Audit Report

Snowbridge Extension

v1.0

February 12, 2024

Table of Contents

Table of Contents **2**

License **3**

Disclaimer **3**

Introduction **5**

Purpose of This Report 5

Codebase Submitted for the Audit 5

Methodology 6

Functionality Overview 6

How to Read This Report **7**

Code Quality Criteria 8

Summary of Findings **9**

Detailed Findings **10**

1. Potential locked funds in case the user-defined destinationFee does not cover the actual fee 10

2. Unverified registerToken function execution outcome could lead to funds being stuck in the agent contract 11

3. Inconsistent state in case agent contract instantiation fails on Ethereum 11

4. Lack of cross-chain error handling could lead to inconsistent states, loss of funds, and frozen channels 12

5. The throttling mechanism could delay critical governance operations 13

6. The current DOT/ETH exchange rate implementation is not optimal in the context of price volatility 14

7. State updates for the primary governance channel are transmitted through the slower channel 15

8. Channel-specific outbound_fee is not fully implemented 16

9. The agent may not be able to reimburse the relayers 16

10. Inefficient overflow protection for the nonce 17

11. Events do not contain information about the channel in use 17

12. Comments do not reflect the implementation 18

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 IS ADDRESSED EXCLUSIVELY TO THE CLIENT. THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THE CLIENT OR THIRD PARTIES.

COPYRIGHT OF THIS REPORT REMAINS WITH THE AUTHOR.

This audit has been performed by

Oak Security

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

Introduction

Purpose of This Report

Oak Security has been engaged by Snowfork to perform a security audit of the Snowbridge EVM Contracts and Substrate Pallets.

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/snowfork/snowbridge
Commit	`1e27bce2c34989fd48f77d0ac8b6909ff09793c7`
Scope	Only the changes since our previous audit of the Solidity contracts in the `contracts` directory (previously in `core/packages/contracts`) and the Parachain code in the `parachain` directory excluding the `parachain/runtime` directory were in scope of this audit. Our previous audit has been performed at commit `fa632e665aa560a799f8396718ef81f31e26dc3d`.

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.

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. Solidity smart contracts use assembly and memory pointers.
Code readability and clarity	**Medium**	-
Level of documentation	**Medium**	The protocol is well documented. However, there is little documentation of integrations with third-party code.
Test coverage	**Medium**	Test coverage for Solidity contracts reported by `forge coverage` is `69.72%`. Test coverage for Substrate pallets reported by `cargo tarpaulin` is `74.20%`.

Summary of Findings

No	Description	Severity	Status
1	Potential locked funds in case the user-defined `destinationFee` does not cover the actual fee	Critical	Acknowledged
2	Unverified `registerToken` function execution outcome could lead to funds being stuck in the agent contract	Major	Acknowledged
3	Inconsistent state in case agent contract instantiation fails on Ethereum	Major	Acknowledged
4	Lack of cross-chain error handling could lead to inconsistent states, loss of funds, and frozen channels	Minor	Acknowledged
5	The throttling mechanism could delay critical governance operations	Minor	Acknowledged
6	The current DOT/ETH exchange rate implementation is not optimal in the context of price volatility	Minor	Acknowledged
7	The primary governance channel is updated through the secondary governance channel	Minor	Resolved
8	Channel-specific `outbound_fee` is not fully implemented	Minor	Resolved
9	The `agent` may not be able to reimburse the relayers	Minor	Acknowledged
10	Inefficient overflow protection for the `nonce`	Informational	Partially Resolved
11	Events do not contain information about the channel in use	Informational	Acknowledged
12	Comments do not reflect the implementation	Informational	Resolved

Detailed Findings

1. Potential locked funds in case the user-defined `destinationFee` does not cover the actual fee

Severity: Critical

In `contracts/src/Gateway.sol`, the `sendToken` function accepts `destinationFee` as an input, allowing users to set any arbitrary fee value.

This allows users to send a message with a `destinationFee` set to zero for executing the transaction as cheaply as possible, which does not cause errors on the Ethereum side.

On the Polkadot side though, this may lead to errors: When receiving the message, the `inbound-queue` pallet executes the `convert_send_token` function in `parachain/primitives/router/src/inbound/mod.rs:124-290`. During this conversion, in case `dest_para_id` is defined, the `dest_para_fee` is included in a `DepositReserveAsset` XCM instruction.

However, if `dest_para_fee` is zero or a value less than the required fee, the XCM instruction will fail because of the missing funds for the execution leading to funds being stuck in the agent.

