Audit Report

Snowbridge

v1.1

May 24, 2024

Table of Contents

License

Disclaimer

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	`fa632e665aa560a799f8396718ef81f31e26dc3d`
Scope	Only the Solidity contracts in `core/packages/contracts` and the Parachain code in `parachain` were in scope of this audit.
Fixes verified at commit	`1e27bce2c34989fd48f77d0ac8b6909ff09793c7` Note that changes to the codebase beyond fixes after the initial audit have not been in scope of our fixes review.

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 allows users to bridge ERC-20 tokens from Ethereum to 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**, 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 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 `67.09%`. Test coverage for Substrate pallets reported by `cargo tarpaulin` is `73.93%`. We recommend implementing unit test for XCM scripts in an emulated environment.

Summary of Findings

No	Description	Severity	Status
1	Attackers can drain relayer funds and hence DoS the bridge by spamming `create_agent` transactions	Critical	Resolved
2	Attackers can drain sovereign funds and hence DoS the bridge by spamming `registerToken` transactions	Critical	Resolved
3	`submit` extrinsic always returns `Ok`, causing stuck funds in the `Agent` contract in case of errors	Critical	Acknowledged
4	Administrative `Commands` are not implemented on Polkadot side	Major	Resolved
5	Throttling mechanism could delay critical governance operations	Major	Acknowledged
6	Unrestricted `Agent` funds transfer via `transferNativeFromAgent` command	Major	Resolved
7	Multiple attempts of random value draw through replicated ticket submissions	Major	Resolved
8	Corrupted messages in the queue are skipped, leading to stuck funds	Major	Resolved
9	The outbound queue continues processing messages even if `PalletOperatingMode` is `Halted`	Major	Resolved
10	Implementation of the `generalized_index_bit` function differs from the specification	Major	Resolved
11	Partial update may temporarily stall the bridge	Major	Resolved
12	Risk of token loss and channel immobilization after Gateway contract upgrade	Minor	Acknowledged
13	Malicious ERC-20 contracts could be used to mislead users	Minor	Acknowledged
14	Updates of the `Gateway` address in the `inbound-queue` pallet would result in a stuck bridge	Minor	Resolved

15	Missing mechanism for excessive fee reimbursement	Minor	Resolved
16	Misleading query output for non-existent channels	Minor	Resolved
17	Missing validations in `Gateway` contract's `constructor`	Minor	Resolved
18	XCM transfers are subject to limitations	Minor	Acknowledged
19	Message replay attack possible after `u64::MAX` messages	Minor	Resolved
20	Hard-coded gas usage can make the Ethereum side of the bridge unusable	Minor	Resolved
21	Excessive gas usage in corner cases	Informational	Resolved
22	Change in function order or indexes during parachain upgrades could lead to execution of wrong functions	Informational	Acknowledged
23	Risk of Ethereum side bridge disruption, if Gateway contract upgrades accidentally overwrite immutable variables	Informational	Acknowledged
24	Possible pallet centralized control	Informational	Resolved
25	Missing constant attribute for `MaxUpgradeDataSize`	Informational	Resolved
26	Test logic in production code	Informational	Resolved
27	Duplicated code	Informational	Partially resolved
28	Transaction lifecycle order issues in the `outbound-queue` pallet	Informational	Acknowledged
29	Excessive flexibility of the cross-chain interface	Informational	Resolved
30	Unused code	Informational	Partially resolved
31	Outdated comment	Informational	Resolved
32	`TODO` comments in the codebase	Informational	Resolved
33	Pallets should implement a two step ownership transfer	Informational	Resolved

Detailed Findings

1. Attackers can drain relayer funds and hence DoS the bridge by spamming `create_agent` transactions

Severity: Critical

In `parachain/pallets/control/src/lib.rs:128-161`, the `create_agent` extrinsic creates a `CreateAgent` message and adds it to a queue for eventual commitment. This allows it to be selected and sent by a relayer to the Ethereum `Gateway` contract.

For each such message, the relayer is expected to execute the `submit` function of the `Gateway` contract in order to process the `CreateAgent` Command. This requires the relayer to pay the gas to instantiate a new `Agent` contract.

Since the `create_agent` extrinsic can be sent permissionlessly by anyone, a potential vulnerability arises where an attacker could create a large number of accounts and spam the network with `create_agent` messages while only bearing the relatively low fees of the `create_agent` extrinsic defined by its weights.

This would cause the `queue` to be loaded with `CreateAgent` messages and, since their execution could be relatively expensive for the relayer, relayers may run out of funds or cease operations in anticipation of high costs.

This results in a denial-of-service (DoS) of the channel due to its ordered nature. In order to execute any message, all the other messages before it need to be executed. If the cost of executing all the previous messages is not economically feasible for relayers, the channel is stuck and messages are not relayed anymore.

Recommendation

We recommend charging an additional fee for executing the `create_agent` extrinsic and using it to compensate relayers executing `CreateAgent` commands.

Status: Resolved

2. Attackers can drain sovereign funds and hence DoS the bridge by spamming `registerToken` transactions

Severity: Critical

In `contracts/src/Gateway.sol:418-424`, the `registerToken` function enables users to register a new asset on Asset Hub.

However, the Asset Hub `create` extrinsic mandates a deposit of `10 DOT` when creating a new asset, as detailed in the documentation found at https://wiki.polkadot.network/docs/learn-assets#creation-and-management.

Consequently, every execution of the `registerToken` function within the `Gateway` contract necessitates the sovereign account to perform a `10 DOT` deposit to the Asset Hub.

Given that the origin is not covering this fee, there exists the potential for malicious actors to exploit this process. Attackers could spam the network with `registerToken` messages to exhaust sovereign account's funds, effectively implying a denial-of-service (DoS) attack of the channel.

In fact, since channels are ordered by the `nonce`, to execute any message, all the other messages before it need to be executed. If the cost of executing all the previous messages is not economically viable, the channel could be stuck and messages will not be relayed anymore.

Recommendation

We recommend making the `extraFee` proportional to the cost of token registration on Asset Hub for the execution of the `registerToken` function, taking the most recent DOT/ETH price as well as a buffer for price volatility into account, and compensating the sovereign account on the Polkadot side accordingly.

Status: Resolved

3. `submit` extrinsic always returns `Ok`, causing stuck funds in the `Agent` contract in case of errors

Severity: Critical

In `parachain/pallets/inbound-queue/src/lib.rs:206-241`, the `submit` function returns `Ok` regardless of whether errors occur during its execution.

This potential oversight poses the risk of funds becoming stuck on the Ethereum `Agent` contract in case an error emerges during the `SendToken` Command process on the Polkadot side.

If a user transfers tokens from Ethereum, they are transferred to the escrow of the corresponding `Agent` contract. Retrieving these funds requires the user to possess the bridged tokens on a Polkadot chain and bridge them through the Bridge Hub and the `outbound-queue` pallet. If an error occurs during the submit function, no bridged tokens will be minted on the Polkadot side. Without those bridged tokens, the locked funds will be unrecoverably stuck in the Ethereum `Agent` contract, resulting in a loss for the user.

Recommendation

We recommend implementing an error-handling mechanism in order to return tokens from the `Agent` to the user if the bridging process does not succeed on the Polkadot side.

Status: Acknowledged

The client acknowledges this issue and aims to address it in a future revision.

4. Administrative `Commands` are not implemented on Polkadot side

Severity: Major

After analyzing the code included in the scope of the audit, it was found that in the case of the `outbound-queue` palette, core `Commands` were not implemented, making it impossible to actively manage the bridge. This includes `Commands` such as:

- `CreateChannel`
- `UpdateChannel`
- `SetOperatingMode`
- `TransferNativeFromAgent`

It is worth noting that the codebase submitted for this audit contains implementations for these commands on the Ethereum side.

