Hyperbridge-ISMP Baseline Security Assurance

Threat model and hacking assessment report

V1.1, 28 June 2024

Cayo Fletcher Smith	cayo@srlabs.de
Haroon Basheer	haroon@srlabs.de
Mostafa Sattari	mostafa@srlabs.de
Rachna Shriwas	rachna@srlabs.de
Regina Biró	regina@srlabs.de

Abstract. This work describes the result of the thorough and independent security assurance audit performed by Security Research Labs on the Hyperbridge project. Security Research Labs is a consulting firm that has been providing specialized audit services in the Polkadot ecosystem since 2019, including the Substrate and Polkadot projects.

During this study, Hyperbridge provided access to relevant documentation and supported the research team effectively. The code in scope from the Hyperbridge implementation was verified to assure that the business logic of the protocol for bridging blockchain networks is resilient to hacking and abuse.

The research team identified several issues ranging from high to info level severity in both the ISMP codebase and its relevant Smart Contract implementation, many of which concerned the protocol consensus and state transition validation logic of counterparty blockchains, runtime configuration and best practice deviation cases in contracts. In cooperation with the auditors, Hyperbridge already remediated a subset of the identified issues.

In addition to mitigating the remaining open issues, Security Research Labs also recommends improving the security design through implementing penalties for trusted actors like Fishermen to prevent misbehavior and incentivizing off-chain participants to detect and report malicious behavior by submitting valid fraud proof messages to the network.

Content

1 Disclaimer

This report describes the findings and core conclusions derived from the audit carried out by Security Research Labs within the agreed-on timeframe and scope as detailed in Chapter 2. Please note that this report does not guarantee that all existing security vulnerabilities were discovered in the codebase exhaustively and that following all evolution suggestions described in Chapter 7 may not ensure all future code to be bug free.

2 Motivation and scope

Blockchains evolve in a trustless and decentralized environment, which by its own nature could lead to security issues. Ensuring availability and integrity is a priority for Hyperbridge as it is a multichain interoperability protocol. As such, a security review of the project should not only highlight the security issues uncovered during the audit process, but also bring additional insights from an attacker's perspective, which the Hyperbridge team can then integrate into their own threat modeling and development process to enhance the security of the product.

Hyperbridge is a chain-agnostic interoperability protocol that enables developers to build applications that can securely interoperate with any blockchain. The Hyperbridge protocol presents developers with a unified interface for the multi-chain that allows for secure and trust-free transfer of assets and data between different blockchains. It is designed to be modular and extensible, allowing for the integration of new chains and protocols as the multi-chain ecosystem grows.

At its core, Hyperbridge implements the ISMP messaging framework. ISMP offers developers a familiar HTTP-like API for facilitating cross-chain requests to initiate specific logic on the counterpart chain. This includes the ability to send arbitrary data to connected chains via POST requests and retrieve application storage information on connected chains through GET requests, with verification through state proofs.

Additionally, Hyperbridge provides a blockchain network that serves as a crypto-economic coprocessor for aggregating interoperability proofs built using the Substrate framework. Like other Substrate-based blockchain networks, it is written in Rust, a memory safe programming language.

Mainly, Substrate-based chains utilize three technologies: a WebAssembly (WASM) based runtime, decentralized communication via libp2p, and a block production engine. The Hyperbridge runtime consists of multiple modules compiled into a WASM Binary Large Object (blob) that is stored on-chain. Nodes execute the runtime code either natively or will execute the on-chain WASM blob.

Security Research Labs collaborated with the Hyperbridge team to create an overview containing the runtime modules in scope and their audit priority. The in-scope components and their assigned priorities are reflected in Table 1. During the audit, Security Research Labs used a threat modelling to guide efforts on exploring potential security flaws and realistic attack scenarios.

Repository	Priority	Component(s)	Reference
polytope-labs/hyperbridge	High	ISMP	[1]
	High	Nexus runtime	[2]
polytope-labs/evm-solidity	High	handlerV1.sol Evmhost.sol	[3]

Table 1 In-scope Hyperbridge components with audit priority

3 Methodology

This report details the baseline security assurance results for the Hyperbridge parachain with the aim of creating transparency in four steps, treat modeling, security design coverage checks, implementation baseline check and finally remediation support:

1. Threat Modeling. The threat model is considered in terms of *hacking incentives*, i.e., the motivations to achieve the goals of breaching the integrity, confidentiality, or availability of Hyperbridge parachain nodes. For each hacking incentive, hacking *scenarios* were postulated, by which these goals could be achieved. The threat model provides guidance for the design, implementation, and security testing of Hyperbridge. Our threat modeling process is outlined in Chapter 4.

2. Security design coverage check. Next, the Hyperbridge design was reviewed for coverage against relevant hacking scenarios. For each scenario, the following two aspects were investigated:

a. **Coverage**. Is each potential security vulnerability sufficiently covered?

b. **Underlying assumptions**. Which assumptions must hold true for the design to effectively reach the desired security goal?

3. Implementation baseline check. As a third step, the current Hyperbridge implementation was tested for openings whereby any of the defined hacking scenarios could be executed.

To effectively review the Hyperbridge codebase, we derived our code review strategy based on the threat model that we established as the first step. For each identified threat, hypothetical attacks were developed and mapped to their corresponding threat category, as outlined in Chapter 4.

Prioritizing by risk, the codebase was assessed for present protections against the respective threats and attacks as well as the vulnerabilities that make these attacks possible. For each threat, the auditors:

1. Identified the relevant parts of the codebase, for example the relevant pallets and the runtime configuration.
2. Identified viable strategies for the code review. Manual code audits, fuzz testing, and manual tests were performed where appropriate.
3. Ensured the code did not contain any vulnerabilities that could be used to execute the respective attacks, otherwise, ensured that sufficient protection measures against specific attacks were present.
4. Immediately reported any vulnerability that was discovered to the development team along with suggestions around mitigations.

We carried out a hybrid strategy utilizing a combination of code review and dynamic tests (e.g., fuzz testing) to assess the security of the Hyperbridge codebase.

