Security Audit Report

Snowbridge v2

v1.0

May 29, 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 GmbH has been engaged by Snowfork to perform a security audit of Snowbridge v2.

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
Scope	The scope is restricted to the following folders: • `bridges/snowbridge/pallets/inbound-queue-v2/src` • `bridges/snowbridge/pallets/outbound-queue-v2/src` • `bridges/snowbridge/pallets/outbound-queue-v2/runtime-api/src/` • `bridges/snowbridge/pallets/system-frontend/src`

	• `bridges/snowbridge/pallets/system-v2/src` • `bridges/snowbridge/primitives/core/src` • `bridges/snowbridge/primitives/inbound-queue/src/lib.rs` • `bridges/snowbridge/primitives/inbound-queue/src/v2` • `bridges/snowbridge/primitives/outbound-queue/src/lib.rs` • `bridges/snowbridge/primitives/outbound-queue/src/v2` • `bridges/snowbridge/primitives/verification/src/lib.rs` of the changes applied in the https://github.com/paritytech/polkadot-sdk/pull/7402 pull request reviewed at commit `05afd0253c70967950439c299c0bf9f7acd2623f`.
Fixes verified at commit	`38f1aee87918101c60db11b011b7d937b19bf7cb` 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
Scope	The scope is restricted to the following folders: • `contracts/src` of the changes applied in the https://github.com/Snowfork/snowbridge/pull/1371 pull request reviewed at commit `6b3bb32cb6d549ae410936e74ba9eb98b39da855`.
Fixes verified at commit	`9bcac977155be1be0fc55e2d16ac3244667a50f4` 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.

Snowbridge v2 allows users to bridge tokens and relay instructions between Ethereum and Polkadot/Kusama parachains.

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 of the Polkadot SDK. 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-High**	The protocol is well documented.
Test coverage	**Medium**	Test coverage for Solidity contracts reported by `forge coverage` is `81.45%`. Test coverage for Substrate pallets reported by `cargo tarpaulin` is `60.81%`.

Summary of Findings

No	Description	Severity	Status
1	Digest item validation flaw enable inconsistent commitment verification	Major	Resolved
2	State collision after upgrade of the Gateway contract	Major	Resolved
3	Incorrect decoding causes lack of incentives for relayers	Major	Resolved
4	Reward misallocation due to ignored `reward_address` field	Major	Resolved
5	Lack of atomicity in `v2_dispatch` could lead to state inconsistencies	Major	Partially Resolved
6	Missing input validation in `sendMessage`	Minor	Acknowledged
7	Token address is not validated during registration	Minor	Resolved
8	Unbounded fee charging in XCM execution	Minor	Acknowledged
9	Unrestricted XCM instructions	Minor	Acknowledged
10	Failure to advance nonce on error allows repeated execution of failed events without relayer incentives	Minor	Acknowledged
11	Unrestricted empty message submission	Informational	Acknowledged
12	Redundant checks in `registerToken` function	Informational	Resolved
13	Non-sequential call indexes in `systemv2` pallet	Informational	Resolved
14	Incorrect NatSpec in `register_token` function	Informational	Resolved
15	Redundant operating mode updates	Informational	Acknowledged
16	Missing automatic `RefundSurplus` instruction	Informational	Acknowledged
17	XCM payloads using unlimited weight	Informational	Acknowledged
18	Potentially confusing message hashing implementation	Informational	Resolved
19	`SparseBitmap` can be optimized	Informational	Resolved
20	Edge case can cause infinite loop under particular	Informational	Resolved

	network conditions		
21	Use of assembly `invalid` instruction obscures error context and reduces maintainability	**Informational**	**Resolved**
22	Misleading implementation details	**Informational**	**Partially Resolved**
23	Inefficiency in rejecting messages	**Informational**	**Resolved**
24	Unused code	**Informational**	**Partially Resolved**

Detailed Findings

1. Digest item validation flaw enables inconsistent commitment verification

Severity: Major

In `contracts/src/Verification.sol:139-163`, the `isCommitmentInHeaderDigest` function verifies the presence of a message commitment within a header digest. The protocol uses distinct digest items depending on the protocol version: `DIGEST_ITEM_OTHER_SNOWBRIDGE` for version `1` and `DIGEST_ITEM_OTHER_SNOWBRIDGE_V2` for version `2`.

However, the implementation fails to correctly enforce version-specific digest item checks. Specifically, the initial condition in the function does not verify the `isV2` flag, resulting in acceptance of a version `1` digest item even during a version `2` protocol flow. If `isV2` is `true` but the digest item corresponds to `DIGEST_ITEM_OTHER_SNOWBRIDGE` (version `1`), the function prematurely returns `true` without evaluating the condition intended for version `2`.

