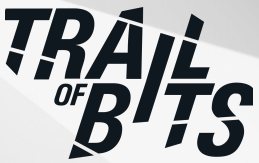

Hyperlane V3

Security Assessment (Summary Report)

November 6, 2023

Prepared for:
Abacus Works, core developer for Hyperlane
Hyperlane

Prepared by: **Michael Colburn, Damilola Edwards, and Samuel Moelius**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Brooke Langhorne, Project Manager
brooke.langhorne@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Michael Colburn, Consultant
michael.colburn@trailofbits.com

Damilola Edwards, Consultant
damilola.edwards@trailofbits.com

Samuel Moelius, Consultant
samuel.moelius@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
September 20, 2023	Pre-project kickoff call
October 3, 2023	Delivery of report draft
October 3, 2023	Report readout meeting
October 27, 2023	Delivery of fix review draft
October 30, 2023	Delivery of supplementary review draft
October 30, 2023	Supplementary review readout meeting
November 7, 2023	Delivery of summary report

Executive Summary

Engagement Overview

Abacus Works engaged Trail of Bits to review the security of the changes to the Hyperlane cross-chain messaging protocol smart contracts, as part of the V3 upgrade in commit `fcfecdf` of PR #2733.

A team of two consultants conducted the review from September 25 to September 29, 2023, for a total of two engineer-weeks of effort. With full access to source code and documentation, we performed static and dynamic testing of the codebase, using automated and manual processes.

Observations and Impact

This review was focused on the changes introduced in the new version of the Hyperlane smart contracts. The core `Mailbox` contract saw heavy modification, with some functionality having been generalized into a new modular hook pattern. Other changes were smaller in scope and spread across a number of contracts. Our review focused on identifying any unhandled edge cases that may have been introduced by these architectural changes, reviewing the protocol fee logic to check that funds are being handled properly and cannot be accessed without permission, and exploring how malicious custom hooks could affect the protocol.

This review was scoped to review only the changes to the Solidity contracts, and as a result, we did not perform an in-depth review of unmodified contracts or functionality but did refer to them when necessary to improve our understanding of the overall system. Any Rust components, such as the relayer and validator codebases, were not in scope for this review.

Our review did not identify any security issues. In appendix B, we provide a list of code quality recommendations that could help improve the readability or maintainability of the code but that are not directly related to any security concerns.

The inclusion of end-to-end tests in the testing suite helps improve confidence in the functionality of the code and is crucial for a cross-chain protocol like this. The new features in this version do simplify users' interactions with the system but also increase the overall complexity of the codebase, so ensuring documentation is up to date both inside the contracts and in higher-level documentation will be important going forward.

Recommendations

The codebase could benefit from a comprehensive review of all on-chain and off-chain components to help ensure correctness. Due to the short timeframe and specific focus of this engagement, some contract functionality was not reviewed as thoroughly, and none of

the off-chain components were reviewed. Abacus Works should also continue to update the documentation and investigate how the current configuration for fuzz tests affects coverage and whether the use of the default number of runs is adequate.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The codebase uses version 0.8 of the Solidity compiler, which includes built-in overflow detection. The arithmetic in the system is generally simple, as it primarily relates to gas and fee calculations and loop counters.	**Satisfactory**
Auditing	The contracts have adequate events to facilitate monitoring, which is critical for a cross-chain messaging protocol. The user documentation includes instructions for setting up monitoring for a validator, which consumes the messages being sent through the protocol.	**Satisfactory**
Authentication / Access Controls	The access control schema used by the system is straightforward. Functions are restricted to specific contracts, or the contract owner where necessary. The hook and interchain security module mechanisms (ISMs) also perform an important role with respect to authorization in the system. We did not identify any issues in these mechanisms, but their modular natures will make proper configuration critical.	**Satisfactory**
Complexity Management	While the new version of the contracts provides a simpler interface to users, under the hood there are long call sequences and complex contract inheritances, which can make tracing the flow of execution difficult at times. The modularity of the hook and ISM architecture allows a lot of flexibility in configuring the system, but this also inherently increases the complexity. We did not identify any significant code duplication, and despite the long call sequences, individual functions are easy to understand.	**Moderate**