We report this issue as major instead of critical since none of the missing `Commands` present an immediate security threat. However, the repercussions could be significant, leading to the inability to manage the bridge and limiting its functionality.

Recommendation

We recommend completing the implementation on the Polkadot side. The `outbound-queue` pallet should have appropriate extrinsics to enable authorized calls from the privileged user or through governance, which then triggers the corresponding action on the Ethereum side. Additionally, we recommend implementing in-depth tests for each `Command`.

Status: Resolved

5. Throttling mechanism could delay critical governance operations

Severity: Major

In `parachain/pallets/outbound-queue/src/lib.rs:351-375`, a throttling mechanism for the queue is implemented in order to bound the execution 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 to serve 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 in a timely manner in order to address malfunctioning or exploits of the bridge.

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

Recommendation

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

Status: Acknowledged

The client acknowledges this issue and aims to implement a new queue system in a future revision.

6. Unrestricted `Agent` funds transfer via `transferNativeFromAgent` command

Severity: Major

In `contracts/src/Gateway.sol:400-412`, the `transferNativeFromAgent` function enables agents to transfer any number of funds from any agent, without any restriction on the originating channel.

This implies that a user creating an agent could potentially exploit this message to steal all funds within the channel.

While there is currently no way to call `transferNativeFromAgent`, we report this issue with major severity since it could have critical consequences if not addressed properly.

Recommendation

We recommend restricting fund access to the agent itself.

Status: Resolved

7. Multiple attempts of random value draw through replicated ticket submissions

Severity: Major

In `contracts/src/BeefyClient.sol:231-290`, attackers could exploit the light client by initiating multiple ticket submissions to obtain a favorable random value from `prevRandao`. This could be accomplished through the replication of actions such as `submitInitial` or `submitInitialWithHandover` and `commitPrevRandao`.

Although a safeguard in line `272` disallows the invocation of `commitPrevRandao` for the same ticket, it remains feasible for attackers to use `submitInitial` or `submitInitialWithHandover` to generate a new ticket to attempt to draw a favorable ticket.

Recommendation

We recommend requiring a temporary deposit for submitting tickets to economically discourage attackers as well as not allowing multiple ticket submissions from the same address.

Status: Resolved

8. Corrupted messages in the queue are skipped, leading to stuck funds

Severity: Major

In `parachain/pallets/outbound-queue/src/lib.rs:270-274`, the execution of `do_process_message` is initiated by the `message-queue` pallet to handle messages. In line `272`, the message is parsed, and if it fails, the `ProcessMessageError::Corrupt` error is returned. This error subsequently prompts the `message-queue` pallet to permanently discard the message. This outcome can lead to a situation where tokens have already been burned in Asset Hub, and the discarded message results in the corresponding tokens on Ethereum being indefinitely stuck in the `Agent` contract.

Moreover, having corrupted messages in the queue means that the state is corrupted and actions should be taken in order to protect user funds, for example pausing pallet operations.

Recommendation

We recommend pausing bridge operations in case of a corrupted state.

Status: Resolved

The client implemented message versioning to handle different message structures. Messages that are not parsable in any versioned message, are discarded under the assumption that Bridge Hub constructs always valid ones.

9. The outbound queue continues processing messages even if `PalletOperatingMode` is `Halted`

Severity: Major

In `parachain/pallets/outbound-queue/src/lib.rs:351-374`, the `process_message` function lacks a validation step for the `PalletOperatingMode` status.

While it is not possible to `submit` new messages while `PalletOperatingMode` is `Halted`, existing messages already in the queue (which could hold a significant number of messages) can be further processed.

In the event of an emergency situation this would result in messages continuing to be relayed even after governance halts operations.

Recommendation

We recommend implementing a validation step for the `PalletOperatingMode` status within the `process_message` function.

Status: Resolved

10. Implementation of the `generalized_index_bit` function differs from the specification

Severity: Major

In `parachain/primitives/beacon/src/merkle_proof.rs:44`, the `generalized_index_bit` function implementation differs from the specification given in https://github.com/ethereum/consensus-specs/blob/dev/ssz/merkle-proofs.md#get_generalized_index_bit.

Because of this, the generated `merkle_root` is not compliant with the specification.

Recommendation

We recommend reworking the implementation of the `generalized_index_bit` to follow the specification.

Status: Resolved

11. Partial update may temporarily stall the bridge

Severity: Major

During the execution of the `submit` extrinsic of the `ethereum-beacon-client` pallet, the `verify_update` function defined in `parachain/pallets/ethereum-beacon-client/src/lib.rs:362-482` is invoked to validate the provided update.

However, if the update includes a header from the next period but does not provide the `next_sync_committee_update`, no error is raised, leading to a partial state update where the light client reference period is not updated.

Consequently, it becomes impossible to update the light client state due to the period mismatch, causing all the subsequent calls to the `submit` extrinsic to return an error and halting bridge operations until governance forces a new checkpoint.

This issue has been found by the client after the audit report has been published, but has been reviewed by our team and added to this report for completeness.

Recommendation

We recommend rejecting new updates for the Ethereum light client if the update includes a header from the next period but does not contain the `next_sync_committee_update`.

Status: Resolved

12. Risk of token loss and channel immobilization after `Gateway` contract upgrade

Severity: Minor

In `contracts/src/Gateway.sol:367-373`, during the course of `upgrade` function execution, governance has the capability to execute the `initialize` function providing a new set of `initParams` as input.

However, since the `initialize` function creates two new agents for Bridge Hub and Asset Hub that will overwrite the addresses of existing ones, this process inadvertently results in the inaccessibility of tokens bridged by these agents, leading to an irreversible loss.

Moreover, it has the unintended consequence of indefinitely immobilizing the associated channels, as the initialization process also resets the `nonce` to zero.

We classify this issue as minor since it can only be caused through governance.

Recommendation

We recommend revising the initialization process for the Bridge Hub and Asset Hub agents in lines `537-549` and updating the contract with a different `initialize` function when performing an upgrade.

Status: Acknowledged

13. Malicious ERC-20 contracts could be used to mislead users

Severity: Minor

Due to the ability to permissionlessly bridge arbitrary tokens from Ethereum to Polkadot, attackers could create malicious ERC-20 token contracts that do not actually put tokens into escrow on Ethereum and use this mechanism to mislead users. This can be achieved through the logic in `contracts/src/utils/SafeTransfer.sol:13-21`.

An attacker could implement a contract featuring a `TransferFrom` function that consistently yields `true` results. Using this contract, they could execute the `registerToken` function and subsequently invoke `sendTokens` to facilitate a successful execution. This process would lead to event emission, command relay, and token creation on the bridge without the necessity of holding funds in the Ethereum agent's escrow.

Attackers could use this behavior to pursue phishing campaigns or otherwise mislead users by using tokens not redeemable on Ethereum.

Recommendation

We recommend implementing a message to enable governance to unregister malicious tokens.

Status: Acknowledged

14. Updates of the `Gateway` address in the `inbound-queue` pallet would result in a stuck bridge

Severity: Minor

In `parachain/pallets/inbound-queue/src/lib.rs:243-249`, the `set_gateway` extrinsic enables governance to update the `Gateway` Ethereum contract address.

However, since the `Gateway` address is used to compute the `GlobalConsensusEthereumAccountConvertsFor`, updating it would cause the pallet to not be able to control funds on Asset Hub anymore.

Furthermore, messages already in the queue would retain data tied to the old gateway address, rendering their processing impossible. Consequently, advancing the `nonce` becomes unachievable, effectively causing the bridge to become non-operational.

We classify this issue as minor since it can only be caused through governance.

Recommendation

We recommend enforcing immutability of the `Gateway` address.

