HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Hyperlane
Date: Apr 03, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Hyperlane
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	Interchain messaging
Platform	EVM
Language	Solidity
Methodology	Link
Website	hyperlane
Changelog	28.01.2023 - Initial Review 03.04.2023 - Second Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Hyperlane (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is review and security analysis of smart contracts in the repository:

Initial review scope

Repository	https://github.com/hyperlane-xyz/hyperlane-monorepo/tree/audit-v2/solidity/contracts https://github.com/hyperlane-xyz/hyperlane-token/tree/main/contracts
Commit	b450181c7fc255149ce86551f21dacd7ab47d5f5 ca7bb745a6fb9f21ec727a1a41a55c800d2f453a
Whitepaper	Link
Functional Requirements	Link
Contracts	File: ./solidity/contracts/Call.sol SHA3: 93dbcebdbbf479563e3c43ba33e392a17df3513073fd6df69edb1461f5d7780d File: ./solidity/contracts/HyperlaneConnectionClient.sol SHA3: 146bc2a168ff2752e01a317d7c09ea581ca91567edc48eb4b80d9d0f099ec771 File: ./solidity/contracts/InterchainGasPaymaster.sol SHA3: 694e10520d18dbb217d52b62d0461b4c16c8a528490ff8ff610fe0f8d6aad91c File: ./solidity/contracts/isms/MultisigIsm.sol SHA3:cf9b43321dfc90f3914fc81ebb1ba78916ed828573146935a7f5e79e26bc30a7 File: ./solidity/contracts/libs/EnumerableMapExtended.sol SHA3: c4115c2ce6f2a0efc48c63e5869464eda34f8b5cfee9fc42eba149d74d23ef7c File: ./solidity/contracts/libs/Merkle.sol SHA3: 26bda8009ee094c60b3c0c10ec454bc3bfffbc74d45c49fd1ed50d696aed7751 File: ./solidity/contracts/libs/Message.sol SHA3: f9ff058e169c9d5ff5a869293fdba1a8531a1e0db5c61a7b8885a26deda95b64 File: ./solidity/contracts/libs/MinimalProxy.sol SHA3: 510914afe4a147b567a630146199b8fe3c17d1f5834698c47f1ad097f39797f9 File: ./solidity/contracts/libs/MultisigIsmMetadata.sol SHA3: 3fcd0f0657d0e9568ad90367a63cd6f39e90849c4396e2fe6e7d9ddd570f0689 File: ./solidity/contracts/libs/TypeCasts.sol SHA3: a6aba3f1569a897fef473a4a05d632db8b111eb41fe48f8c36c27a7d382a028a