Consequently, this logic flaw enables the acceptance of commitments unrelated to the current version flow.

Recommendation

We recommend refactoring the conditional logic in the `isCommitmentInHeaderDigest` function to ensure that the digest item type is validated in strict conjunction with the protocol version flag.

Status: Resolved

2. State collision after upgrade of the Gateway contract

Severity: Major

In `contracts/src/storage/CoreStorage.sol`, the `Layout` structure has been modified to support the updated V2 interface of the system.

Specifically, the legacy `agentAddresses` mapping in line `16` was removed, and two new fields, `inboundNonce` and `outboundNonce`, were introduced. These new fields are intended exclusively for use with the V2 interface. The `Layout` structure defines the persistent state of the Gateway contract, which is expected to undergo an upgrade while preserving its existing state.

However, the modification introduces a state collision: the new `inboundNonce` mapping is allocated at the same storage slot previously occupied by the `agentAddresses` mapping.

As a result, immediately after the contract upgrade, the `inboundNonce` mapping is unintentionally initialized with residual data from the deprecated `agentAddresses` field. From the perspective of the V2 interface, this data is invalid and leads to erroneous behavior. Specifically, valid inbound messages may be incorrectly identified as already processed due to misinterpreted nonces, causing legitimate transactions to be rejected.

Recommendation

We recommend carefully redesigning the storage layout to avoid overlap between legacy and new fields. Options include:

- Allocating new, uncontested storage slots for the `inboundNonce` and `outboundNonce` mappings.
- Keeping the older field `agentAddresses` but ensuring that it is not used in the V2 interface.

Status: Resolved

3. Incorrect decoding causes lack of incentives for relayers

Severity: Major

In `bridges/snowbridge/primitives/outbound-queue/src/v2/delivery_receipt.rs:45`, the `InboundMessageDispatched` event is decoded from raw Ethereum logs using the `decode_raw_log` function. This decoder is auto-generated via the `sol!` macro and is configured to expect three fields: `nonce`, `success`, and `reward_address`.

However, the corresponding event definition in `contracts/src/v2/IGateway.sol` includes four fields: `nonce`, `topic`, `success`, and an additional `bytes32` field.

Due to this discrepancy, the `topic` field is not accounted for during the decoding process, resulting in decoding failures for delivery receipts. When decoding fails, the relayer responsible for delivering the message does not receive the intended reward for their service.

The lack of relayer incentives introduces a mild denial-of-service condition. Without financial motivation, relayers are unlikely to continue processing messages, potentially stalling cross-chain communications. While the issue does not escalate to a critical severity, since technically capable users could still operate relayers at their own expense to process their personal transfers, it nonetheless undermines the expected incentive model and degrades the reliability of the system.

Recommendation

We recommend correcting the definition of `InboundMessageDispatched` event.

Status: Resolved

4. Reward misallocation due to ignored `reward_address` field

Severity: Major

In `bridges/snowbridge/pallets/outbound-queue-v2/src/lib.rs:374`, the `process_delivery_receipt` function processes a `DeliveryReceipt` parameter.

However, while the function correctly validates the gateway and nonce fields, it disregards the `reward_address` field contained within the `DeliveryReceipt` structure.

As a result, delivery rewards are always assigned to the transaction sender rather than to the beneficiary specified by the `reward_address`.

Recommendation

We recommend validating and utilizing all fields of the `DeliveryReceipt` structure to correctly allocate rewards.

Status: Resolved

5. Lack of atomicity in `v2_dispatch` could lead to state inconsistencies

Severity: Major

In `contracts/src/Gateway.sol:515`, the `v2_dispatch` function processes and executes all commands provided within a message. Each command is intended to operate independently.

However, the current implementation does not ensure atomic execution. If a single command within the batch fails, subsequent commands are not executed, while prior successful commands are not reverted. This behavior contradicts the expectation of atomic or isolated processing common in transactional systems with independent commands.

Such partial execution introduces the risk of inconsistent system states. For instance, consider a scenario where a message contains two commands: `CallContract` followed by `MintForeignToken`. If the contract call fails or runs out of gas, the minting operation is skipped entirely, resulting in loss of funds.

Although this could be attributed to user error in command ordering, external factors (such as contract state changes) may cause sequences that were initially valid in simulation to fail in live execution, leading to unpredictable behavior and incomplete operations.

Recommendation

We recommend enforcing atomic execution within the `v2_dispatch` function. If any command in the sequence fails, all prior state changes should be reverted to maintain system integrity.