Status: Resolved

15. Missing mechanism for excessive fee reimbursement

Severity: Minor

In `contracts/src/Gateway.sol:466-485`, while executing the `_submitOutbound` function, verification is conducted in line `474` to ensure that the user has sent an adequate amount of ETH to cover the required fees.

However, the current implementation lacks a mechanism for reimbursing users who have sent an excessive amount of fees.

Consequently, any surplus funds sent to this contract will be stuck in the contract.

Recommendation

We recommend introducing a refund mechanism within the `_submitOutbound` function.

Status: Resolved

16. Misleading query output for non-existent channels

Severity: Minor

In `contracts/src/Gateway.sol:196-204`, in the event of a non-existent channel provided as input, the `channelNoncesOf` and `channelFeeRewardOf` functions will always return a misleading `(0,0)` tuple, which does not accurately represent store values due to the channel's absence.

Recommendation

We recommend returning an error message instead to communicate the non-existent channel state.

Status: Resolved

17. Missing validations in `Gateway` contract's `constructor`

Severity: Minor

In `contracts/src/Gateway.sol:74-93`, while executing the `Gateway` constructor, there is no validation to ensure that `ASSET_HUB_AGENT_ID` and `BRIDGE_HUB_AGENT_ID` are distinct.

Similarly, `ASSET_HUB_PARA_ID` and `BRIDGE_HUB_PARA_ID` are not validated to not be equal.

This validation is essential to prevent potential conflicts during the execution of `instantiate`, wherein the bridge hub data could overwrite the asset hub data.

Recommendation

We recommend ensuring that `ASSET_HUB_AGENT_ID` and `BRIDGE_HUB_AGENT_ID` as well as `ASSET_HUB_PARA_ID` and `BRIDGE_HUB_PARA_ID` are distinct.

Status: Resolved

18. XCM transfers are subject to limitations

Severity: Minor

In `parachain/primitives/router/src/outbound/mod.rs:38` the `validate` function is employed to check XCM messages originating from the Asset Hub parachain.

It translates XCM instructions to `AgentExecuteCommand` and, if validation passes, generates a new ticket for the `OutboundQueue`.

However, not all valid XCM scripts pass this validation due to their fee execution handling. In `215-228`, the `fee_info` function identifies expected fees from XCM instructions. Specifically:

- A combination of `WithdrawAsset` and `BuyExecution` indicates the message's origin intends to pay the fees.
- The `UnpaidExecution` instruction signifies the origin anticipates zero-cost execution.

Yet, lines `111-114` reject XCM messages from origins willing to pay the fee. This is inconsistent since both fee payment scenarios should be viable, especially given the assumption that Asset Hub transfers utilize the `bridge-transfer` pallet.

Recommendation

We recommend providing support for both fee-handling scenarios in XCM messages. In cases where one scenario becomes redundant (such as the bridge consistently covering fees), we recommend clearly documenting these limitations.

Status: Acknowledged

19. Message replay attack possible after `u64::MAX` messages

Severity: Minor

In `parachain/pallets/inbound-queue/src/lib.rs:192-200`, the `nonce` value increment overflows after `u64::MAX` messages.

Consequently, after `u64::MAX` messages, it will be possible to replay in sequence all previous messages.

We classify this issue as minor since it is very costly to execute `u64::MAX` messages.

Recommendation

We recommend using `saturate_add` as a precaution against possible overflows.

Status: Resolved

20. Hard-coded gas usage can make the Ethereum side of the bridge unusable

Severity: Minor

In `contracts/src/Gateway:29`, a static value known as `DISPATCH_GAS` is employed for executing smart contract calls within the `Gateway` contract. However, it is possible that Ethereum's gas model changes over time due to the continual introduction of Ethereum Improvement Proposals (EIPs) that alter the gas consumption of `opcodes`. This introduces a risk wherein certain function call executions within the `Gateway` contract may become infeasible if the gas required surpasses the sum of `DISPATCH_GAS` and `BUFFER_GAS`. In such cases, transactions would revert, thereby rendering core operations like upgrades or the creation of agents impossible.

This is most problematic for the `upgrade` function call, since the logic contract's `initialize` function definition, which plays a pivotal role in the upgrade process, is not predetermined. This lack of predictability can lead to a situation where the upgrade function reverts if an insufficient gas value is provided. Consequently, determining the precise value for `DISPATCH_GAS` becomes an exceedingly challenging endeavor.

Recommendation

We recommend incorporating `gasLeft()` within the function execution rather than relying on a static value like `DISPATCH_GAS`. Since it is already checked in the code that `gasLeft()` is greater than the sum of `DISPATCH_GAS` and a buffer, this approach empowers `msg.sender` to supply additional gas as needed, thereby enabling successful function execution even in cases where higher gas consumption is encountered.

Status: Resolved

21. Excessive gas usage in corner cases

Severity: Informational

Within `contracts/src/Gateway.sol:99-181`, the `submitInbound` Solidity function behaves as follows:

1. Validate the channel from the originating parachain.
2. Verify the message proof.
3. Confirm the header commitment.
4. Validate the nonce and increment it.
5. Reward the relayer.
6. Check the remaining gas.
7. Dispatch the message.
8. Emit an event for completion.

Steps 2 and 3 are resource-intensive, consuming significant gas. Meanwhile, steps 4 and 5 are significantly less resource demanding. If an erroneous nonce is submitted by a relayer, or the agent lacks sufficient funds for rewarding the relayer, the function reverts, wasting the computation from steps 2 and 3.

Recommendation

We recommend optimizing gas efficiency by reordering the calls to `MerkleProof.processProof` and `verifyCommitment` behind nonce validation and reward allocation.

Status: Resolved

22. Change in function order or indexes during parachain upgrades could lead to execution of wrong functions

Severity: Informational

In `contracts/src/Assets.sol:108`, the param `createTokenCallID` is designated as the reference for the execution of a specific function on the parachain side. However, it is possible that the indices associated with functions may change during the course of parachain pallet upgrades. This can occur when developers of the pallets fail to employ the `call_index` attribute for dispatchables or neglect to maintain consistent call indices across various upgrades.

As a consequence, the Snowbridge implementation might inadvertently trigger a dissimilar dispatchable function on the predefined index if it does not adapt its bridge settings to align with the evolving codebase of the parachain.

Recommendation

We recommend having close coordination between Snowbridge's configuration and the parachain's codebase, particularly in the context of potential modifications to call indices resulting from pallet upgrades. Such alignment is essential to ensure the seamless and accurate execution of intended functions across the bridge. We also recommend clearly documenting the need for this coordination.

Status: Acknowledged

23. Risk of Ethereum side bridge disruption, if `Gateway` contract upgrades accidentally overwrite immutable variables

Severity: Informational

In `contracts/src/Gateway:369`, the `Gateway` contract relies on several immutable variables. These variables are embedded within the bytecode of the logic contract associated with the `GatewayProxy`. Importantly, in the event that the logic contract undergoes changes due to an upgrade, these immutable variables are at risk of being lost.

This is a critical consideration because the Ethereum side of the bridge could face operational disruptions if the new logic contract does not preserve these crucial immutable variables. Therefore, it becomes imperative for the developers of Snowbridge to ensure that any subsequent iterations of the logic contract encompass all the necessary immutable variables to maintain the seamless functionality of the bridge.

Recommendation

We recommend performing thorough manual checks on the new logic contract before the upgrade to make sure it preserves those immutable variables. We also recommend clearly documenting these manual checks.

Status: Acknowledged

24. Possible pallet centralized control

Severity: Informational

The `inbound-queue`, `outbound-queue` and `ethereum-beacon-client` pallets implement the `OwnedBridgeModule` trait, which allows on-chain governance to appoint an "owner":

```
pub fn set_owner(origin: OriginFor<T>, new_owner:
Option<T::AccountId>)
```

The owner has the power to halt the operation of these pallets. While a pallet is halted, all its operations (except operating mode change) are prohibited. Halting any of the pallets effectively leads to halting the entire bridge. This feature can serve as an additional security measure, particularly in the face of a potential attack where on-chain governance might be too slow to respond.