While fuzz testing and dynamic tests establish a baseline assurance, the focus of this audit was manual code review of the Hyperbridge codebase to identify logic bugs, design flaws, and best practice deviations. We reviewed the Hyperbridge repository up to the commit *922aafb.* The approach of the review was to trace the intended functionality of the runtime modules in scope and to assess whether an attacker can bypass/misuse/abuse these components or trigger unexpected behavior on the blockchain due to logic bugs or missing checks. Since the Hyperbridge codebase is entirely open source, it is realistic that a malicious actor would analyze the source code while preparing an attack.

Fuzz testing is a technique to identify issues in code that handles untrusted input, which in Hyperbridge's case is extrinsics in the runtime. (Note that the network part is handled by Substrate, which was not in scope for this review, but is built with a strong emphasis on security and where fuzz testing is also used). Fuzz testing works by taking some valid input for a method under test, applying a semi-random mutation to it, and then invoking the method under test again with this semi-valid input. Through repeating this process, fuzz testing can unearth inputs that would cause a crash or other undefined behavior (e.g., integer overflows) in the method under test. The fuzz testing methods written for this assessment utilized the test runtime Genesis configuration as well as mocked externalities to execute the fuzz test effectively against the extrinsics in scope.

4. Remediation support. The final step is supporting Hyperbridge with the remediation process of the identified issues. Each finding was documented and published with mitigation recommendations. Once the mitigation solution is implemented, the fix is verified by the auditors to ensure that it mitigates the issue and does not introduce other bugs.

During the audit, findings were shared via a private GitHub repository [4]. We also used a private Telegram channel for asynchronous communication and status updates.

4 Threat modeling and attacks

The goal of the threat model framework is to be able to determine specific areas of risk in Hyperbridge's blockchain system. Familiarity with these risk areas can provide guidance for the design of the implementation stack, the actual implementation of the stack, as well as the security testing. This section introduces how risk is defined and provides an overview of the identified threat scenarios. The *Hacking Value*, categorized into *low*, *medium*, and *high*, considers the incentive of an attacker, as well as the effort required by an attacker to successfully execute the attack. The hacking value is calculated as:

$$Hacking\ Value = \frac{Incentive}{Effort}$$

While *incentive* describes what an attacker might gain from performing an attack successfully, *effort* estimates the complexity of this same attack. The degrees of incentive and effort are defined as follows:

Incentive:

- Low: Attacks offer the hacker little to no gain from executing the threat.
- Medium: Attacks offer the hacker considerable gains from executing the threat.
- High: Attacks offer the hacker high gains by executing this threat.

Effort:

- Low: Attacks are easy to execute. They require neither elaborate technical knowledge nor considerable amounts of resources.
- Medium: Attacks are difficult to execute. They might require bypassing countermeasures, the use of expensive resources or a considerable amount of technical knowledge.
- High: Attacks are difficult to execute. The attacks might require in-depth technical knowledge, vast amounts of expensive resources, bypassing countermeasures, or any combination of these factors.

Incentive and Effort are divided according to Table 2.

Hacking Value	Low incentive	Medium Incentive	High Incentive
High effort	Low	Medium	Medium
Medium effort	Medium	Medium	High
Low effort	Medium	High	High

Table 2. Hacking value measurement scale.

Hacking scenarios are classified by the risk they pose to the system. The risk level, also categorized into low, medium, and high, considers the hacking value, as well as the damage that could result from successful exploitation. The risk of a threat scenario is calculated by the following formula:

$$Risk = Damage \times Hacking\ Value = \frac{Damage \times Incentive}{Effort}$$

Damage describes the negative impact that a given attack, performed successfully, would have on the victim. The degrees of damage are defined as follows:

Damage:

- Low: Risk scenarios would cause negligible damage to the Hyperbridge network
- Medium: Risk scenarios pose a considerable threat to Hyperbridge's functionality as a network.
- High: Risk scenarios pose an existential threat to Hyperbridge's network functionality.

Damage and Hacking Value are divided according to Table 3.

Risk	Low hacking value	Medium hacking	High hacking value
Low damage	Low	Medium	Medium
Medium damage	Medium	Medium	High
High damage	Medium	High	High

Table 3. Risk measurement scale

After applying the framework to the Hyperbridge system, different threat scenarios according to the CIA triad were identified.

The CIA triad describes three security promises that can be violated by a hacking attack, namely confidentiality, integrity, availability.

Confidentiality:

Confidentiality threat scenarios concern sensitive information regarding the blockchain network and its users. Native tokens are units of value that exist on the blockchain - confidentiality threat scenarios include for example attackers abusing information leaks to steal native tokens from nodes participating in the Hyperbridge ecosystem and claiming the assets (represented in the token) for themselves.

Integrity:

Integrity threat scenarios threaten to disrupt the functionality of the entire network by undermining or bypassing the rules that ensure that Hyperbridge transactions/operations are fair and equal for each participant. Undermining Hyperbridge's integrity often comes with a high monetary incentive, like for example, if an attacker can double spend or mint tokens for themselves. Other threat scenarios do not yield an immediate monetary reward, but rather, could threaten to damage Hyperbridge's functionality and, in turn, its reputation. For example, replaying bridged transactions would violate the core promise that transactions on the bridge are not replayable.

Availability:

Availability threat scenarios refer to compromising the availability of data stored by the Hyperbridge network as well as the availability of the network itself to process normal transactions. Important threat scenarios regarding availability for blockchain systems and bridges in particular include Denial of Service (DoS) attacks, stalling the bridge operations, and spamming.

Table 4 provides a high-level overview of the hacking risks concerning Hyperbridge with identified example threat scenarios and attacks, as well as their respective hacking value and effort. The complete list of threat scenarios identified along with attacks that enable them are described in the threat model deliverable. This list can serve as a starting point to the Hyperbridge developers in order to guide their security outlook for future feature implementations. By thinking in terms of threat scenarios and attacks during code review or feature ideation, many issues can be caught or even avoided altogether.