Alternatively, implement isolated execution semantics where each command's success or failure does not impact others, and the system properly reports partial completions to prevent user confusion.

Status: Partially Resolved

6. Missing input validation in `sendMessage`

Severity: Minor

In `contracts/src/v2/Calls.sol:92`, the `_sendMessage` function constructs the `Payload` for the `IGatewayV2.OutboundMessageAccepted` event. The `claimer` parameter within this payload is defined as bytes memory and is utilized on the Polkadot side to identify the recipient for fund retrieval.

However, the function does not implement any validation for this parameter. As a result, in case of incorrect or misplaced data, it could lead to issues in funds retrieval.

Recommendation

We recommend validating on the `claimer` parameter.

Status: Acknowledged

7. Token address is not validated during registration

Severity: Minor

In `contracts/src/Functions.sol:96`, the `registerNativeToken` function is responsible for registering new ERC-20 tokens. This function is invoked by the V2 `registerToken` function, defined in `contracts/src/v2/Calls.sol`.

By contrast, the legacy V1 `registerToken` function, located in `contracts/src/v1/Calls.sol`, includes an additional validation step: it verifies the token address using the `isContract` predicate and reverts with an `InvalidToken` error if the provided address is not a valid contract.

However, this validation is absent in the V2 implementation.

As a result, the V2 registerToken function permits the registration of any address, including externally owned accounts (EOAs) and invalid addresses, as native assets.

Recommendation

We recommend validating the token addresses early, during their registration.

Status: Resolved

8. Unbounded fee charging in XCM execution

Severity: Minor

In `bridges/snowbridge/pallets/inbound-queue-v2/src/lib.rs:288`, within the `send_xcm` function, the relayer is charged using `T::XcmExecutor::charge_fees`.

However, there's no cap on the maximum amount the relayer is willing to pay. This could result in the relayer paying more than anticipated, especially under particular network conditions.

Recommendation

We recommend introducing a `max_fee_amount` parameter that defines the upper limit of acceptable fees, allowing relayers to cap the amount they are willing to pay for message delivery.

Status: Acknowledged

9. Unrestricted XCM instructions

Severity: Minor

In `bridges/snowbridge/primitives/inbound-queue/src/v2/converter.rs:365`, the contents of `message.remote_xcm` are directly appended to the final XCM payload without any form of validation or restriction on the instruction types. This design choice permits users to submit arbitrary XCM instructions as part of their message.

Permitting unrestricted instruction injection significantly broadens the protocol's attack surface and introduces various risks. For instance, malicious instructions could be crafted to manipulate the execution flow, such as suppressing or obscuring errors that occur within the protocol layer of the XCM process.

Recommendation

We recommend introducing an allowlist of permitted instruction types to limit what users can include in their `remote_xcm`.

Status: Acknowledged

10. Failure to advance nonce on error allows repeated execution of failed events without relayer incentives

Severity: Minor

In `bridges/snowbridge/pallets/inbound-queue-v2/src/lib.rs:216`, the `submit` extrinsic allows submission of inbound events originating from Ethereum. These events are decoded into messages and processed by the `process_message` function.

However, if the `send_xcm` operation fails, the transaction reverts without storing the nonce or registering rewards. Consequently, the same inbound message remains unmarked and eligible for re-execution.

This creates multiple risks: the same message can be replayed multiple times, potentially in varying orders and at a specific timing decided by the relayer under different conditions, potentially causing unintended state transitions.

Additionally, since rewards are only registered upon successful execution, relayers are incentivized to reorder messages to maximize successful transactions, potentially at the expense of the user and system fairness.

Recommendation

We recommend storing the nonce and reward relayers also in case the `send_xcm` operation fails.

Status: Acknowledged

11. Unrestricted empty message submission

Severity: Informational

In `contracts/src/v2/Calls.sol:61`, the `sendMessage` function enables users to submit messages containing various parameters, including fees, asset lists, and XCM payloads.

However, the function currently lacks input validation to prevent the submission of empty or no-op messages. Specifically, it allows:

- No fee
- No assets
- An empty XCM payload

While this behavior does not constitute a direct security vulnerability, it facilitates the transmission of messages that have no operational effect. These no-op messages

unnecessarily consume processing resources and gas, and contribute to infrastructure clutter across the bridge system.

Recommendation

We recommend introducing input validation to ensure that submitted messages contain at least one meaningful component e.g., a non-zero fee, at least one asset, or a non-empty XCM instruction.

Status: Acknowledged

12. Redundant checks in `registerToken` function

Severity: Informational

In `contracts/src/v2/Calls.sol:72`, the `registerToken` function includes a validation check on `msg.value` to ensure the correct fee is provided.