However, such a halt can potentially interrupt cross-chain transfers mid-process. These transfers would remain on hold until the bridge is reactivated.

Since the owner possesses the right to halt and resume the bridge's normal operations, voluntarily step down, or transfer ownership to another party, a compromised owner account may disrupt bridge operations, causing a potential loss of trust in the bridge and leading to delays of time-critical messages.

Note that a malicious owner's capacities are limited. They cannot change the `Gateway` smart contract's address and can be removed via the on-chain governance process.

Recommendation

We recommend following key management and continuous system monitoring best practices for the owner account.

Status: Resolved

25. Missing constant attribute for `MaxUpgradeDataSize`

Severity: Informational

In `parachain/pallets/control/src/lib.rs:55`, there is a missing `#[pallet:constant]` attribute for the `MaxUpgradeDataSize` type.

Because of this, `MaxUpgradeDataSize` would not be included in the pallet's metadata.

Recommendation

We recommend using the `#[pallet::constant]` attribute to add an associated type trait bounded by `Get`.

Status: Resolved

26. Test logic in production code

Severity: Informational

In `contracts/src/Gateway.sol:457-464` exists code related to testing.

It is best practice to eliminate test logic from production code in order to enhance code readability and maintainability.

Recommendation

We recommend removing test logic from the production code or using features to selectively build them.

Status: Resolved

27. Duplicated code

Severity: Informational

Several instances of code duplication have been found during this audit:

- The file `parachain/primitives/router/src/inbound/mod.rs` contains exact duplicates in lines `101-120` and `146-165`.

  There is a recurring sequence of 5 XCM instructions in command handlers, specifically: `UniversalOrigin`, `DescendOrigin`, `WithdrawAsset`, `BuyExecution`, and `SetAppendix`. Moreover, the `SetAppendix` instruction is nested and contains further instructions.

While these repetitions might seem minor, the execution of XCM instructions can be intricate. Hence, in this context, streamlining is recommended. This can be achieved by setting up a closure outside the handlers as illustrated below:

```
let create_instructions = |origin_location: Junction| ->
Vec<Instruction<()>> {
...
}
```

- The file `contracts/src/Assets.sol` contains near-identical code fragments in lines `39-59` and `61-81`.

  Two definitions of the function `sendToken` are declared, each catering to a distinct data type for the `destinationAddress` parameter: One for `bytes32` and another for `address`. The only difference between these two functions is the handling of the case when `destinationChain` is equal to `assetHubParaID`. In this case, the transaction reverts if `destinationAddress` is passed as an Ethereum-style `address` type.

  While both functions are needed to support the variance of argument types, it's advised to consolidate the overlapping logic by implementing a helper function. After extracting the shared logic, it is possible to route the original `sendToken` functions to this helper function post necessary argument validation.

It is best practice to remove code duplication, as they negatively impact the readability and maintainability of the codebase.

Recommendation

We recommend extracting common logic into helper functions or closures.

Status: Partially resolved

28. Transaction lifecycle order issues in the `outbound-queue` pallet

Severity: Informational

In `parachain/pallets/outbound-queue/src/lib.rs204-220`, transaction lifecycle hooks are defined for the `outbound-queue` pallet.

The `on_initialize` function erases all data from the `Messages` and `MessageLeaves` of the preceding block. However, there is no enforcement mechanism ensuring that the `on_initialize` function of the `message-queue` pallet (which populates these vectors) is executed after the `outbound-queue` pallet's `on_initialize` function. If the order of these `on_initialize` is reverted, the `kill` method would erase all messages from these queues before their processing.

Moreover, the assigned `Weight` for the `on_finalize` function is an arbitrary value. Depending on the `MaxMessagesPerBlock` setting, this weight might not be sufficient to process the Merkle tree of messages effectively.

Recommendation

We recommend documenting the required sequencing of the `on_initialize` functions and revisit the `Weight` for the `on_finalize` function.

Status: Acknowledged

29. Excessive flexibility of the cross-chain interface

Severity: Informational

In the `parachain/primitives/router/src/inbound/mod.rs` file, the `Command` enumeration is defined. It features the `RegisterToken` command variant which holds the `create_call_index` field, sized 2 bytes.

This command originates from the Ethereum side via the bridge. Specifically, the `contracts/src/Gateway.sol` file defines the `registerToken` external function. In line `419`, the `CREATE_TOKEN_CALL_ID` constant fills in the `RegisterToken` command's `create_call_index` field. During the `Gateway` smart contract's initialization, the Solidity constant can be set arbitrarily by its deployer. Similarly, the deployer can also configure the `ASSET_HUB_PARA_ID`.

On the Polkadot side, the `RegisterToken` command gets translated into an XCM script with the `Transact` instruction, targeting a specific parachain. The `create_call_index` field is used here to select the right asset creation extrinsic.

While this design showcases the bridge's adaptability to alternative asset parachains, it is more error-prone due to the necessity of writing Solidity code with consideration for the internal details of the parachain implementation. If a mistake occurs in newer versions of the contract, tracking its impact can become challenging.

Recommendation

For easier maintenance, we recommend keeping internal data, like the call selector, closer to its point of use. Embedding it in a static location can minimize deployment errors. Should there be a need to engage with multiple asset parachains, we recommend integrating a mapping between parachain IDs and call indices directly in the Rust pallet.

Status: Resolved

30. Unused code

Severity: Informational

There are redundant data structures in the codebase, especially events and error messages. Below is a list of these findings:

- Events not emitted in the Ethereum contracts:
  - `OperatingModeChanged`
  - `AgentFundsWithdrawn`
  - `TokenRegistered`
- Errors not thrown in the Ethereum contracts:
  - `InvalidAgentExecutor`
  - `InvalidConfig`
  - `NotProxy`
  - `FailedPayment`
  - `UnknownChannel`
  - `WithdrawalFailed`
- Unused storage imports in the Ethereum contracts:
  - `contracts/src/storage/AssetsStorage.sol:5-6`
  - `contracts/src/storage/CoreStorage.sol:5-6`
- Redundancies in the Substrate parachain:
  - `impl<L> ContainsPair<MultiLocation, MultiLocation>`
  - Error message `InvalidAccountConversion`

It is best practice to remove unused code, as it negatively impacts the readability and maintainability of the codebase.

Recommendation

We recommend removing unused code.

Status: Partially resolved

31. Outdated comment

Severity: Informational

In `contracts/src/BeefyClient.sol:564-579`, the comment above the `isValidatorInSet` function implementation probably refers to a previous version of the code, since the `index` parameter is verified as part of the `SubstrateMerkleProof.verify` call.

Recommendation

We suggest updating the comment accordingly and describing all parameters passed to the `isValidatorInSet` function.

Status: Resolved

32. `TODO` comments in the codebase

Severity: Informational

In `parachain/primitives/router/src/inbound/mod.rs:82-83` and `parachain/primitives/router/src/outbound/mod.rs:138`, TODO comments have been found in the codebase. TODO comments in production code are a deviation from best practices.

Recommendation

We recommend resolving and removing the TODO comments.

Status: Resolved

33. Pallets should implement a two step ownership transfer

Severity: Informational

The pallets within the scope of this audit allow the current owner to execute a one-step ownership transfer. While this is common practice, it presents a risk for the ownership of the pallet to become lost if the owner transfers ownership to the incorrect address. A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address.
2. The new owner account claims ownership.

Status: Resolved

Security Model

The security model has been carefully crafted to delineate the various assets, actors, and underlying assumptions of Snowbridge. They will then be analyzed in a threat model to outline high-level security risks and proposed mitigations.