Recommendation

We recommend enforcing a minimum `destinationFee` in the `sendToken` function that covers the `dest_para_fee`.

Status: Acknowledged

The client acknowledges the risks associated with users who directly engage with the low-level Solidity APIs, particularly the potential for insufficient `destinationFee` leading to an unrecoverable loss of funds. They emphasize that their design does not intend for users to directly use the `sendToken` API.

To mitigate these risks, the client has implemented an off-chain UX layer that serves as an intermediary, ensuring the accurate calculation and retrieval of the correct XCM fees for all destination parachains. This proactive approach aims to streamline user interactions and enhance the security and reliability of transactions.

Long term, the client is committed to introducing a mechanism that enables the recovery of funds in scenarios where messages fail to execute at their intended destination. This solution will leverage the asset claims feature of XCMv3.

2. Unverified registerToken function execution outcome could lead to funds being stuck in the agent contract

Severity: Major

In contracts/src/Assets.sol:154-172, the registerToken function sets the isRegistered flag to indicate the token registration status on the Asset Hub.

However, this flag is set regardless of the success or failure of the token registration command, which could fail for instance if the token is already registered and the Asset Hub's create extrinsic fails in https://crates.parity.io/src/pallet_assets/lib.rs.html#385.

Consequently, if the token registration fails or is delayed, every subsequent sendToken function invocation will fail as well on the Polkadot side, leading to user tokens being stuck in the agent contract.

Recommendation

We recommend checking the outcome of the token registration process before enabling users to transfer tokens.

Status: Acknowledged

The client acknowledges the risks associated with users who directly engage with the low-level Solidity APIs.

To mitigate these risks, the client has implemented an off-chain UX layer that serves as an intermediary, which is designed to conduct multiple cross-chain integrity checks prior to authorizing users to transfer tokens via the bridge. This approach ensures an added layer of security and reliability in the transaction process. Additionally, it offers users the opportunity to initiate token registration for tokens that have not been previously registered on the platform.

Moreover, the client plans to register the top 100 most popular tokens proactively. This registration will be completed before granting users access to utilize the bridge.

3. Inconsistent state in case agent contract instantiation fails on Ethereum

Severity: Major

In parachain/pallets/system/src/lib.rs:373-394, the create_agent extrinsic updates the Agents map and then sends a message to Ethereum.

However, since the message dispatch and execution is async, but the Agents map is already updated, the pallet will act as if the agent contract is instantiated already, even if the instantiation message has not yet been processed.

Additionally, in case the message fails in the Ethereum Gateway contract, the state on the Polkadot side will not be reverted, leading to an inconsistent state between the two sides of the bridge.

Recommendation

We recommend checking that the Agent contract has been created on Ethereum before storing it in the `Agents` map.

Status: Acknowledged

The client informs that the `CreateAgent` API will be disabled at the launch of their protocol. This decision aligns with their planning, as the API is intended for future functionalities they aim to incorporate, such as the general-purpose XCM Transact. They emphasize that their current token bridge feature does not depend on this API. Instead, it requires agents only for the BridgeHub and AssetHub parachains, which are set up manually through their deployment scripts.

Furthermore, the client addresses the potential reasons for the failure of the `CreateAgent` operation on Ethereum, highlighting that due to the bridge's design, such instances are expected to be rare. Possible causes of failure include:

1. The occurrence of an exceptional flaw that remained undetected during their comprehensive testing and audit processes.
2. A significant update to Ethereum that leads to more than a 25% increase in gas costs for operations like contract instantiation, coupled with a scenario where their contracts have not been updated to align with these changes.

4. Lack of cross-chain error handling could lead to inconsistent states, loss of funds, and frozen channels

Severity: Minor

In `parachain/pallets/inbound-queue/src/lib.rs`, potential errors may arise during the processing of inbound messages, specifically on lines `262-263`, `268`, and `278`.

These errors can be related to issues in the inbound message structure, XCM conversion, insufficient balance to cover processing fees, or problems with XCM delivery.

Consequently, in case an error is raised on the Polkadot side, the Ethereum contract's state is already mutated. Failure to handle the error and rollback changes on the Ethereum side for a valid inbound message on the Polkadot side may lead to inconsistent states between the two sides of the bridges.

Furthermore, since channels are ordered, a failure in message handling will revert the transaction and will not update the next `nonce`. As a result, only the erroring message can be proposed, the channel will be stuck and no other message will be able to be executed through it.