Decentralization	There does not appear to be support for slashing at this time, so validators have a high degree of trust in the system. However, anybody can set up an independent deployment of the system with a different validator set. This could provide a viable alternative to untrusting users, especially for cases outside of token bridging (which may rely more heavily on network effects).	**Moderate**
Documentation	The system has good documentation of the current version for users and external integrators. Ensure that this documentation is promptly updated to match the new version. Much of the codebase has good NatSpec comment coverage; however, the newer code is commented less consistently. In particular, there is a documentation gap around the functionality and intended use of the various hooks.	**Moderate**
Low-Level Manipulation	Assembly use is minimal and most instances are trivial. The assembly block in the `MetaProxy` library could be better documented. The contracts leverage libraries for handling low-level calls, and return values are checked where appropriate.	**Satisfactory**
Testing and Verification	Tests for the codebase are run in CI, including end-to-end tests. Any gaps in coverage are highlighted in new pull requests. The test suite also includes some fuzz tests, but they are run only for the Foundry default of 256 runs, so it is unclear if the fuzz tests meaningfully improve test coverage.	**Satisfactory**
Transaction Ordering	We did not identify any concerns related to transaction ordering in the codebase.	**Satisfactory**

A. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Front-Running Resistance	The system's resistance to front-running attacks
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

B. Code Quality Recommendations

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

- The `Indexed` contract is inherited by several contracts, but its functionality is not currently being used. Consider removing it if it is not required.
- Consider renaming the `DestinationRecipientRoutingHook` contract to be more in line with the `DomainRoutingHook` contract it inherits from. Similarly, the `configCustomHook` function could be renamed to `setHook` to align with a similar function in the `DomainRoutingHook` contract.
- The `ERC5164Hook` contract's `_quoteDispatch` function reverts while the `_quoteDispatch` function in other hook contracts returns `0`. Update this function to be consistent with the others, or document why it is expected to behave differently.

C. Fix Review Results

On October 26, 2023, Trail of Bits reviewed fixes for four issues identified by Abacus Works concurrent to our review, as well as minor additional changes to the smart contracts. We reviewed each fix to determine its effectiveness in resolving the associated issue. No additional issues were identified during this fix review.

The first issue was the inability to unenroll a remote router from the system. This issue was addressed in PR #2760 by adding the `unenrollRemoteRouter` function in the `Router` contract and a `remove` function in both the `DefaultFallbackRoutingIsm` and `DomainRoutingIsm` contracts.

The second issue identified an edge case where calls to `Mailbox.process` unexpectedly revert if they specify a recipient contract that has a fallback function but does not have an `interchainSecurityModule` function to specify a non-default ISM. This issue was addressed in PR #2767, which updated the logic to check for a nonzero return data length before attempting to decode the return value as an address.

The third issue identified an edge case where users receive an unclear error message if they include insufficient payment with their call to `Mailbox.dispatch`, as this triggers an arithmetic underflow. This issue was addressed in PR #2769 by capping the quoted payment amount to the amount provided by the caller and deferring the error handling to the underlying hook that is being underpaid.

The fourth issue was a potential race condition with the `AbstractMessageIdAuthorizedIsm` contract's `verify` function. This issue could allow an attacker to directly call `verify` after the authorized entity calls the `verifyMessageId` function to mark a message as verified, but before the relayer triggers the call to `verify` as part of the expected flow. This would potentially allow the attacker to transfer funds more times than intended. This issue was addressed in PR #2835 by adding logic to reduce the outstanding value to be transferred during each call to `verify`. However, this fix is not yet included in the tagged pre-release as of the commit listed below.

The remaining code changes mainly consist of reorganizing the directory structure of the contracts, changing visibility of some functions, and making minor simplifications to some contracts. Full details of the additional changes can be found on the `audit-remediations` tagged release at commit `d69d76a`.