	File: ./solidity/contracts/Mailbox.sol SHA3: fb168497e3a11a0915c11b7a870db5d9b58ed64575d5c37adae097129d23c8ca File: ./solidity/contracts/middleware/InterchainAccountRouter.sol SHA3: 2ace30bb884683fa36e49efa2509b7921ad14bb1c34df25708faf65a644768de File: ./solidity/contracts/middleware/InterchainQueryRouter.sol SHA3: dc5f3899c2dee5eec6fb58a6167036f2277b58de81e3e3f4cae0eefd6481a298 File: ./solidity/contracts/middleware/liquidity-layer/adapters/CircleBridgeAdapter.sol SHA3: 641fbcbfc6fa305e58fac7862e705b23ea92c43e66c2a662abe4257dfe207cd8 File: ./solidity/contracts/middleware/liquidity-layer/interfaces/circle/ICircleBridge.sol SHA3: 21d6baf6b1c2573cdc496920e4a6e69cbebc53dbbd26b7ebbb60fb883efb50f0 File: ./solidity/contracts/middleware/liquidity-layer/interfaces/circle/ICircleMessageTransmitter.sol SHA3: 16239fdf7eb2cdba4972b61bd3200626d8116d52152fdb618788df78a6698f13 File: ./solidity/contracts/middleware/liquidity-layer/interfaces/ILiquidityLayerAdapter.sol SHA3: 8fb04b2bdb660334756796dc3168ea0518ac537257b3bf59db0b8b9321099d8d File: ./solidity/contracts/middleware/liquidity-layer/LiquidityLayerRouter.sol SHA3: 45c016018f1bb69881744038a496febebbe590943197d8f4aba3e45d620f32a5 File: ./solidity/contracts/OwnableMulticall.sol SHA3: 7366a1ce357c857421b952c35c9d5a253af8a178c883a42f891b9b0b6154c1cb File: ./solidity/contracts/PausableReentrancyGuard.sol SHA3: f21a36f320243422aaf896d1127d7a2ad44c9e04610ba79bc23ac68ddaacf849 File: ./solidity/contracts/Router.sol SHA3: 6b8d3ffaeff8d2935b893a74e4705341c5b317771b8496246094781078adafc0 File: ./solidity/contracts/upgrade/Versioned.sol SHA3: 7b1e8357094ec9676a04cf8b6b6e022155c12f9949688dd9a6479e5640716a60 File: ./solidity/interfaces/IInterchainAccountRouter.sol SHA3: bd6fa3bc80ec1faec01f41f744cae3323ee14d469ecd537995f457da4aa07ddf File: ./solidity/interfaces/IInterchainGasPaymaster.sol SHA3: 2684fb8703fc8b29c3e7ad6c8e617322c8912eaadb400aad88255781a2621024 File: ./solidity/interfaces/IInterchainQueryRouter.sol SHA3: 42283382b3e8a82969972b25334d486d91edbed4e5bfa44c0adfbadf8d634984 File: ./solidity/interfaces/IInterchainSecurityModule.sol SHA3: abd2e0879ec812a8ae5f1975977128ca1dd19f7a2af239682096b05e82ea7f4f File: ./solidity/interfaces/ILiquidityLayerMessageRecipient.sol SHA3: 2c4b4723e71a016152f7ddd66ab55cea8ab6beca88d9f39c885f4c2e97bb177c

	File: ./solidity/interfaces/ILiquidityLayerRouter.sol SHA3: 5dc89eab3cf04fe17e3d56c24d860935e8e62294d329e296e423ea46e46d25bd File: ./solidity/interfaces/IMailbox.sol SHA3: c75094d68ec0a1ad5d92d62629c77e37c555995a35b16311309099c3f59e1329 File: ./solidity/interfaces/IMessageRecipient.sol SHA3: df770d6ff439c2300c4ba039ecdc5eb5757db06d8df98d64a47659204c4bfd1b File: ./solidity/interfaces/IMultisigIsm.sol SHA3: 2e3e81056b2987fd8a206897d21a91868d2e3ec48960ce457f0b14e56d54fda2 File: ./contracts/extensions/HypERC721URICollateral.sol SHA3: 6949190f1a83296434106aae00399bf5dad21048a5469bdac99ca09165706dd0 File: ./contracts/extensions/HypERC721URIStorage.sol SHA3: 53c59911b31de8aad9331f029d0a321f1332fcdc9b462d1cd5e9c113484b5a82 File: ./contracts/HypERC20.sol SHA3: cd2d9a01a07e65c74def51566ce8ff0810168af9c31fc593312ffb65fd4b247e File: ./contracts/HypERC20Collateral.sol SHA3: fc5b7355f253543dbbc37eec0400bb451f058d676ff3f8905085b3eb5f2b3a75 File: ./contracts/HypERC721.sol SHA3: 11a4e5d5b394b16174fff30fb13e6b98bcce0a9f699ed6e31a1b5b5da3640913 File: ./contracts/HypERC721Collateral.sol SHA3: bbe96712d420bab07480b970682e5f7e7d1b02c9cd8b442f09a759db2b2dd5f9 File: ./contracts/libs/Message.sol SHA3: b9647698d1d28efae16d0194226fbcbb77b3a9dd1bfa1e0886ae6d0506aaed8b File: ./contracts/libs/TokenRouter.sol SHA3: 6bf57431c6a35160fe02deec424664a92094cbb14a3c903cb7b29fa8087958eb File: ./solidity/contracts/upgrade/ProxyAdmin.sol SHA3: e986bb64e43d1929a57096e06456ab29c007ca3df4d95cebc02a809bdaef9eb5 File: ./solidity/contracts/upgrade/TransparentUpgradeableProxy.sol SHA3: d7f2f0942f57e0c6dafe7ba9be364e50f2381082f6de7d70ce2b04c4c627fb8e File: ./solidity/contracts/Create2Factory.sol SHA3: 6f46791901284cb413b66462a391a2ef5d13ca29cfa19173878fdaf1ec407a36