However, this check is already performed within the internal `_sendMessage` function at `contracts/src/v2/Calls.sol:92`, which is called by `registerToken`.

This results in redundant validation logic, adding unnecessary code duplication without providing additional security or functional benefits.

Recommendation

We recommend removing the redundant check from `registerToken` and rely on the validation within `_sendMessage` to keep logic centralized and reduce code duplication.

Status: Resolved

13. Non-sequential call indexes in `systemv2` pallet

Severity: Informational

In `bridges/snowbridge/pallets/system-v2/src/lib.rs`, the call indexes for extrinsics are inconsistently assigned.

Specifically, while indexes `0`, `3`, and `4` are implemented, indexes `1` and `2` remain undefined.

Although this does not currently pose a functional or security risk, the non-sequential indexing undermines the clarity and consistency of the codebase. In the context of a newly developed pallet, maintaining orderly call indexes aids in readability, developer experience, and future maintainability.

Recommendation

We recommend assigning sequential call indexes to all implemented extrinsics within the `systemv2` pallet.

Status: Resolved

14. Incorrect NatSpec in `register_token` function

Severity: Informational

In `bridges/snowbridge/pallets/system-v2/src/lib.rs:182`, the `register_token` function is documented to include a fee parameter, but no such parameter is actually used in the function.

Recommendation

We recommend updating the NatSpec documentation to accurately reflect the function parameters and behavior, removing any reference to a non-existent fee.

Status: Resolved

15. Redundant operating mode updates

Severity: Informational

In `bridges/snowbridge/pallets/inbound-queue-v2/src/lib.rs:230`, the `set_operating_mode` function allows setting the operating mode to its current value. This leads to unnecessary writes and events.

Recommendation

We recommend introducing a check to ensure the new mode is different from the current one before updating it.

Status: Acknowledged

16. Missing automatic `RefundSurplus` instruction

Severity: Informational

In `bridges/snowbridge/primitives/inbound-queue/src/v2/converter.rs:340`, the `PayFees` instruction is appended manually, but there is no automatic addition of a `RefundSurplus` instruction. This introduces an implicit assumption that the user includes

`RefundSurplus` as part of their custom `remote_xcm`. If omitted, leftover assets from overpaid fees may not be refunded.

Recommendation

We recommend appending `RefundSurplus` automatically at the end of the XCM message to ensure unused assets are consistently returned, regardless of user-provided instructions.

Status: Acknowledged

17. XCM payloads using unlimited weight

Severity: Informational

It was observed that the XCM built in `bridges/snowbridge/pallets/system-frontend/src/lib.rs:254-265` uses an `UnpaidExecution` as an execution variant and sets its weight limit to `Unlimited`.

Although this message is built in a call coming from a trusted origin of `RegisterTokenOrigin`, it is considered a good practice to provide an actual weight limit.

Recommendation

We recommend estimating the weight needed for XCM associated with token registration. Then, based on that value, the actual weight limit should be provided.

Status: Acknowledged

18. Potentially confusing message hashing implementation

Severity: Informational

In `bridges/snowbridge/pallets/system-v2/src/lib.rs:230-235`, the `Message` struct includes an `id` field, which is intended to store the `blake2_256` hash of the message itself.

However, during the hash computation, the `id` field is first initialized with its default value. The hash is then calculated over the struct containing this default `id`, and only afterward is the computed hash assigned back to the id field.

This approach, while functional, introduces complexity for external systems or tools attempting to verify the message. Verifiers must replicate this specific sequence: extract the `id` value from the message, reset the `id` field in the struct to its default value, compute the hash, and then compare it against the originally extracted `id`.

Recommendation

We recommend implementing a wrapper data structure over the `Message` that would contain its hash as a separate member, not part of the `Message` itself.

Status: Resolved

19. `SparseBitmap` can be optimized

Severity: Informational

In `bridges/snowbridge/primitives/core/src/sparse_bitmap.rs:9`, the mapping from buckets to masks is declared, used to define the `SparseBitmap` structure. The type of keys in this mapping is `u128`.

Then, within lines `22-27`, the function `compute_bucket_and_mask` is defined, which accepts parameter index of type `u128`. This function is called both from get and set functions, which are called only with nonces of type `u64`. The type of the parameter index can be replaced by `u64`.

The function `compute_bucket_and_mask` computes the index of the corresponding bucket by dividing the index by `128`, so the value range of the output is less than the value range of the input. Hence, the return type of the function, as well as the type of keys in the mapping declared on line `9`, can be declared as `u64`.