For Hyperbridge, the auditors attributed the most hacking value to the integrity class of threats. Since the efforts required to exploit this kind of issue is considered lower, we identified threat scenarios to the integrity of Hyperbridge as of the highest risk category. Undermining the integrity of the Hyperbridge chain means making unauthorized modifications to the system. Some of the scenarios can have a direct effect on the financials of the system. This can include griefing attacks by timeout messages, replaying ISMP messages, gaining tokens through message failures or double spending.

Security promise	Hacking value	Example threat scenarios	Hacking effort	Example attack ideas
Confidentiality	High	- Take over another user's account	High	- Social engineering
Integrity	High	- Replay ISMP messages - Modify the request and response receipts - Abuse bridge transaction failures for financial profit	Medium	- Exploit a bug in validation scheme to accept an invalid transaction - Abuse different encoding schemes across the chains to perform malicious activity

Availability	Mediu m	- Stall the bridge operations - DoS the ISMP handlers - Harm the chain functionality by cluttering its storage	Low	- DoS the Fishermen/Relayers by spamming them with fraudulent messages - Cheaply fill up blockchain storage - Transaction spam attacks by users

Table 4. Risk overview. The threats for Hyperbridge were classified using the CIA security triad model, mapping threats to the areas: (1) Confidentiality, (2) Integrity, and (3) Availability.

5 Findings summary

We identified 13 issues - summarized in Table 5- during our analysis of the ISMP module in scope in the Hyperbridge Rust codebase that enable some of the attacks outlined above. In summary, 6 high severity, 3 medium severity, 1 low severity and 3 info severity issues were found.

Please note that in our methodology, critical severity issues refer to high severity issues that could be exploited immediately by an attacker on already deployed infrastructure, including a parachain or a non-incentivized testnet.

Issue	Severity	References	Status
Nexus runtime waives XCM message delivery fee	High	[5]	Risk accepted
No XCM delivery fees configured for sibling parachain messages	High	[6]	Risk accepted
Stack exhaustion due to missing *DecodeLimit*	High	[7]	Fixed [8]
Incorrect weight configuration in Nexus runtime	High	[9]	Fix in progress [8]
Missing runtime benchmark for ISMP pallets	High	[10]	Fix in progress [8]
Unsigned extrinsics allow executing ISMP messages for free	High	[11]	Closed
Fishermen can veto *StateCommitment* after the *challenge_period*	Medium	[12]	Risk accepted
Missing priority in Fraud Proof messages	Medium	[13]	Risk accepted
Same consensus client can be frozen repeatedly	Medium	[14]	Fixed [15]
Missing benchmarking to calculate weights	Low	[16]	Risk accepted
Unbounded loop leading to block production stalling	Info	[17]	Closed

Timeout overflow due to unsafe arithmetic in request dispatch	Info	[18]	Fixed [19]
Missing fraud proof implementation for Beacon consensus client	Info	[20]	Risk accepted

Table 5 Issue summary for the ISMP Rust codebase

We identified 5 security vulnerabilities and 3 code efficiency issues – summarized in Table 6 – throughout our investigation of the ISMP Solidity modules *EvmHost* and *HandlerV1*. Our analysis concluded with 1 high severity, 1 medium severity, 3 low severity, and 3 gas optimization issues.

Issue	Severity	References	Status
Transfer failure could cause unexpected behavior and financial loss	High	[21]	Fixed [22]
Host configuration can be permanently locked by denial of service	Medium	[23]	Fixed [22]
Configuration could be locked due to missing zero-address sanity checks	Low	[24]	Fixed [22]
Configuration could be locked due to the finality of updates	Low	[25]	Open
Missing event emissions would cause incomplete off-chain monitoring	Low	[26]	Fixed [22]
Gas optimization report	Info-Gas	[27] [28] [29]	Fixed [22]

Table 6 Issue summary for the Solidity contracts

6 Detailed findings

6.1 Nexus runtime waives XCM message delivery fee

Attack scenario	**An attacker can send XCM messages without paying any delivery fee**
Location	parachain/runtime/nexus
Tracking	[5]
Attack impact	An attacker can cause congestion, storage exhaustion and/or dropping of messages
Severity	High
Status	Risk accepted

The Nexus runtime does not set the *FeeManager* for XCM configuration waiving all fees for the XCM messages. An attacker can exploit this to cause congestion, possibly leading to long delivery delays, storage exhaustion and/or dropping of messages.

We recommend following the fix implemented in the polkadot runtime update v1.3 [30] which introduces a proper implementation for the *FeeManager*.

The issue was acknowledged by the Hyperbridge team, and the risk accepted since XCM is not intended to be used in Hyperbridge for parachain messaging.

6.2 No XCM delivery fees configured for sibling parachain messages

Attack scenario	**An attacker can send XCM messages across parachains without paying any delivery fee**
Location	parachain/runtime/nexus
Tracking	[6]
Attack impact	An attacker can send spam messages to cause XCM queue size exhaustion and delay in message delivery
Severity	High
Status	Risk accepted

The Nexus runtime configuration sets the *PriceForSiblingDelivery* to *NoPriceForMessageDelivery* implying that there are no fees charged for XCM messages across parachains.

An attacker can exploit this vulnerability, to send spam messages across chains without paying any fee. Excessive messages could lead to XCM queue size exhaustion by excessive storage usage until messages are delivered. This could also lead to delays in message delivery for other users.

We recommend charging adequate message delivery fees in the runtime configuration. An exponential fee mechanism such as in Kusama [31], should be used to prevent excessive delivery time and storage exhaustion.

The issue was acknowledged by the Hyperbridge team, and the risk accepted since XCM is not intended to be used in Hyperbridge for parachain messaging.

6.3 Stack exhaustion due to missing *DecodeLimit*

Attack scenario	**The extrinsic *decompress_call* in *pallet_call_decompressor can be called without a decode depth limit***
Location	modules/ismp/pallets/call-decompressor
Tracking	[7]
Attack impact	An attacker can cause stack exhaustion and denial of service.
Severity	High
Status	Fixed [8]

The extrinsic *decompress_call* in *pallet-call-decompressor* accepts a compressed call as an argument which then is decoded without a depth limit.

