a7937a75152a2793b23` 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.

Repository	https://github.com/Snowfork/snowbridge
Commit	`581fb3af82cb350f5ba2ac4de295ce2778ba7236`

Scope	The scope was restricted to the changes compared to commit `10875e90cd1cf35cb039a382586f767ba19a8b9e` for the following directories and files:
	
	`.`
	`└── contracts`
	`    └── src`

Fixes verified at commit	`0c0e04618af1a23147e9af88078844d88d33ac8c` 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 scope of this audit is restricted to the updates and changes that have been implemented since the completion of our previous audit.

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**, **Partially Resolved**, 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-High**	Test coverage for Solidity contracts reported by `forge coverage` is `72.77%`. Test coverage for `pallets`, `primitives`, and `runtime-common` reported by `cargo tarpaulin` is `75.20%`.

Summary of Findings

No	Description	Severity	Status
1	Unauthorized minting of PNA assets due to missing origin validation	Major	Resolved
2	Potential asset lock on AssetHub due to failure in creating the wrapped token on Ethereum after token registration.	Minor	Acknowledged
3	`OPERATOR` cannot be updated	Minor	Acknowledged
4	Insufficient documentation for the `initialize` function in the Gateway contract	Informational	Resolved
5	Unused code should be removed	Informational	Resolved
6	Querying costs of sending a native asset is not exposed	Informational	Resolved
7	Code duplication decreases maintainability	Informational	Resolved
8	Distinct event should be used when a Polkadot native asset is sent	Informational	Resolved

Detailed Findings

1. Unauthorized minting of PNA assets due to missing origin validation

Severity: Major

In `bridges/snowbridge/primitives/router/src/outbound/mod.rs:109`, the `agent_id` is resolved within the message exporter, derived from the XCM origin.

However, while the resolved `agent_id` is available in the minting process as `self.agent_id`, it is not utilized in the creation of the mint message.

Instead, in `bridges/snowbridge/primitives/router/src/outbound/mod.rs:126` the `channel_id` is inferred directly from the parachain ID of the XCM origin. It is important to note that the channel must be already registered at this stage.

Then, in `bridges/snowbridge/primitives/core/src/outbound.rs262-267`, the mint message is dispatched without passing the `agent_id`, meaning the Solidity side lacks knowledge of the true origin location.

Upon dispatch, the Gateway contract processes the mint message in `Assets.sol`. While the `mint` function is restricted to the `onlyGateway` modifier, this restriction is insufficient. Any agent can trigger this code path and exploit Gateway Proxy permissions to execute the mint.

Consequently, this allows any parachain with a channel and agent created by governance to construct and dispatch a mint message to the Gateway contract.

The client has independently identified this issue.

Recommendation

We recommend providing the `agent_id` to the contract to correctly perform origin validation.

Status: Resolved

2. Potential asset lock on AssetHub due to failure in creating wrapped token on Ethereum after token registration

Severity: Minor

The `register_token` function, located in `polkadot-sdk:bridges/snowbridge/pallets/system/src/lib.rs:621`, allows

the `root` to register a new token using the provided `AssetMetadata`. It is assumed that this registration will be relayed and executed on the Ethereum side, leading to the creation of a corresponding wrapped token on the Ethereum blockchain.

However, there is an edge case where a token is successfully registered on Snowbridge, but the corresponding wrapped asset has not been created yet on Ethereum. Suppose a user initiates a transaction to send the newly registered token from AssetHub to Ethereum. Since the token is already registered on the Snowbridge side, the transaction will be processed. However, the corresponding minting of the wrapped asset on Ethereum would fail, either because the registration transaction was executed after the user's transfer transaction, or because the asset registration transaction was missed in the relayer queue.

As a result, the user's asset would remain locked until the relayers successfully re-execute the minting transaction following the proper registration of the token on Ethereum.

Recommendation

We recommend introducing a confirmation mechanism to ensure that token registration is only considered complete once the corresponding wrapped token has been successfully created on Ethereum.

Status: Acknowledged

The client states that the `registerForeignToken` Solidity code has minimal failure risk as it operates internally within the bridge. Failures, such as malformed messages, would be detected through integration tests or testnets and can be retried if transient. Users do not interact with Bridge Hub directly, so a confirmation mechanism is unnecessary.

3. `OPERATOR` cannot be updated

Severity: Minor

In `snowbridge:contracts/src/Shell.sol`, the `OPERATOR` role is declared, which can be used to perform an upgrade.