The purpose of this security model is to recognize and assess potential threats, as well as the derivation of recommendations for mitigations and counter-measures.

There is a limit to which security risks can be identified by consructing a security/threat model. Some risks may remain undetected and may not be covered in model described below (see disclaimer).

Assets

The following outlines assets that hold significant value to potential attackers or other stakeholders of the system.

Messages

Data packets play a pivotal role in the functioning of Snowbridge, facilitating the exchange of crucial information among its various components. These messages encompass transaction data, requests, and other relevant information necessary for the bridge's operation like commitments for the light client. The security of these messages is of paramount importance, as it safeguards against potential threats like tampering, eavesdropping, or unauthorized access, ensuring the integrity of bridge communications.

ERC-20 tokens

ERC-20 tokens are digital assets established on the Ethereum blockchain, adhering to the well-defined ERC-20 standard. These tokens embody the concept of fungibility, making them highly versatile and extensively utilized across a broad spectrum of applications, such as decentralized finance (DeFi) platforms, token sales, and digital asset transfers. In the context of Snowbridge, ERC-20 tokens take center stage as the prime assets that users seek to transfer between the Ethereum blockchain and Polkadot parachains or vice versa.

The security of ERC-20 tokens becomes a paramount concern, necessitating robust protective measures against potential theft or unauthorized access. Proper authorization protocols must be implemented to ensure that only permitted transfers occur, guarding against malicious activities and verifying their integrity during the transfer process.

Asset Hub tokens

Asset Hub tokens are specific digital assets intrinsic to the Asset Hub parachain, purposefully employed by the bridge to serve as representations of bridged ERC-20 tokens within the Polkadot/Kusama ecosystem.

The main focus lies in guaranteeing the security of Asset Hub tokens, which necessitates strong safeguards against any unauthorized transfer, mint, burn, access attempts, manipulative actions, or fraudulent transactions. Given their crucial role in the seamless functioning of the blockchain bridge, ensuring the integrity and protection of these tokens is of utmost significance to maintain the trust and reliability of the entire system.

Polkadot staking tokens

In the Polkadot network, DOT is the native token and is used for various purposes, including staking. Staking involves locking up a certain amount of tokens to support the network's operations and security. Participants who stake their DOT have the opportunity to become validators and contribute to the consensus process.

Additionally, DOT acts as a governance token, which constitutes digital assets that bestow their holders with the privilege to actively participate in the decision-making processes of decentralized governance systems.

Within Snowbridge, this governance token is DOT. DOT plays a pivotal role by empowering owners to exercise their voting rights in critical matters concerning the bridge's operation, upgrades, and emergencies.

The highest importance lies in securing these governance tokens to prevent unauthorized voting attempts or manipulative actions in governance-related activities, as they can directly impact the bridge's governance.

Ethereum staking tokens

In the Ethereum network, ETH is the native token, which is used for various purposes, including staking. Staking involves locking up a certain amount of tokens to support the network's operations and security.

ETH is not used as a governance token in Snowbridge, since all the operations and decisions are demanded by the Polkadot governance.

Private keys controlling external accounts

Regarding externally owned accounts, a specific entity retains control over the private key associated with the account. The significance of this private key lies in granting full authority over the account, making it an attractive target for attackers.

Access to the private key bestows the attacker with the ability to impersonate the legitimate account owner and execute actions from that account.

For validators, a private key serves the additional function of signing the BEEFY commitments. If an attacker was able to compromise a sufficient number of such validator private keys, they could forge BEEFY commitments.

Agent

Agents play a critical role within the bridge ecosystem. Deployed on Ethereum as smart contracts, they act as sovereign agents for arbitrary consensus systems on Polkadot. These consensus systems can include parachains as well as nested consensus systems within a Parachain. The generation of agents can occur in a permissionless manner, but ensuring their security is of utmost importance for maintaining the integrity of bridge communication. If an attacker gains control over an agent, they possess the capability to carry out malicious transactions on the connected chain and potentially impersonate that agent or abscond with fee funds.

RANDAO outcome

The RANDAO mechanism holds significant importance in guaranteeing an equitable rotation of sync committee members and the operations in the BEEFY light client.

The protocol incorporates the RANDAO outcome for both light clients. Within the Ethereum light client, the utilization of RANDAO is implicit in the Altair sync committee election. Conversely, within the BEEFY light client, RANDAO is explicitly used to select validators for signature verification.

However, if the outcome of RANDAO can be manipulated, it creates a worrisome scenario where attackers could collaborate with malicious or compromised validators. Such collusion poses a grave threat to the security of light clients.

Upgrade authorization

Code upgrade authorization pertains to the permissions or procedures necessary for implementing alterations to the Ethereum smart contracts or pallets of Snowbridge. Given the critical dependence of the bridge's security on the integrity of its code, any modifications must undergo rigorous examination and obtain proper authorization to mitigate potential vulnerabilities or exploits. A comprehensive and secure code upgrade process ensures that only authorized entities possess the ability to modify the bridge's code. This stringent control minimizes the risk of introducing security flaws or compromising the bridge's overall functionality.

Liveness of the protocol

An attacker may find it advantageous to compromise the liveness of the protocol, aiming to hinder users from sending messages effectively. By doing so, the attacker could potentially exploit unsent messages preemptively or cause disturbances in the regular operations of the system. This is critical for time-sensitive messages, for example price oracles that need to be up-to-date or votes in cross-chain governance protocols that need to be received before a poll ends. A sufficiently long attack on the liveness could even be exploited to let the price of bridged tokens diverge, since liquidity provision, arbitrage and redemption of tokens could be delayed.

Liveness of the underlying blockchains and relayers

Deliberately targeting the liveness of underlying blockchains by attacking validators and relayers can present significant value to an attacker. Through such actions, the attacker may exploit opportunities for profit by shorting the native tokens or coercing stakeholders into halting the attack. As a result, this malicious activity might lead to disruptive and unstable conditions within the blockchain ecosystem.

Essential gas price requirements for protocol functionality

If the gas price gets prohibitively expensive, it may lead to a circumstance where the protocol or its regular operation (e.g. deployment of Agents or normal operation of BEEFY at regular intervals) becomes economically non-viable or impractical from a business perspective.

Stakeholders/Potential Threat Actors

The following outlines the various stakeholders or potential threat actors that interact with the system.

Bridge Hub system parachain

Bridge Hub serves as a system parachain, incorporating the Snowbridge pallets to facilitate cross-chain transfers.

The uncompromising security and continuous liveliness of the hosting parachain are indispensable in safeguarding the integrity of bridged messages, smooth message relay, and the overall efficiency of bridge operations.

It is important to highlight that the Bridge Hub parachain will serve as a host for other bridges. Ensuring that other pallets deployed on Bridge Hub do not disrupt Snowbridge operations or excessively consume resources that could hinder the optimal functioning of Snowbridge is of significant importance.

Asset Hub system parachain

Asset Hub is a system parachain, used by Snowbridge, operating as a vital component of the system. It assumes the responsibility of managing bridged tokens and ensures the bridge’s smooth and reliable operation. Preserving the security of Asset Hub is crucial to prevent unauthorized access to tokens, governance manipulation, and potential attacks that could impact the bridge's integrity.

Polkadot Governance

Polkadot Governance holds a crucial role in overseeing the operation of the bridge, with the authority to upgrade both the Snowbridge pallets and its Ethereum smart contracts. Additionally, it possesses the capability to enforce checkpoints within light clients. Ensuring the integrity and security of Polkadot Governance is paramount to safeguard against any unauthorized alterations or interference with the bridge's checkpoints and related processes.

Relayers

Off-chain software, referred to as relayers, plays a crucial role in monitoring and relaying messages between different blockchains. As blockchains do not directly exchange messages, these relayers form a vital part of the infrastructure facilitating cross-chain communication.