An attacker can exploit this vulnerability to create deeply nested calls that may exhaust the stack size during decoding due to excessive recursion. If this call is sent through a bridge that requires forced execution, it can permanently stall the receiving chain.

We recommend using *DecodeLimit* [32], which accepts a nesting level.

The issue was fixed by the Hyperbridge team. The remediation included fixing 2 parts: *decode_with_depth_limit* [8] and fine tuning the maximum depth for the decoding that is custom to Hyperbridge runtime [33].

6.4 Incorrect weight configuration in Nexus runtime

Attack scenario	**Incorrect weight configuration in the Nexus runtime can be exploited for zero cost execution**

Location	parachain/runtimes/nexus
Tracking	[9]
Attack impact	An attacker may spam and bloat the network storage
Severity	High
Status	Fix in progress [8]

The weights for the ISMP, XCM and other Substrate pallets are not configured correctly in the Nexus runtime.

The following misconfigurations were identified:

1. pallets using *TestWeightInfo*
2. pallet runtime weights set to Zero with *type WeightInfo = ()*
3. weights configured using substrate-node template

Setting the weights to *()* effectively makes it a zero-cost execution for extrinsics. This leads to an attacker spamming and bloating network storage freely using an underweight extrinsic.

We recommend configuring the weights for the custom and FRAME pallets using the actual Nexus runtime configuration. An example for configuring weights with the actual runtime be found in the Asset Hub Polkadot parachain runtime [34].

The issue was partially mitigated in PR #234 [8], by using the weights generated by the benchmark. The missing updated weights for *pallet-collator-selection* will be added by the Hyperbridge team after the upgrade to a higher polkadot-sdk version.

6.5 Missing runtime benchmark for ISMP pallets

Attack scenario	**An attacker can spam Hyperbridge with underweight extrinsics**
Location	parachain/runtimes/nexus
Tracking	[10]
Attack impact	Incorrect benchmarking can lead to spamming, storage bloating and block stalling
Severity	High
Status	Fix in progress [8]

The configurable weights of the following pallets in the ISMP module are not included in the runtime benchmarks of the Nexus runtime:

- *pallet-assets*
- *pallet-ismp*
- *cumulus-pallet-parachain-system*
- *pallet-message-queue*
- *pallet-sudo*
- *pallet-xcm*

Excluding pallets with configurable weights from benchmarking may result in overweight or underweight extrinsics. This potentially leads to low-effort attacks such as spamming, storage bloating, and block stalling when invoking extrinsics.

We recommend including the above-mentioned pallets in the *try_benchmarks!* macro for appropriate runtime benchmarks.

The issue was partially mitigated in PR #234 [8] by adding the pallets in the runtime benchmarks, except for *pallet-xcm* which needs more changes to be included in the benchmarking logic.

6.6 Unsigned extrinsincs allow to execute ISMP messages for free

Attack scenario	**An attacker can use unsigned extrinsics to submit ISMP messages for free**
Location	modules/ismp
Tracking	[11]
Attack impact	An attacker can spam the network without facing any consequence
Severity	High
Status	Closed

The ISMP messages are implemented as unsigned extrinsics, which allows a malicious attacker to freely execute these messages.

Aside from the spamming risk considered, an attacker can exploit the extrinsic design flaw without facing any punishments or blacklisting.

We recommend implementing a check in the extrinsic such that only messages of type *FraudProofMessages* are accepted, and other message types (Consensus, Response, Request and Timeout) are blocked from being handled at *handle_incoming_message.*

It was clarified by the Hyperbridge team that unsigned transactions are checked for validity before adding to the transaction pool thus preventing spamming the transaction pool and the issue was closed. It was later identified in Finding 6.9, that this can be circumvented if valid *FraudProofMessages* are sent repeatedly. Since the messages don't have a nonce, they will be added to the transaction pool allowing for spamming.

6.7 Fishermen can veto *StateCommitment* after the *challenge_period*

Attack scenario	**Fishermen as trusted actors can veto a state commitment even after the challenge period is elapsed**
Location	modules/ismp/pallets/fishermen
Tracking	[12]
Attack impact	The integrity and trustworthiness of the system might be compromised if errors or fraud are discovered after the challenge period
Severity	Medium
Status	Risk accepted

Fishermen are trusted actors in the network that check if the *StateCommitment* describes a valid state transition on the counterparty network. If the *challenge_period* elapses without any veto from Fishermen, it can be safely concluded that the provided *StateCommitment* are indeed canonical.

However, in the implementation, Fishermen may veto the *StateCommitment* without checking whether the *challenge_period* has elapsed in the extrinsic *veto_state_commitment.*

Any attempt to veto the state commitment after the challenge period would require expensive solutions such as rolling back the blockchain state. The integrity and trustworthiness of the system might be compromised if errors or fraud are discovered after the challenge period.

Since Fishermen are not required to submit any fraud proofs, this allows compromised or rogue Fishermen to repeatedly veto the canonical chain's *StateCommitment* creating instability in the system and still gain rewards for the veto.

As a safety measure, implement additional safeguards for the Fishermen before vetoing any *StateCommitment.* Handle and implement the timeout error as described in the design document: *"Assert that the configured challenge_period for the StateCommitment has elapsed"* [35].

The issue was acknowledged by the Hyperbridge team, and the risk accepted to allow vetoes occurring past the challenge period because of slow block propagation through the network.

6.8 Missing priority in Fraud Proof messages

Attack scenario	**A fraud proof message may be censored or delayed for processing**
Location	modules/ismp
Tracking	[13]
Attack impact	A malicious consensus client can inflict more faults into the consensus mechanism until frozen
Severity	Medium
Status	Risk accepted

ISMP supports different message types including Fraud Proof messages where they are handled with same priority as other types of ISMP messages on the chain. This can lead to a Fraud Proof message being censored or getting delayed in processing if there are multiple unhandled ISMP messages in the queue.