D. Supplementary Review Summary

Engagement Overview

Abacus Works engaged Trail of Bits to review the security of the changes to the Hyperlane cross-chain messaging protocol smart contracts, as part of the V3 upgrade in commit `fcfecdf` of PR #2733.

To ensure the changes received sufficient coverage, one consultant performed a subsequent review from October 25 to October 27, 2023, for a total of three additional engineer-days of effort. With full access to source code, documentation, and the results of the September 25–29 review, we performed static and dynamic testing of the codebase, using automated and manual processes.

Observations and Impact

The system is extremely complex by virtue of being an interchain messaging system. Thus, it would benefit from a comprehensive review in its entirety.

The need for a comprehensive review is further evidenced by the following events. While reviewing the changes introduced by `fcfecdf`, we found a double spend bug not related to those changes.[1] Fortunately, the bug was also noticed by Abacus Works, and fixed by PR #2835. However, based on our understanding of the code, the bug could have been catastrophic for the protocol.

The V3 audit remediations document suggests that producing helpful revert messages is preferable to relying on Solidity panics, and we agree. We encourage Abacus Works to continue this practice, even though the view is not held industry wide.

Recommendations

Based on the findings identified during the security review, Trail of Bits recommends that Abacus Works take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.
- **Seek a comprehensive review of the code in its entirety.** Interchain messaging systems are complex by their nature. Moreover, we happened upon a potentially

[1] To elaborate, we ran Slither as we normally do for Solidity audits. Slither produced a false positive regarding potentially locked ether in the `ERC5164Ism` contract via the `AbstractMessageIdAuthorizedIsm.verifyMessageId` function. While investigating the Slither report, we noticed the potential double spend bug.

catastrophic bug by accident. These facts suggest a comprehensive review is needed.

- **Continue to prefer informative revert messages over Solidity panics.** Doing so allows one to distinguish anticipated failures (the former case) from unanticipated failures (the latter case).

E. Supplementary Review Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- **Documentation review.** We reviewed the Hyperlane documentation to get an overview of what the system does and what its main components are. We also reviewed the V3 audit remediations document, to understand the issues discovered following the September 25–29 Trail of Bits audit.
- **Static analysis.** We ran Slither over the codebase and reviewed the results.
- **Test coverage analysis.** We verified that the project's Hardhat and Foundry tests pass. We also computed their test coverage and looked for obvious gaps.
- **Pull request review.** We reviewed the pull requests mentioned in the V3 audit remediations document in attempt to understand how they address the issues named in that document.
- **Manual review.** We cursorily reviewed the changes introduced by commit `fcfecdf`.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- **Interchain messaging/bridging systems are inherently complex.** The semantics of such systems depend upon the semantics of the chains with which they operate. Since such systems necessarily involve multiple chains, they are more complex than typical, single-chain blockchain projects.
- **Reviewing a diff is inherently limited.** Determining whether change introduces a bug requires an understanding of how the code operates prior to the change. Without such an understanding, one must make assumptions about how the code operates.

F. Supplementary Review Findings

1. ERC165 (standard interface detection) not used

Severity: **Low**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-HYPERLANE-1
Target: `solidity/contracts/client/MailboxClient.sol`, `solidity/contracts/hooks/DomainRoutingHook.sol`	

Description

In several places, an address is cast to an interface without first using ERC165: Standard Interface Detection to check that the interface is implemented. Assuming an interface is implemented, when it is not, could lead to undesired behavior or confusing error messages.

Examples where interface detection could be used appear in figures 1.1 through 1.3. Note also that the `setHook` function in figure 1.3 appears to lack an `onlyContractOrNull` modifier. However, the problem would become moot if interface detection were used.

```
63    function setHook(address _hook) public onlyContractOrNull(_hook) onlyOwner {
64        hook = IPostDispatchHook(_hook);
65    }
```

Figure 1.1: Example where ERC165: Standard Interface Detection could be used (1 of 3) (hyperlane-monorepo/solidity/contracts/client/MailboxClient.sol#63–65)

```
71    function setInterchainSecurityModule(address _module)
72        public
73        onlyContractOrNull(_module)
74        onlyOwner
75    {
76        interchainSecurityModule = IInterchainSecurityModule(_module);
77    }
```