The Snowbridge architecture comprises three types of relayers:

- **Beacon Chain Relayer**
  This relayer’s primary responsibility is to fetch headers from an Ethereum node and transmit updates to Snowbridge's beacon chain light client pallet on Polkadot.

- **BEEFY Relayer**
  This relayer is tasked with fetching headers from the Polkadot Relay Chain and transmitting updates to the BEEFY light client smart contract on Ethereum.

- **Message Relayer**
  The message relayer is responsible for transmitting messages between different components of the bridge. It acts as a bridge between different blockchain networks. A compromised message relayer could introduce disruptions by tampering with messages or causing delays. Such interference may potentially lead to issues like disrupted asset transfers or other security-related concerns.

Ethereum Validators

Ethereum validators play a pivotal role in validating transactions and blocks on the Ethereum network.

Regarding Snowbridge, they hold the capacity to influence the generation of RANDAO values, which impact the election of the sync committee and the group of validators randomly selected for the BEEFY commitment verification.

The utmost priority lies in safeguarding the security and integrity of Ethereum validators to prevent any potential manipulation or unauthorized influence on the bridge's operation.

Within the system, validators could hold specific roles:

- **Altair Sync Committee**
  The Altair sync committee consists of 512 validators who undergo random selection in each sync committee period, approximately every 27 hours. Validators within the active sync committee receive an additional 0.1 ETH (each) during that period if they successfully sign and submit attestations to the latest beacon chain block. However, failing to include these attestations in the subsequent block leads to a corresponding ETH penalty.

  The sync committee holds a vital role within the Altair light client protocol, and any misconduct or collusion posed among its members present a significant threat to the bridge security.

  It should be emphasized that while the sync committee is accountable for block signing, the ultimate block finality is guaranteed by beacon committees. In the unlikely scenario where a block, endorsed by the sync committee's supermajority, fails to be finalized by the beacon committee and experiences a rollback, light clients that have synchronized with that block would acquire an incorrect view of the blockchain's state. In such cases, these clients must perform a resynchronization to align with the correct chain. This could be achieved by forcing a checkpoint as well. In general, importing only finalized blocks prevents this issue.

Another concern related to sync committees in the current implementation of Ethereum is the absence of slashing for severe misbehaviors. Such misbehaviors could include actions like double voting and surround voting. Presently, the only slashable form of misbehavior is the failure to sign a block while being a member of the sync committee.

- **Beacon Committees**
  Finalization is accomplished through the collective agreement of validators within the beacon committees, who validate the legitimacy of blocks. Each epoch in Ethereum spans 32 slots, approximately 6.4 minutes. Within each slot, a committee of validators is chosen randomly to attest to the validity of the proposed block for that slot. Once a block garners a sufficient number of attestations, it becomes justified. A justified block B_1 becomes finalized if a justified block B_2 from the next epoch refers to B_1 as its parent. A beacon committee's size may vary depending on the total number of active validators, but the protocol aims to maintain at least 128 validators in each committee. Overall, beacon committees divide up the validator set so that every active validator attests in every epoch, but not in every slot.

Polkadot Validators

Validators play two pivotal roles in upholding the integrity of the Polkadot network. Firstly, they create and vote on new blocks on the Relay Chain. Secondly, they scrutinize and authenticate the data contained within the designated parachain blocks.

Users

Users refer to individuals or entities who utilize the bridge to transfer assets between different blockchains.

In a more granular sense, users can fall into several categories:

- a parachain
- a protocol on a parachain
- an end-user of a protocol on a parachain
- an end-user on the Polkadot network
- an end-user on the Ethereum network
- a protocol on Ethereum
- an end-user of a protocol on Ethereum

Users may be at risk of attacks such as social engineering, phishing, unauthorized access to wallets, or receiving fraudulent information.

Supply chain

The software supply chain encompasses various components like libraries, dependencies, and compilers used within Snowbridge's codebase.

If any of these components are compromised or contain vulnerabilities, it could lead to the introduction of malicious code or potential exploits within the bridge.

Assumptions

The following outlines various assumptions upon which the system's functioning is predicated.

Maximum number of active validators

During the calculation of the minimum threshold signatures for a current validator set, the BEEFY light client assumes that the Polkadot Relay Chain will not have more than 20,000 active validators in a session.

At the time of the BEEFY light client implementation, Polkadot had 300 validators and Kusama had around 1000 validators. The Snowbridge team assumes that an order of magnitude increase of the validator set size will likely require a re-architecture of Polkadot that would make the current light client obsolete, see `contracts/src/BeefyClient.sol:425-447` for details.

RANDAO manipulation

Interactive Update Protocol commitment verification assumes that the integrity of the protocol remains robust even when subjected to limited RANDAO manipulation.

The protocol is designed to maintain a high level of security by relying on a maximum threshold for potential RANDAO manipulation. Even at this upper limit, the associated risk of manipulation is assumed to be exceedingly small as described by the Snowbridge team in this analysis.

Ethereum's finalization time

Snowbridge works on the assumption that Ethereum's finalization time is reliable and consistent enough to ensure that messages will not be reverted. This assumption forms the basis for relying on the approximately 3-epoch time period, corresponding to the waiting period for the block containing a message to be finalized.

However, a longer finalization time has been observed, for example on May 11, 2023, when finality took 8 epochs. The current Ethereum roadmap includes research for single slot finality.

Initial checkpoint

The synchronization process of the BEEFY light client relies on the governance entity to furnish a dependable initial block and the sync committee that is presently operational within the designated sync period of the provided block.

Honest Majority

In the simple case of static presence of malicious validators, i.e. when a proportion of them does not change over time, we can estimate the chance of sync committee collusion under the security assumption of the honest majority of Ethereum validators.

Let's model random sampling involved in sync committee formation using a hypergeometric distribution. The binomial distribution is similar but provides a less precise analysis because it models a random process over an infinite collection, while we always have a finite amount of nodes in the network. Here are the parameters of the model:

N: Total number of validators
m: Global proportion of malicious validators
M: Number of malicious members of the sync committee
512: The size of the sync committee
341: Supermajority threshold

Then the probability of random sampling a sync committee with malicious supermajority of validators is:

$$P(M \geq 341) = \sum_{k=341}^{512} \frac{\binom{N \cdot m}{k} \cdot \binom{N \cdot (1-m)}{512-k}}{\binom{N}{512}}$$

Assuming ***N*** is fixed at **500,000** validators, the probabilities of randomly sampling a malicious supermajority of the sync committee in an epoch are as follows for various values of ***m***:

m = **33%:** 2.2049650326112223e-54
m = **50%:** 2.3145410602851383e-14
m = **66%:** 0.40667958538739035
m = 70%: *0.9*568626324636733

This estimation implies that when the proportion ***m*** of malicious validators does not exceed one-third, the likelihood of sync committee collusion can be considered improbable to occur in practice.

Note, that if the honest majority assumption does not hold on Ethereum, then cross-chain bridging based on light client architecture becomes extremely vulnerable. The total economic security of the sync committee, i.e. ⅔ of tokens locked by all sync committee members, is capped at 10,923 ETH, which is significantly lower than a typical TVL of an established cross-chain bridge.

Threat Model

Process Applied

The process performed to analyze the system for potential threats and build a comprehensive model is based on the approach first pioneered by Microsoft in 1999 that has developed into the STRIDE model (https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)).

Whilst STRIDE is aimed at traditional software systems, it is generic enough to provide a threat classification suitable for blockchain applications with little adaptation (see below).

The result of the STRIDE classification has then been applied to a risk management matrix with simple countermeasures and mitigations suitable for blockchain applications.