Furthermore, a malicious consensus client inflicting more faults into the consensus mechanism. They will also not be frozen promptly for their byzantine behaviour if there are no incentive for Fraud Proof messages submission.

We recommend considering the following:

- Fraud Proof messages should be handled with a high priority over other ISMP messages. This can be achieved by handling *FraudProofMessage* separately from other messages in the *validate_unsigned* and assigning a high priority in *ValidTransaction.*

- Incentivize Fraud Proof messages by for example, implementing rewarding the submitter or refunding the protocol fee, after the freeze is in effect. This can be done if the *freeze_client* results in *MessageResult::FrozenClient.* Incentivizing Fraud Proofs will encourage users to submit Fraud Proof and help in freezing the malicious consensus client in time.

The issue was acknowledged by the Hyperbridge team, and the risk accepted since the current implementation handles *FraudProofMessages* in a batch with the other messages.

6.9 Same consensus client can be frozen repeatedly

Attack scenario	**An attacker can submit Fraud Proof for the same consensus client repeatedly**
Location	modules/ismp
Tracking	[14]
Attack impact	1. A consensus client can be frozen repeatedly 2. A malicious user can spam the chain by sending Fraud Proof messages for free
Severity	Medium
Status	Fixed [15]

When a valid Fraud Proof message is submitted through an extrinsic, the consensus client can be frozen for their byzantine behaviour. Since Fraud Proof messages do not contain a nonce, when submitted through unsigned calls, they will be processed again to freeze the same client. As a security measurement, a hashing mechanism in place that prevents duplicate messages. However, as the hash represents the hash of all messages in the batch, a malicious actor can easily circumvent this by submitting different batches of different lengths and therefore, producing different hashes.

The lack of checks for already submitted Fraud Proofs for the same consensus client via the unsigned call *handle_unsigned* or the singed *handle* call, exposes the following risks:

- The same valid Fraud Proof can be submitted multiple times to repeatedly freeze the same consensus client.

- Since anyone is allowed to submit unsigned extrinsics in *pallet-hyperbridge*, a malicious user may spam the chain via repeatedly submitting the same valid Fraud Proof for free [36].

We recommend adding an additional protection in *freeze_client* through *is_consensus_client_frozen* such that Fraud Proof message for already frozen client is rejected.

The issue was fixed by the Hyperbridge team by adding the check as suggested [15].

6.10 Missing benchmarking to calculate weights

Attack scenario	**Underweight extrinsics can be utilized to spam the network**
Location	modules/ismp/pallets
Tracking	[16]
Attack impact	An attacker may perform denial of service attack cheaply in comparison to the actual weight.
Severity	Low
Status	Risk accepted

The weights of the extrinsics in pallets *call-decompressor*, *fishermen* and *relayer* are not based on benchmark values and have a fixed value of *1_000_000.*

The affected extrinsics are:

- *decompress_call*
- *add*
- *remove*
- *veto_state_commitment*
- *accumulate_fees*
- *withdraw_fees*

Incorrect fee calculation due to lack of benchmarking can aid an attacker to exploit low fees to the flood the network with transactions. Unsigned extrinsics can be particularly vulnerable to DoS attacks since they don't require authentication and can be submitted in large numbers to overwhelm the network.

We recommend benchmarking all signed extrinsics to reflect the accurate estimation. The issue was partially fixed and acknowledged [37].

6.11 Unbounded loop leading to block production stalling

Attack scenario	**The call to *dispatch_to_evm* can result in an unbounded loop**
Location	modules/ismp/pallets/pallet
Tracking	[17]
Attack impact	The extrinsics can be abused to halt block production on the chain
Severity	Info
Status	Closed

The pallet ismp-demo allows dispatching request messages to EVM chains via *dispatch_to_evm* extrinsic. However, this extrinsic has an unbounded loop that can be abused to halt block production on the chain.

There are two ways of fixing this issue, and ideally both should be implemented to assure maximal safety:

1. Implement benchmarking and make the benchmark dependent on count
2. Bound the value of count to a reasonable range

The issue was marked as 'Closed' after the Hyperbridge team clarified that the demo pallet will not be used in the production network.

6.12 Timeout overflow due to unsafe arithmetic in request dispatch

Attack scenario	**Integer overflow can be triggered by setting a very high number in the timeout field making it represent a timeout in the past**
Location	modules/ismp/pallets/demo
Tracking	[18]
Attack impact	The request will expire before it is even sent out. In future, if bridge operators are penalized for timeouts, it can also affect the availability of the network
Severity	Info

Status	Fixed [19]

When a request is dispatched over ISMP, a malicious actor could trigger an integer overflow in the *timeout_timestamp* computation logic.

The *timeout_timestamp* will then represent a timestamp in the past, and the request will have expired before it was even sent-out. This can be triggered using *pallet-ismp-demo*'s *get_request* extrinsic by setting a very high number in the timeout field of the params struct.

Considering the current design and implementation of the Hyperbridge project, we do not see any security risk arising in case the timeout overflow is triggered. In the future, if the bridge operators can be penalized for timeouts, this issue could become problematic, affecting the availability of the network.

We recommend using safe arithmetic (e.g. saturating math) on the *timeout_timestamp* operation. As a generic defensive programming suggestion, we recommend using safe arithmetic throughout the codebase.

The issue was fixed by the Hyperbridge team by adding *saturating_add* for the calculation [19].

6.13 Missing Fraud Proof implementation for Beacon consensus clients

Attack scenario	**Fraud Proof message submitted on the Beacon client will return an unimplemented error and the proof will be rejected**
Location	modules/ismp/clients/sync-committee
Tracking	[20]
Attack impact	The Beacon consensus client will not be frozen after an invalid consensus state resulting in multiple invalid transactions
Severity	Info
Status	Risk accepted

Hyperbridge supports different consensus clients where each instance should implement the *ConsensusClient* trait. One fundamental function from this trait is the *verify_fraud_proof* which is used to verify a Fraud Proof.

However, for the Beacon consensus client this function is not implemented. Any attempt to submit Fraud Proof messages for the Beacon chain through the consensus client will thus return an error and get rejected. Hence, the Beacon consensus client will not be frozen and the Relayer will continue to relay request and response messages to the Beacon chain resulting in multiple invalid transactions.