However, the bridge has no means to control or update it since the address used for this role is set only during deployment, in `contracts/scripts/Deploy.sol:15`.

If a malicious party compromises the private key controlling the `OPERATOR` account, they can upgrade the Ethereum side of the bridge to any contract, without a way to rotate the operator account. Even if a multi-party computation (MPC) or multi-signature account is used, there remains a risk of collusion or multiple key compromises.

Recommendation

We recommend allowing Polkadot governance to update the address of the `OPERATOR` account.

Status: Acknowledged

The client states that the Shell contract is solely used for initial trusted bootstrapping and is no longer necessary for production.

4. Insufficient documentation for the `initialize` function in the Gateway contract

Severity: Informational

The state changes introduced in the `Gateway` contract, specifically in `snowbridge:contracts/src/Gateway.sol:653` and `688-689`, are intended solely for use during the initial deployment of the contract.

However, this intention is not explicitly documented, leading to potential confusion and incorrect use by developers.

Recommendation

We recommend improving the documentation to explicitly state that the `initialize` function is designed only for initial deployment and not for use during contract upgrades.

Status: Resolved

5. Unused code should be removed

Severity: Informational

The code in the following locations is not used:

- In `snowbridge:contracts/src/Gateway.sol:114-121`, the modifier `onlyAgent` is defined. This modifier is not utilized anywhere in the codebase. Several other definitions are only used within this modifier.
- The function `_ensureAgentAddress`, defined in `snowbridge:contracts/src/Gateway.sol:586-592` is only called from the modifier `onlyAgent`, which is not used.
- The mapping `agentAddresses`, defined in `snowbridge:contracts/src/storage/CoreStorage.sol:16`, is updated

on lines `653` and `666`. However, the only query made to this mapping is located within the `onlyAgent`.

- Error messages `AgentExecutionFailed` and `AlreadyInitialized`, declared in lines `100` and `103` respectively, are not used.
- The `_isTokenRegistered` internal function from `snowbridge:contracts/src/Assets.sol:344` is not used.
- In `snowbridge:contracts/src/AgentExecutor.sol:10`, the `Gateway` is imported from `Gateway.sol`, but it is not used in the contract's logic.
- Error message `UnsupportedFeeAsset`, defined in `polkadot-sdk:ridges/snowbridge/primitives/router/src/inbound/mod.rs:135`, is not used.

Recommendation

We recommend removing unused code.

Status: Resolved

6. Querying costs of sending a native asset is not exposed

Severity: Informational

The `_sendTokenCosts` function is accessible via the `quoteSendTokenFee` function through `sendTokenCosts`.

However, there is no equivalent query function that utilizes the `_sendForeignTokenCosts` function, defined in `snowbridge:contracts/src/Assets.sol:197`.

This inconsistency limits the accessibility of `_sendForeignTokenCosts`, preventing it from being queried in the same manner as `_sendTokenCosts`.

Recommendation

We recommend providing users with the ability to query the costs associated with sending a native asset.

Status: Resolved

7. Code duplication decreases maintainability

Severity: Informational

The `_sendForeignTokenCosts` function is an exact duplicate of the `_sendTokenCosts` function. This duplication complicates maintenance and increases the likelihood of errors during updates.

Handling Ethereum tokens and native assets separately, as done here, could be refactored to streamline the code, reduce redundancy, and make future code reviews and updates more efficient.

Recommendation

We recommend refactoring the logic for handling Ethereum tokens and native assets to eliminate duplicate code.

Status: Resolved

8. Distinct event should be used when a Polkadot native asset is sent

Severity: Informational

In `snowbridge:contracts/src/Assets.sol:195` and `snowbridge:contracts/src/Assets.sol:257`, the `IGateway.TokenSent` event is being emitted, signaling that assets have been sent cross-chain. One of the values communicated using this event is the address of the Ethereum token contract.

However, a separate event type should be declared and utilized for Polkadot native assets since their primary identifier is a 32-byte hash of their location in Polkadot, rather than an Ethereum address.

The location directly determines a Polkadot native asset and does not require knowledge of the `tokenAddressOf` mapping defined in `snowbridge:contracts/src/storage/AssetsStorage.sol:20`.

Recommendation

We recommend using a dedicated event to notify users about Polkadot native asset transfers.

Status: Resolved