Similarly, in `contracts/src/utils/SparseBitmap.sol:4-5` the mapping data is defined with both keys and values of type `uint256`. Exactly the same reasoning as with the Polkadot counterpart applies in the Solidity code: nonces of type `uint64` are tracked using the structure and type of buckets in the `SparseBitmap` can be declared as `uint64` instead of `uint256`.

Recommendation

We recommend adopting `u64` and `uint64` as the types of keys in the internal mappings of both Rust and Solidity implementations of `SparseBitmap`.

Status: Resolved

20. Edge case can cause infinite loop under particular network conditions

Severity: Informational

In `contracts/src/Bitfield.sol:130-140`, the function `makeIndex` does not handle the case when `length` is `0`. In this case, the return value causes an infinite loop in the function `subsample`, which calls `makeIndex`.

This must be a rare situation since the validator set must be empty. However, this could be important when deploying the bridge to other EVM-compatible networks.

Recommendation

We recommend handling the case of `length` equal to `0`.

Status: Resolved

21. Use of assembly `invalid` instruction obscures error context and reduces maintainability

Severity: Informational

In `contracts/src/Gateway.sol:519-521`, the `invalid` assembly instruction is used to terminate execution.

While this achieves the intended low-level control flow effect, it does so without providing any descriptive error message or context.

As a result, when this instruction is triggered, debugging and maintenance become more challenging, as developers and integrators receive no meaningful information about the cause of the failure.

Recommendation

We recommend using named errors like `NotEnoughGas`.

Status: Resolved

22. Misleading implementation details

Severity: Informational

Throughout the codebase, various commentaries and in-line documentation are outdated or misleading:

- In `contracts/src/Functions.sol`, the commentary in lines `97-99` states that repeated PNA registration is allowed, however, it is rejected as it is seen from line `103`
- File `bridges/snowbridge/pallets/outbound-queue/src/send_message_impl.rs` should be placed in the `v1` folder
- In `bridges/snowbridge/primitives/outbound-queue/src/lib.rs:25-28`, the message `RejectingOutboundMessages` is declared. However, its name is misleading since it must affect only the `Gateway` contract by suspending all

messages going from Ethereum to Polkadot, but it is declared within the `outbound-queue` pallet.

- In `bridges/snowbridge/primitives/inbound-queue/src/lib.rs:23`, the commentary states that the structure `EthereumLocationsConverterFor` is deprecated. However, it is still used in the V2 source file: `bridges/snowbridge/primitives/inbound-queue/src/v2/converter.rs`

Recommendation

We recommend resolving the aforementioned misleading implementation details.

Status: Partially Resolved

23. Inefficiency in rejecting messages

Severity: Informational

In `contracts/src/Gateway.sol:418`, the `v2_submit` function validates the `message.nonce`, and reverts if the validation fails. However, it was observed that before that check is executed, it first calculates the hash of the provided message.

Similarly, in `contracts/src/Verification.sol:121-123`, returning `false` can be done before computing `parachainHeadHash`.

In both locations, the design does not adhere to the return-early pattern and will be consuming more gas than required for the calls with an invalid nonce.

Recommendation

We recommend performing validations before any computation when it is possible.

Status: Resolved

24. Unused code

Severity: Informational

Throughout the codebase, various definitions are not used:

- `contracts/src/Functions.sol:106`
    - Checking `info.isRegistered` is redundant since its value is always `false` in the `else` case
- `bridges/snowbridge/pallets/system-frontend/src/lib.rs`

    - The storage item `ExportOperatingMode` is written to, but is never queried
- `bridges/snowbridge/primitives/inbound-queue/src/v1.rs:5`
    - error `ConvertMessageError::UnsupportedVersion`
    - error `ConvertMessageError::UnsupportedFeeAsset`
- `contracts/src/v2/IGateway.sol:13`
    - error `InvalidFee`
- `contracts/src/Functions.sol:117`
    - return type `Token`
- `contracts/src/SubstrateTypes.sol`
    - functions `MultiAddressID`, `OptionParaID` and `OptionVecU8`
- `contracts/src/Types.sol` is probably not needed, because it contains only re-exported imports, and the dependencies listed there are used in many locations directly

Separately, these files import dependencies but don't use them, totaling 45 unused imports:

- `contracts/scripts/DeployLocal.sol`
- `contracts/scripts/FundGateway.sol`
- `contracts/src/AgentExecutor.sol`
- `contracts/src/Functions.sol`
- `contracts/src/Gateway.sol`
- `contracts/src/Initializer.sol`
- `contracts/src/storage/CoreStorage.sol`
- `contracts/src/v2/Calls.sol`
- `contracts/src/v2/Handlers.sol`

Recommendation

We recommend unused code.

Status: Partially Resolved