We recommend implementing Fraud Proof verification for Beacon consensus clients.

The issue was acknowledged by the Hyperbridge team, and the risk accepted, planning to rely on Fishermen as a contingency plan.

6.14 Failure of transfer success may result in unexpected behavior

Attack scenario	**An attacker, using an account with insufficient ERC-20 tokens can invoke functionality that requires the transfer of funds**
Location	evm/src/hosts/EvmHost.sol
Tracking	[21]

Attack impact	System behavior would be undefined and may result in financial loss
Severity	High
Status	Fixed [22]

Throughout the codebase there are multiple instances of unchecked return values when interacting with the *transfer()* and *transferFrom()* functions of an external ERC-20 contract. These missing checks make it impossible to halt execution upon failed transfers.

In the case of *fundRequest()* and *fundResponse(),* the funding status and value would be recorded in *metadata.fee* irrespective of the aforementioned *transferFrom()* succeeding.

We recommend ensuring that all calls to external ERC-20 contracts have appropriate return value checks. Furthermore we recommend integrating the use of the SafeERC20 library from Open Zeppelin [38] to appropriately handle arbitrary ERC-20 tokens which may either revert on failure or return false.

This issue was fixed in [22] by Hyperbridge by implementing the safeERC20 library, and appropriately checking the result of transfer.

6.15 Denial of service to host configuration updates

Attack scenario	**Configuration update forces too many entries into fisherman array, causing denial of service on iteration via out-of-gas**
Location	evm/src/hosts/EvmHost.sol
Tracking	[23]
Attack impact	The host configuration *_hostParams* would be completely locked and unchangeable
Severity	Medium
Status	Fixed [22]

When the host configuration *_hostParams* is updated, the logic iterates over an unbounded storage array *_hostParams.fishermen*. Each address entry inside the array is read, matched to the key value pair, and then deleted from the *_fishermen* mapping.

This process serves to remove old entries from the *_fishermen* mapping, before updating the *_hostParams.fishermen* array, and reinitializing the new entries (via iteration) into the *_fishermen* mapping.

If the entries inside *_fishermen* are inadvertently set too high by the host manager, it would be impossible to subsequently update any configuration parameters since execution will revert with out-of-gas on the initial iteration process.

We recommend bounding the number of entries inside the *_hostParams.fishermen* array to some safe defined threshold. Alternatively, if a large volume of Fishermen addresses is required for system functionality, implement appropriate mechanisms to segment the update process over multiple blocks.

This issue was fixed in [22] by Hyperbridge by implementing input bounding for the *_hostParams.fishermen* array.

6.16 Configuration could be locked without zero-address sanity checks

Attack scenario	**Misconfiguration could set privileged roles to unusable addresses**
Location	evm/src/hosts/EvmHost.sol
Tracking	[24]
Attack impact	Critical functionality such as configuration updates would be permanently unreachable with no remediation
Severity	Low
Status	Fixed [22]

When updating the host parameters configuration struct via *updateHostParamsInternal()*, the critical config data *_hostParams* is directly updated using the input struct *params*.

Misconfiguration of the *hostManager* data field within the input *params* would result in *updateHostParamsInternal()* being unreachable due to the *onlyManager* access control modifier, thereby permanently locking all host configurations.

Privileged roles, especially *hostManager* should be safeguarded from misconfiguration to the zero-address. We recommend integrating address zero input filtering to ensure that data fields storing privileged roles such as *hostManager* have appropriate sanity checks.

This issue was resolved in [22] by the Hyperbridge team by implementing zero address and contract existence checks for the host manager address.

6.17 Configuration could be locked due to the finality of updates

Attack scenario	**Misconfiguration in updates could set privileged roles to insufficient participants with absolute finality**
Location	evm/src/hosts/EvmHost.sol
Tracking	[25]
Attack impact	General usage functionality alongside critical processes would be permanently unreachable with no remediation.
Severity	Low
Status	Risk accepted

Similar to the concerns and rationale expressed in 6.16: the finality of single-step updates to the privileged *hostManager* contract address increases the attack surface to include misconfiguration and human error.

Such misconfiguration could result from overlooking critical functionality required for the new *hostManager* contract address to interact sufficiently with the host. Such mistakes would be unrectifiable, potentially locking certain functionality in the host, in-line with the misconfiguration's severity.

We recommend implementing a 2-step, propose and accept, solution to offer appropriate escape clauses to privilege transfer. In this mitigation the current manager should propose a *pendingManager* that must be claimed by the recipient before full privilege transferal occurs.

6.18 Missing event emissions would impede off-chain monitoring

Attack scenario	**The exploitation of unknown future edge-cases could occur unnoticed**
Location	evm/src
Tracking	[26]
Attack impact	The inability to efficiently triage live-exploitations in a timely manner would amplify the associated impact of any given exploit.
Severity	Low
Status	Fixed [22]

Throughout the codebase we noticed various instances of unsatisfactory support for off-chain monitoring solutions, in that the transmission of necessary system data was often neglected.

Shipping data off-chain is important in most smart contract systems to maintain continued security, although in cross-chain applications, such as Hyperbridge, it should be considered as a critical design decision. We argue this, since cross-chain exploit remediation is often more challenging, especially if caught late into the attack lifecycle.

We recommend integrating more event emissions throughout all system functionality, and indexing event parameters that should be reasonably searchable by off-chain tools. Such parameters would include participant addresses and cross-chain messages.

Hyperbridge implemented this guidance in [22], thereby increasing their ability to trace an attacker's actions and improve response times to edge-cases.

6.19 Gas optimization report

Impact	Inefficient Solidity source code often results in higher execution costs which may impact community sentiment
Location	evm/src
Tracking	[27] [28] [29]
Severity	Info – Gas
Status	Fixed [22]

Throughout our investigation we found multiple instances where source code logic could be optimized to improve runtime transaction fees for system participants.