STRIDE Interpretation in the Blockchain Context

STRIDE was first designed for closed software applications in permissioned environments with limited network capabilities. However, the classification provided can be adapted to blockchain systems with small adaptations. The below table highlights a blockchain-centric interpretation of the STRIDE classification:

Spoofing	In a blockchain context, the authenticity of communications is built into the underlying cryptographic public key infrastructure. However, spoofing attack vectors can occur at the off-chain level and within a social engineering paradigm. An example of the former is a Sybil attack where an actor uses multiple cryptographic entities to manipulate a system (wash-trading, auction smart contract manipulation, etc.). The latter usually consists of attackers imitating well-known actors, for instance, the creation of an impersonation token smart contract with a malicious implementation.
Tampering	Similarly to spoofing, tampering of data is usually not directly relevant to blockchain data itself due to cryptographic integrity. It can still occur though, for example through compromised developers of the protocol that have access to deployment keys or through supply chain attacks that manages to inject malicious code or substitutes trusted software that interacts with the blockchain (node software, wallets, libraries).
Repudiation	Repudiation, i.e. the ability of an actor to deny that they have taken action is usually not relevant at the transaction level of blockchains. However, it makes

	sense to maintain this category, since it may apply to additional software used in blockchain applications, such as user-facing web services. An example is the claim of a loss of a private key and hence assets.
Information Disclosure	Information disclosure has to be treated differently at the blockchain layer and the off-chain layer. Since the blockchain state is inherently public in most systems, information leakage here relates to data that is discoverable on the blockchain, even if it should be protected. Predictable random number generation could be classified as such, in addition to simply storing private data on the blockchain. In some cases, information in the mempool (pending/unconfirmed transactions) can be exploited in front-running or sandwich attacks. At the off-chain layer, the leakage of private keys is a good example of operational threat vectors.
Denial of Service	Denial of service threat vectors translates directly to blockchain systems at the infrastructure level. At the smart contract or protocol layer, there are more subtle DoS threats, such as unbounded iterations over data structures that could be exploited to make certain transactions not executable.
Elevated Privileges	Elevated privilege attack vectors directly translate to blockchain services. Faulty authorization at the smart contract level is an example where users might obtain access to functionality that should not be accessible.

STRIDE Classification

The following threat vectors have been identified using the STRIDE classification, grouped by components of the system.

	Spoofing	**Tampering**	**Repudiation**	**Information Disclosure**	**Denial of Service**	**Elevated Privileges**
Light clients: Beacon chain and BEEFY	-	Invalid headers	Lack of accountability of sync committee members	RANDAO value prediction	High gas prices	Unauthorized forced checkpoint
Messages relay / execution	-	Reordering of messages	-	Messages front-running	High gas prices High gas	-

					volatility Encode/Decode issues Lack of incentives Queue overload Censorship	
Bridged asset management	Spoofed tokens registering	-	-	-	Inaccessible funds	Unauthorized access to funds in Agents or Asset Hub
Governance operations	Proposals social engineering	Upgrade of Substrate pallets or Ethereum smart contracts with malicious or malfunctioning code	-	-	Missing replenishment of incentives Removal of Snowbridge pallets from Bridge Hub Pausing mechanisms misusage	Unauthorized access to restricted actions
Externally owned account	Lost account	Pharming/phishing/social engineering	Compromised account	Private key leakage Doxxing/identity disclosure	DOS of infrastructure	Compromised private key

Mitigation Matrix

The following mitigation matrix describes each of the threat vectors identified in the STRIDE classification above, assigning an impact and likelihood and suggesting countermeasures and mitigation strategies. Countermeasures can be taken to identify and react to a threat, while mitigation strategies prevent a threat or reduce its impact or likelihood.

Light clients: Beacon chain and BEEFY

Threat Vector	Impact	Likelihood	Mitigation	Countermeasures
Invalid headers Light clients might opt for the sync committee as their exclusive source for headers, relying solely on their signatures for network state verification. In this case, if a supermajority of the sync committee were to act maliciously, forge a block header and sign it, the light client risks accepting an incorrect network state. More details on the Polkadot forum.	High	Low	Ensure that only finalized headers are imported, which eliminates the risk of accepting forged or invalid blocks as well as resynchronization.	In the event that a discrepancy is detected, the light client must be temporarily suspended.
Unauthorized forced checkpoint Light clients rely on checkpoints to quickly sync with the current state of the network. If an unauthorized entity could trick the light client into accepting a malicious forced checkpoint, it could lead to a variety of issues from spoofing over tampering to denial of service.	High	Low	We recommend ensuring that only authorized entities can set checkpoints, and that these checkpoints are properly validated. Invalid checkpoints must be rejected by the light client. Any network participant must be able to timely observe new forced checkpoints and trace them back to their originating governance	If the checkpoint was not authorized, governance should suspend the smart contract or pallet until a correct checkpoint is set.

			proposals.	
Lack of accountability of sync committee members Sync committee members and BEEFY validators are used for updating light clients with the latest finalized blocks. Light clients do not perform full validation as regular nodes do. If misbehaving validators are not detected, they cannot be slashed, and the light clients are at risk of receiving incorrect or malicious updates. Ethereum's sync committee is composed of 512 validators elected from the large pool of Ethereum validators and employs BLS to aggregate signatures. However, once the signatures are aggregated, it becomes impossible to disaggregate them and determine whether specific validators contributed to the aggregated signature. Misbehaving validators can attempt double-voting or voting for invalid blocks. More details can be found in this forum thread from the Snowbridge team.	High	Low	To trace sync committee votes, one might try to establish a validator node and record the signed contributions. However, this method is not reliable because such a node must always be elected to receive votes. BEEFY votes can be traced easily since they are passed to regular network participants in non-aggregated form.	On Polkadot, it is possible to slash a validator that acted maliciously within the BEEFY protocol using the misbehavior reporting mechanisms.

Polkadot has a comparatively lower number of validators (around a thousand) and currently lacks support for signature aggregation. The sync protocol is more flexible regarding required number of signatures to obtain prior to accepting a block. Polkadot includes the protocol of misbehavior reporting, so BEEFY validators are disincentivized to double-vote or vote for invalid blocks.				
RANDAO value prediction During each sync committee period which lasts about 27 hours, an attacker can attempt bribing future sync committee members. If the attacker managed to predict the outcome of RANDAO, the time window could be prolonged.	Medium	Medium	-	-
High gas prices If gas prices increase significantly, it could become prohibitively expensive to keep the light client up-to-date.	Medium	High	Optimize the implementation of the bridge's components to minimize gas consumption.	Perform cost analysis for different prices of ETH/DOT and potential fluctuations in gas prices.

Messages relay/execution

Threat Vector	Impact	Likelihood	Mitigation	Countermeasures
Reordering of messages Reordering of messages is related to the potential issues due to the alteration of their original order, leading to unintended outcomes.	High	Low	Each transaction should include a sequential nonce, and they must be processed in the sequential order, ensuring that no transaction can be processed out of order.	-
Messages front-running Messages are queued in the mempool on the source chain before being executed and then processed by the relayer to be sent to the target chain. This leads to an extended delay between message visibility and inclusion, increasing the risk of front-running.	High	Medium	A commit/reveal scheme or message encryption might be possible to protect the content of messages.	-
High gas prices High gas prices can make message relay unprofitable.	High	Medium	Optimize the implementation of the bridge’s components to minimize gas consumption.	Perform cost analysis for different prices of ETH/DOT and potential fluctuations in gas prices.
High gas volatility The demand for block space on Ethereum determines its gas price. Since that demand can vary significantly over time, gas prices might	Medium	Medium	Perform a stress profitability estimation with an appropriate business continuity plan (BCP) and add corresponding buffers to gas prices to account for	-

experience high volatility at times, affecting the profitability of message relay. During network congestion, message relay might become unprofitable, leading to delays. Such delays might affect the execution of the transaction, for example prices might have changed, or polls might have ended before a vote has been processed.			volatility. Alternatively, implement a fee system that allows users to pay fees based on a user-defined max gas price, potentially refunding any unused amount.	
Lack of incentives Relayers are rewarded for passing Messages from Ethereum to Polkadot from Sovereign Accounts associated with parachains. In the opposite direction, rewards are transferred by Agents. If their wallets do not have enough funds, there will be missing incentives for relayers to operate.	Medium	Low	Processes for regular and cyclical replenishment of funds in the appropriate accounts could be developed. This should be based on an estimation of the expected cost of operating the bridge over a given period of time.	Monitor actively the accounts holding incentives.
Encode/Decode issues Encode/Decode issue refers to a situation where the relayer receives a large amount of malformed messages, which can lead to slowing down the relayer and hence impact the message relay. Examples could be spam or issues related to the	High	Low	The source chain should be responsible for validating messages. It should verify adherence to the schema, and decodebaility. In addition, a rate-limiting mechanism should be implemented on the source chain, which will protect	Conduct in-depth stress-tests and implement test scenarios based on fuzzing the structure of transmitted messages.