Recommendation

We recommend implementing packet acknowledgment and error-handling mechanisms to let the sending chain react to possible failures similarly to how they are implemented in the IBC protocol.

Status: Acknowledged

The client states: “We acknowledge that our bridge has not been designed to include IBC-inspired cross-chain packet acknowledgment and error handling (such as reverting state changes). We believe that these features would significantly increase the complexity, operating costs, and latency of the bridge. As such, we do not think cross-chain packet acknowledgment is a better design for our specific bridging needs, and will not be implementing it.

“Reverting state changes at the source when an error happens remotely would also not compose well with XCM, which is Polkadot's native cross-chain messaging standard.

“The auditors have listed specific concerns about lines `262-263`, `268`, and `278` that could result in errors. Here is our analysis:

“Lines `262-263`: If a message conversion error occurs here, it means our Gateway contract on Ethereum is producing messages in an unexpected format. This is considered a highly exceptional failure, and we have extensive unit tests and automated cross-chain integration tests that cover this path. In the highly unlikely event that an error occurs here, it can be remedied with a runtime upgrade to our parachain, and the message submission can be retried, resulting in the channel being unblocked.

“Line `268`: Errors here are expected during the normal operation of the bridge, and signify that the channel owner needs to top up their sovereign account used to reward relayers. Message submission can always be retried later on.

“Line `278`: Errors here are expected during the normal operation of the bridge, which means that there is an underlying transient issue with the HRMP channel to the final destination. This would usually be congestion. Message submission can be retried after the HRMP congestion resolves.”

5. The throttling mechanism could delay critical governance operations

Severity: Minor

In `parachain/pallets/system/src/lib.rs`, a throttling mechanism for the queue is implemented to limit the execution complexity of the `on_finalize` function.

This throttling mechanism limits the number of messages retrieved per block from the `outbound-queue` pallet to a maximum of `MaxMessagesPerBlock` messages.

Consequently, the processing of messages could be delayed depending on the number of messages in the queue of the `outbound-queue` pallet, but also depending on the number of other pallets interacting with their queues since the `message-queue` pallet selects autonomously each block which client pallet is served and there is no guarantee that the `outbound-queue` is selected.

This could potentially cause delays in the execution of important commands like `SetOperatingMode` or `Upgrade` that should be dispatched promptly to address malfunctioning or exploits of the bridge.

For instance, an attacker could exploit this by enqueuing a substantial number of commands to saturate the queue.

The implemented segregation of governance channels does not prevent this issue since all the channels rely on the same queue.

Recommendation

We recommend prioritizing commands crucial for the protocol, particularly those executed by governance. Achieving this could entail appending these specific commands directly into the `Messages` and `MessageLeaves`, rather than enqueuing them.

Status: Acknowledged

The client states that in a worst-case scenario, with 100 queues, governance command delays could be 10 minutes. This delay is minor in the overall governance process making the impact of throttling relatively insignificant.

6. The current DOT/ETH exchange rate implementation is not optimal in the context of price volatility

Severity: Minor

The current Snowbridge implementation introduces a parameter responsible for storing the exchange rate of the DOT/ETH pair to calculate a fair transaction fee.

Its value is updated manually via `set_pricing_parameters` extrinsic. This is suboptimal and susceptible to irregularities in the form of large deviations from the actual ETH or DOT price and the stored exchange rate.

According to the comment in `parachain/pallets/outbound-queue/src/lib.rs:53`, this rate will be updated once every few weeks. Given historical price fluctuations in short periods of time between

DOT and ETH, it is possible that forwarding transactions will be unprofitable for relayers or for the protocol itself.

Additionally, since the fees are collected but not directly swapped using the provided exchange rate, the value of fee reserves deposited by users is susceptible to market volatility.

Recommendation

We recommend adapting the logic to update the exchange rate between DOT and ETH automatically to increase the submission frequency.

Another solution can be using a trusted oracle, which could be queried for the latest exchange rate.

We also recommend swapping fees to reduce the impact of market volatility on the operation of the bridge.

Status: Acknowledged

The client indicates that the current operational solution with the existing code involves charging users higher fees, accomplished through a specially calculated exchange rate. This approach aims to guarantee that relayers receive appropriate rewards even in the presence of volatility.

Looking ahead, there are long-term plans to launch a decentralized exchange (DEX) on AssetHub within the Polkadot network. In this scenario, BridgeHub would have the capability to utilize a dynamically updated ETH/DOT price feed. However, this should be thoughtfully evaluated, as DEX price feeds come with their own set of challenges.