We recommend incrementing loop iterators with ++i instead of i++. This improvement, reported in [27], avoids returning the incremented iterator in a secondary temporary variable, which saves gas due to the reduced compiled complexity.

When comparing unsigned integers to zero, as detailed in [28], the not-equal-to operator (!=) is more gas efficient than the greater-than operator (>). This improvement has no impact on execution logic since unsigned integers are non-negative.

Furthermore, as expressed in [29], reverting directly in conditional logic is more efficient than relying on the built-in conditional checks of *require* statements.

Finally, reverting with custom errors also significantly reduces gas over emitting lengthy string descriptions.

Execution costs and source code should be optimized to ensure more positive community sentiment and reduce operational overheads from privileged accounts.

Instance	Description	Saving
Iterator incrementation **[27]**	Use prefix (++i) not postfix (i++) iterator incrementation in loops.	5*i*-gas
Zero comparison **[28]**	Revert with custom errors inside conditional logic not require statements with string descriptions.	220-gas
Reversion logic **[29]**	Use != instead of > when comparing unsigned integers to zero.	22-gas

7 Evolution suggestions

The overall impression of the auditors was that ISMP as a messaging protocol is designed and implemented with protection against typical blockchain bridge threats and attacks such as spamming and message replaying in mind. However, to ensure that the ISMP messaging framework is secure against rogue and collusion of trusted network participants, we recommend considering the evolution suggestions and best practices described in this section.

7.1 Business logic improvement suggestions

We recommend following design suggestions to the ISMP codebase to improve the overall security posture of the protocol:

Implement penalty for Fishermen's misbehavior. To deter rogue behavior from trusted actors like Fishermen, they should be subject to slashing or blacklisting for any misbehavior. Actions such as vetoing a valid state commitment within the challenge period window through collusion with other Fishermen or after the challenge period elapsed to cause state roll back should be punished. Implementing slashing or backlisting will ensures that long term robustness and security guarantee is enshrined through the trusted actors like Fishermen. As the design of the Fishermen evolves to a permissionless entity-[39], slashing and blacklisting mechanisms will become critical safety features to deter malicious behavior.

Implement Fraud Proof submission requirements for Fishermen. Considering the permissionless nature and potential incentivization for the Fishermen role in the future, it is recommended to make the vetoing of Fishermen based on verifiable Fraud Proofs, as such any state commitments vetoed by permissionless Fishermen are backed by valid Fraud Proofs, such that honest behavior is promoted to secure the network. Consider implementing Fraud Proof requirements for vetoes from permissionless Fishermen in the later iterations of the ISMP protocol for a resilient security model.

Implement refund of protocol fees for valid Fraud Proof messages. In the current ISMP design, Fraud Proof messages are submitted by anyone through an unsigned extrinsic with valid proofs for detecting invalid state transition which could result in freezing of the consensus client. A payment is required for any ISMP messages as protocol fee at the source chain to prevent spamming, this could also deter honest participant form submitting a valid proof. As the protocol continues to evolve to provide bridging across different blockchain ecosystems, Fraud Proof messages form the tenet of security for Hyperbridge. As such, submission of valid Fraud Proof messages should be encouraged and even rewarded to make the entire ecosystem robust and mature. Implement refund or reward for Fraud Proof messages that were submitted through signed extrinsics that resulted in freezing of misbehaving clients.

7.2 Address currently open security issues

We recommend addressing known security issues in a timely manner to prevent attackers from exploiting them – even if an open issue has a limited impact, an attacker might use it as part of their exploitation chain, which may have a more severe impact on Hyperbridge.

Implement weight benchmarking. Introduce and regularly update benchmarking to measure and analyze the execution time and resource consumption of different runtime functions based on their weights. Kindly refer the Polkadot runtimes for benchmarking best practices [40] and weight configuration [41].

Prioritize Fraud Proof messages for timely processing. To mitigate the risk of Fraud Proof messages being delayed due to high message volume, introduce a priority mechanism for these messages. This will ensure that the system can swiftly detect and address fraudulent activities, thereby safeguarding the integrity and security of the blockchain network.

Implement Fraud Proof Message verification for Beacon consensus clients. Implement *verify_fraud_proof* for the Beacon consensus client so that the consensus client can be frozen in case of an invalid consensus state and further invalid transactions can be prevented.

Improve the update granularity for configuration parameters. As mentioned in sections 6.16 and 6.17 the update mechanisms for privileged roles are unsafely subject to misconfiguration. This is partially due to missing sanity checks and escape clauses, although the lack of granularity in updates makes integrating appropriate checks unintuitive. Consider separating the update mechanisms targeting the specific configuration parameters in *_hostParams* to allow the seamless application of necessary sanity operations, without impacting the ease of configuration for less critical data. In the described evolution above *_hostParams.hostManager* and *_hostParams.admin* would be updated separately to, for example, *_hostParams.fishermen* or *_hostParams.timeout*.

Design with security monitoring in mind. To enhance security and off chain monitoring capability of the bridges, consider monitoring the Solidity side of the system with relevant data requirements for the off-chain tools. To support this, all ISMP handler's data should be indexed and emitted via events.

7.3 Further recommended best practices

Engage in an economic audit. Although SRLabs has some knowledge of economic attacks, our primary goal during engagements is to find logic vulnerabilities through code assurance. Therefore, an economic audit for the ISMP actors such as Relayer and Fishermen, and the pallet configuration is recommended to ensure the safety of Hyperbridge platform and users.

Improve documentation and coding practice. Increasing the in-line comments to reflect the protocol design could enhance the efficiency of assessing the ISMP features and design choices for security weaknesses. Additionally, naming the variables [42] with meaningful identifiers to reflect their purpose will facilitate understanding the control flow of the code during internal or external code review processes. Introduce expressive names for the data structures as *PostRequest* and *GetRequest* instead of *Post* and *Get* [43]. Remove unimplemented errors to maintain a clean code base [44]. Enhancing these aspects could significantly streamline the review process, facilitate a better understanding of the code's purpose and design, and contribute to a more efficient and effective security evaluation

Improve repository organization. Implementing test logic and business logic in a single file is considered a best practice deviation [45] [46]. Such an approach clutters the code, making it difficult to navigate and maintain. We recommend separating the test logic from the pallet implementation in a test file or module. This will improve the long-term maintainability and may prevent the introduction of bugs to the pallets as the pallet implementation continues to evolve.