incompatibility of the XCM messages structure after decoding.			relayers from receiving too many messages in a short time.	
Queue overload Inbound or outbound queues may receive a very large amount of data to be transferred in a short period of time, which may lead to their overflow or slow down of processing, which may result in a disruption of the bridge's operation.	High	Low	Implementation of queue management on the source chain, which protects against too many messages through a rate-limiting mechanism.	Conduct in-depth stress tests
Censorship Validators could collude to censor messages related to the Bridge.	High	Low	-	-

Bridged assets management

Threat Vector	Impact	Likelihood	Mitigation	Countermeasures
Spoofed tokens registering Spoofed token registration entails the potential risk of malicious actors generating fraudulent tokens on one blockchain network and trying to register them on another blockchain network via the bridge. These spoofed tokens could be intentionally designed to resemble legitimate tokens, causing confusion and potential exploitation of users, dapps, or DeFi platforms operating within the interconnected networks.	Medium	High	Educate users and implement a list of trusted tokens in the UI, publish and regularly update a blacklist of malicious tokens that can be used by users and wallets	Allow governance to remove tokens
Inaccessible Funds Inaccessible funds denote a situation in which user assets are transferred to the bridge, but unforeseen issues arise, rendering them inaccessible or permanently lost. This scenario may arise due to bugs, vulnerabilities, or misconfigurations in the smart contracts or the infrastructure of the bridge.	High	Low	Internal code reviews, unit testing, integration tests, automatic software engineering tools, audits	Monitor actively the bridge components to detect anomalies and errors
Unauthorized access to funds in Agents or Asset Hub Unauthorized access to	High	Low	Internal code reviews, unit testing, integration tests, automatic software engineering tools,	Monitor actively the bridge components to detect

funds on Agents or the Asset Hub refers to the potential risk of threat actors gaining unauthorized control over assets stored within the bridge's Agents or Asset Hub. A security breach in these components could result in the theft of user funds, disruption of operations, and compromise of the bridge's overall integrity.			audits	anomalies and errors

Governance operations

Threat Vector	Impact	Likelihood	Mitigation	Countermeasures
Proposals social engineering This threat vector encompasses attackers' effort to manipulate or deceive participants within the governance system, leading to the proposal and approval of malicious or undesirable changes to the blockchain bridge or any of the components the bridge uses, such as Bridge Hub or Asset Hub. Attackers may employ social engineering techniques to gain the trust of key decision-makers, generate fraudulent proposals, or exploit vulnerabilities in the governance process to implement harmful changes.	High	Low	Educate users and the community about the project and its scope Offer communication channels that are actively monitored for malicious posts and can be moderated	Follow discussions and be active in the community
Upgrade of Substrate pallets or Ethereum smart contracts with malicious or malfunctioning code This threat revolves around attackers compromising the governance process to introduce upgrades that contain malicious or faulty code into the Substrate pallets or Ethereum smart contracts responsible for Snowbridge's operations.	High	Low	Implement and test the code in independent test suites Document the need for code reviews and audits as part of the approval process Perform internal code reviews, unit testing, integration tests, automatic software engineering tools, audits	Follow discussions and be active in the community

By exploiting weaknesses in the governance model, such attacks can lead to inserted vulnerabilities or backdoors into the bridge's essential components, posing significant risks to its security and functionality.				
Missing replenishment of incentives This threat involves attackers manipulating the governance process to obstruct or delay the replenishment of incentives. By hindering relayers from receiving their rewards, the attackers can undermine the bridge's functionality, potentially causing delays or even temporary shutdowns.	High	Medium	Implement a mechanism to automatically replenish incentives	Actively monitor the accounts holding incentives
Removal of Snowbridge pallets from Bridge Hub Snowbridge pallets are essential bridge components. Attackers may attempt to compromise the governance process to remove these critical pallets from the Bridge Hub parachain.	High	Low	Educate users and the community about the project and its scope	Follow discussions and be active in the community
Pausing mechanisms misusage Snowbridge pallets implement a pausing mechanism that enables governance and the pallet owner to temporary disable operations. Attackers	High	Low	Educate users and the community about the project and its scope	Follow discussions and be active in the community

may attempt to compromise the governance process to pause pallets and stop bridge operations.				
Unauthorized access to restricted actions This threat involves attackers circumventing the governance controls to gain unauthorized access to restricted actions within the blockchain bridge system. These restricted actions may include privileged functions like bridge configuration changes, fund transfers, or system-level modifications. By exploiting vulnerabilities in the governance model, attackers can compromise the bridge's security, integrity, and availability, leading to potential asset theft, manipulation, or disruption of services.	High	Low	Internal code reviews, unit testing, integration tests, automatic software engineering tools, audits	Monitor actively the bridge components to detect anomalies and errors

Externally owned account

Threat Vector	Impact	Likelihood	Mitigation	Countermeasures
Lost account The attacker claims that they own an account and the access to the private key has been lost	Low	Low	Have a clear policy not to refund lost assets or restore privileges	Enforce policy and strictness
Pharming/phishing/social engineering The attacker may manipulate users' or development teams' wallets, lure them to malicious front-ends, manipulate DNS records, or use social engineering to trick users/teams into signing manipulated transactions transferring funds/permissions	Medium	Medium	Educate users and team, protect DNS records, create awareness, offer blacklists with malicious sites, create activity on social channels to build reputable channels, deploy front-ends on IPFS or other decentralized infrastructure	Monitor all systems, monitor communities and impersonations/malicious copies of official channels, communicate attempted pharming/phishing/social engineering, have processes in place to recover from DNS manipulation, attacks on front-ends quickly
Compromised account The attacker claims they are a victim of scapegoating, denying responsibility for their attack	Low	Low	Have a clear policy not to refund lost assets or restore privileges	Enforce policy
Private key leakage Private keys are accidentally shared or logged	Medium	Medium	Educate users and team, ensure private keys are properly handled in wallet software, use	Monitor all systems, have policy in place to rotate keys

			hardware wallets/air-gapped devices, security keys, multi-signatures	
Doxxing/identity disclosure Private data such as the off-chain identity of users disclosed	Low	Medium	Educate users and team, no storage of identity/sensible data in databases that link identity to account addresses, follow privacy regulations and guidelines	-
DOS of infrastructure DOS attack on a validator, relayer, an end user's device/network or on the blockchain node they interact with	Low	Low	Educate users and team, use firewalls, sentry architecture, load balancers, VPNs	Monitor infrastructure, and have processes in place to elastically provision and deploy additional resources
Compromised private key Private keys may be compromised	Medium	Medium	Educate users and team	Monitor all systems, have policy in place to rotate keys