7. State updates for the primary governance channel are transmitted through the slower channel

Severity: Minor

In `parachain/pallets/system/src/lib.rs:489-490`, the Primary Governance Channel handles the update of pricing parameters, contract upgrades, and the pausing/resuming of the Gateway. The Secondary Governance Channel, instead, is responsible for creating agents, creating channels, native forced transfers from agents, and forced pausing/resuming of the channel.

Forceful channel updates, which include actions such as pausing or resuming a channel and adjusting usage fees, should be directed to the Primary Governance Channel, but are currently managed through the Secondary Governance Channel.

As a result, if the Secondary Governance Channel becomes more crowded, the governance's response to any issues could become slower.

Recommendation

We recommend prioritizing updates for the primary governance channel over those for other channels by transporting them via the primary governance channel itself.

Status: Resolved

8. Channel-specific `outbound_fee` is not fully implemented

Severity: Minor

In `parachain/pallets/system/src/lib.rs:441-465` and `479-495`, the `update_channel` and `force_update_channel` extrinsics permit to modify channels by specifying the `mode` and the `outbound_fee`.

However, the `outbound_fee` is currently disregarded and not processed either in the pallet or within the Gateway contract – it is solely emitted in the event.

Despite being defined in the comment as the “Fee charged to users for sending outbound messages to Polkadot”, this specific fee per channel is not currently implemented.

Recommendation

We recommend implementing the use of the `outbound_fee`.

Status: Resolved

9. The `agent` may not be able to reimburse the relayers

Severity: Minor

In `contracts/src/Gateway.sol:217-222`, a mechanism is implemented to incentivize relayers to forward messages and receive a reward from the `agent` in return.

However, since there is no automated mechanism to fund agents, such as an explicit requirement for agent creators to fund the `agent` contract or an enforced fee ratio favorable for the protocol, it is possible that due to market conditions, bridge usage, or after bootstrapping the contract, there are no funds left to reward relayers.

Consequently, there would be no incentive for relayers to forward messages, which could cause delays and disruptions in the bridge operations.

Recommendation

We recommend implementing an automated mechanism to replenish funds in the agents, as the current process relies on manual intervention.

Status: Acknowledged

10. Inefficient overflow protection for the `nonce`

Severity: Informational

In `parachain/pallets/outbound-queue/src/lib.rs:313`, the `nonce` is correctly incremented using saturated addition, safeguarding against overflow. However, when the `nonce` reaches its maximum value, messages with that one can still be sent across the bridge. Only the initial message with the maximum `nonce` will succeed; subsequent ones will be rejected.

The contract `contracts/src/Gateway.sol:142-500` increments a channel's `nonce` without protection against overflow. This results in sending a message which will be rejected on the Polkadot side.

While there is no security implication, redundant message transmissions lead to inefficiencies that could be avoided.

Recommendation

We recommend rejecting further messages if the `nonce` overflows.

Status: Partially Resolved

11. Events do not contain information about the channel in use

Severity: Informational

In `parachain/pallets/system/src/lib.rs`, control commands are dispatched over channels to Ethereum using the `send` function, with corresponding events emitted through `deposit_event`.

For example, in line `645`, the `TransferNativeFromAgent` command is sent over a channel, and the corresponding event is emitted in line `647`.

However, events do not provide information about the specific channel used for each command even if some commands, for instance, the `TransferNativeFromAgent`, could be sent via either a parachain channel or a governance one leading to possible issues in off-chain indexers that listen to those events.

Recommendation

We recommended enhancing event emissions by including information about the channel in use.

Status: Acknowledged

12. Comments do not reflect the implementation

Severity: Informational

It has been noticed that in some places comments do not match the implementation. This may lead to problems with understanding the logic by developers and negatively impact maintainability.

Noticed irregularities:

- The comment in `parachain/pallets/outbound-queue/src/lib.rs:54` states that the exchange rate between DOT and ETH can be changed via `set_fee_parameters` extrinsic. However, such a function does not exist. In practice, the `set_pricing_parameters` extrinsic is responsible for this.
- In the `system` pallet, functional requirements and information about fees are specified in the comments for most extrinsics while no fees are charged for governance ones. For the `update_channel` and `transfer_native_from_agent` administrative functions, the so-called `Partial` fee is charged. However, the comments describing them indicate that no fee is included, which differs from the implementation.

Recommendation

We suggest correcting these comments to align with the implementation.

Status: Resolved