Regular code review and continuous fuzz testing. Regular code reviews are recommended to avoid introducing new logic or arithmetic bugs, while continuous fuzz testing can identify potential vulnerabilities early in the development process. Ideally, Hyperbridge should continuously fuzz their code on each commit made to the codebase. The substrate-runtime-fuzzer [47] provides a good example of multiple fuzzing harnesses.

Regular updates. New releases of polkadot-sdk may contain fixes for critical security issues. Since Hyperbridge is a product that heavily relies on polkadot-sdk, updating to the latest version as soon as possible whenever a new release is available is recommended.

Further investigation of Solidity modules. The investigation conducted on *EvmHost.sol* and *HandlerV1.sol* was heavily constrained in scope. Based on our investigation we believe that the Solidity implementation requires further analysis from a team specialized in smart contract auditing. We recommend investing further resources into the security of the Solidity codebase, including the associated custom libraries.

Bibliography

[1] [Online]. Available: https://github.com/polytope-labs/hyperbridge/tree/main/modules/ismp.

[2] [Online]. Available: https://github.com/polytope-labs/hyperbridge/tree/main/parachain/runtimes/nexus.

[3] [Online]. Available: https://github.com/polytope-labs/ismp-solidity.

[4] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit.

[5] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/1.

[6] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/2.

[7] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/3.

[8] [Online]. Available: https://github.com/polytope-labs/hyperbridge/pull/234.

[9] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/4.

[10] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/5.

[11] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/7.

[12] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/6.

[13] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/11.

[14] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/13.

[15] [Online]. Available: https://github.com/polytope-labs/hyperbridge/pull/235.

[16] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/10.

[17] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/9.

[18] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/8.

[19] [Online]. Available: https://github.com/polytope-labs/hyperbridge/pull/234/files#diff-6c9e74a4b6df83a3c561677fcafdfde2aebe38de281ccb57f2cfae85e7fa4c0aR113.

[20] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/12.

[21] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/22.

[22] [Online]. Available: https://github.com/polytope-labs/hyperbridge/pull/237.

[23] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/16.

[24] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/14.

[25] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/15.

[26] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/20.

[27] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/18.

[28] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/17.

[29] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/19.

[30] [Online]. Available: https://github.com/polkadot-fellows/runtimes/pull/87.

[31] [Online]. Available: https://github.com/polkadot-fellows/runtimes/blob/5bf21d73c0456eb7c5910aafa78445f93a61bdc9/system-parachains/asset-hubs/asset-hub-kusama/src/lib.rs#L684-L684.

[32] [Online]. Available: https://github.com/paritytech/polkadot-sdk/blob/b6231c79ca708d9c9280210f41ca38fd816c8ad9/cumulus/pallets/xcmp-queue/src/lib.rs#L955.

[33] https://github.com/polytope-labs/hyperbridge/pull/234/commits/d864430edb64f5ad17d32a99778d0a4242071f8a.

[34] [Online]. Available: https://github.com/polkadot-fellows/runtimes/blob/2663a3a5f3b6a757efe102c3c74c5422499dda39/system-parachains/asset-hubs/asset-hub-polkadot/src/lib.rs#L243.

[35] [Online]. Available: https://github.com/polytope-labs/hyperbridge/blob/7738ba670214c7bbfb41e6417d6cfb3e88684b09/docs/pages/protocol/timeouts.mdx#L57-L58.

[36] [Online]. Available: https://github.com/polytope-labs/hyperbridge/blob/ddf7bf378735a8be94b44fa9dd24ad20a0d887bd/modules/ismp/pallets/pallet/src/lib.rs#L436-L438.

[37] [Online]. Available: https://github.com/Wizdave97/hyperbridge-audit/issues/10#issuecomment-2196634090.

[38] [Online]. Available: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/utils/SafeERC20.sol.

[39] [Online]. Available: https://github.com/polytope-labs/hyperbridge/blob/b3fc8e47dffe47555c2421bc897ee02f1f13f0ce/docs/pages/developers/explore/modules/fishermen.mdx#L18.

[40] [Online]. Available: https://github.com/polkadot-fellows/runtimes/blob/360581ffedd048262177ddc135d66cec455b959a/system-parachains/asset-hubs/asset-hub-kusama/src/lib.rs#L1005C1-L1035C2.

[41] [Online]. Available: https://github.com/polkadot-fellows/runtimes/blob/2663a3a5f3b6a757efe102c

3c74c5422499dda39/system-parachains/asset-hubs/asset-hub-polkadot/src/lib.rs#L243.

[42] [Online]. Available: https://github.com/polytope-labs/hyperbridge/blob/d4fffd7e6672ca3bd1a927c9fc677e14c5066186/modules/ismp/core/src/handlers/request.rs#L89.

[43] [Online]. Available: https://github.com/polytope-labs/hyperbridge/blob/d4fffd7e6672ca3bd1a927c9fc677e14c5066186/modules/ismp/core/src/router.rs#L136-L143.

[44] [Online]. Available: https://github.com/polytope-labs/hyperbridge/blob/d4fffd7e6672ca3bd1a927c9fc677e14c5066186/modules/ismp/pallets/hyperbridge/src/lib.rs#L151-L152.

[45] [Online]. Available: https://github.com/polytope-labs/hyperbridge/blob/c9e23d530da5a9ee8517d71ac8242ac2bb54b57d/modules/ismp/core/src/host.rs#L387.

[46] [Online]. Available: https://github.com/polytope-labs/hyperbridge/blob/c9e23d530da5a9ee8517d71ac8242ac2bb54b57d/modules/ismp/pallets/relayer/src/withdrawal.rs#L77.

[47] [Online]. Available: https://github.com/srlabs/substrate-runtime-fuzzer/tree/main.