Figure 1.2: Example where ERC165: Standard Interface Detection could be used (2 of 3) (hyperlane-monorepo/solidity/contracts/client/MailboxClient.sol#71–77)

```
45    function setHook(uint32 destination, address hook) public onlyOwner {
46        hooks[destination] = IPostDispatchHook(hook);
47    }
```

Figure 1.3: Example where ERC165: Standard Interface Detection could be used (3 of 3) (`hyperlane-monorepo/solidity/contracts/hooks/DomainRoutingHook.sol#45–47`)

Note: Abacus Works noticed a case where the `Mailbox.recipientIsm` function would revert if the recipient did not correctly implement the `ISpecifiesInterchainSecurityModule` interface. The problem was fixed by PR #2767. Similar cases could be averted by using ERC165.

Exploit Scenario

Alice accidentally calls `MailboxClient.setInterchainSecurityModule` with an address that is not for ISM. Her mailbox client does not function correctly and produces confusing error messages. Time and effort are wasted trying to diagnose the problem.

Recommendations

Short term, adjust the code in figures 1.1 through 1.3 to use ERC165: Standard Interface Detection. Doing so will help to catch cases where the functions are called with contracts that do not implement the required interfaces.

Long term, regularly test your code with invalid inputs, including contracts that do not implement the required interfaces. Doing so will help to ensure that such cases are handled correctly, and that useful error messages are produced.

2. Use of outdated dependencies and dependency management tools

Severity: **Undetermined**	Difficulty: **Undetermined**
Type: Patching	Finding ID: TOB-HYPERLANE-2
Target: `.yarnrc.yml`, `yarn.lock`	

Description

The project uses an old version of Yarn, and an old version of the OpenZeppelin contracts. Use of outdated software could mean bug fixes are missed.

The project uses Yarn version 3.2.0, as shown in figure 2.1. The most recent version of Yarn in the 3.2.* series is 3.2.4, and in the 3.*.* series is 3.4.1.

```
11    yarnPath: .yarn/releases/yarn-3.2.0.cjs
```

Figure 2.1: Yarn version the project is pinned to (version 3.2.0) (hyperlane-monorepo/.yarnrc.yml#11)

Similarly, the project uses OpenZeppelin contracts version 4.8.0, as shown in figure 2.2. The most recent version of the contracts in the 4.8.* series is 4.8.3, and in the 4.*.* series is 4.9.3.

```
4892    "@openzeppelin/contracts@npm:^4.8.0":
4893      version: 4.8.0
4894      resolution: "@openzeppelin/contracts@npm:4.8.0"
4895      checksum:
dfab51a7f91735cfb1e94dd5074736b0dac0207e4ebf26eb46b32defd3b67adce5a36b248daa7b841c21
be74863c1e37cf92ed194a9c36d3f8c5326d1a24242a
4896      languageName: node
4897      linkType: hard
```

Figure 2.2: OpenZeppelin contracts version the project is pinned to (version 4.8.0) (hyperlane-monorepo/yarn.lock#4892–4897)

Note that the latter problem seems to be related to the former. If one deletes the `.yarnrc.yml` file and runs the latest stable version of Yarn (version 1.22.19), the `yarn.lock` file is updated to refer to version 4.9.3 of the OpenZeppelin contracts.

Exploit Scenario

A bug is found in version 4.8.0 of the OpenZeppelin contracts. The bug affects Hyperlane. Mallory exploits Hyperlane, knowing it uses the outdated, vulnerable contracts.

Recommendations

Short term, unless there is a good reason not to, use the latest stable version of Yarn (version 1.22.19). For each of Hyperlane's immediate dependencies, verify that the most recent version of the dependency appears in the `yarn.lock` file. Taking these steps will help ensure that Hyperlane does not use outdated, vulnerable dependencies.

Long term, regularly run `yarn upgrade`. Doing so will help ensure that Hyperlane uses up-to-date dependencies.