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Second review scope

Repository	https://github.com/hyperlane-xyz/hyperlane-monorepo/tree/audit-v2-remediation
Commit	def40316e9e0fee6857ece40d60a6ddcf2247e90
Whitepaper	Link
Functional Requirements	Link
Contracts	File: ./solidity/contracts/Call.sol SHA3: c526527679d5e752d2e900e13792ec455a356c3635288249ce908d6f2759f4d6 File: ./solidity/contracts/Create2Factory.sol SHA3: a377db630f17238ec9f32c30c30dff5e181e9e27738fab6ba148bb648e12e3a2 File: ./solidity/contracts/HyperlaneConnectionClient.sol SHA3: 146bc2a168ff2752e01a317d7c09ea581ca91567edc48eb4b80d9d0f099ec771 File: ./solidity/contracts/InterchainGasPaymaster.sol SHA3: 8634708493afd063c1fd81c530e4b5e4db454b5dc7bf2c6c4cd78057b234c610 File: ./solidity/contracts/isms/MultisigIsm.sol SHA3: b16c6b2f95824e33f712444082c4295034803759b9648219a1911cc14ec2eb8e File: ./solidity/contracts/libs/EnumerableMapExtended.sol SHA3: c4115c2ce6f2a0efc48c63e5869464eda34f8b5cfee9fc42eba149d74d23ef7c File: ./solidity/contracts/libs/Merkle.sol SHA3: 26bda8009ee094c60b3c0c10ec454bc3bfffbc74d45c49fd1ed50d696aed7751 File: ./solidity/contracts/libs/Message.sol SHA3: f9ff058e169c9d5ff5a869293fdba1a8531a1e0db5c61a7b8885a26deda95b64 File: ./solidity/contracts/libs/MinimalProxy.sol SHA3: 462abf74fb52f3dae320840027749198c9c067323e562fa1be68d965a299bfaa File: ./solidity/contracts/libs/MultisigIsmMetadata.sol SHA3: 3fcd0f0657d0e9568ad90367a63cd6f39e90849c4396e2fe6e7d9ddd570f0689 File: ./solidity/contracts/libs/TypeCasts.sol SHA3: a6aba3f1569a897fef473a4a05d632db8b111eb41fe48f8c36c27a7d382a028a File: ./solidity/contracts/Mailbox.sol SHA3: cd6b6f1a43658fb060a6416c661ae3c7460882e1be0291edc47b158eca6e158c File: ./solidity/contracts/middleware/InterchainAccountRouter.sol SHA3: 833a3264c484ece80269b385caa1b3842f028e5ffc71094fdf06e90707a5dc7e File: ./solidity/contracts/middleware/InterchainQueryRouter.sol SHA3: 73fc1bb053ca8f86488234108b930b684b4816d99fe0215d670fee27a919249b File: ./solidity/contracts/middleware/liquidity-layer/adapters/CircleBridgeAdapter.sol SHA3: 6e5cdac68c278c07b49a74fbc76066ba93d9ebf7b33bf48c0f8b7434b6cbd7f7

	File: ./solidity/contracts/middleware/liquidity-layer/interfaces/circle/ICircleBridge.sol SHA3: 21d6baf6b1c2573cdc496920e4a6e69cbebc53dbbd26b7ebbb60fb883efb50f0 File: ./solidity/contracts/middleware/liquidity-layer/interfaces/circle/ICircleMessageTransmitter.sol SHA3: 16239fdf7eb2cdba4972b61bd3200626d8116d52152fdb618788df78a6698f13 File: ./solidity/contracts/middleware/liquidity-layer/interfaces/ILiquidityLayerAdapter.sol SHA3: 8fb04b2bdb660334756796dc3168ea0518ac537257b3bf59db0b8b9321099d8d File: ./solidity/contracts/middleware/liquidity-layer/LiquidityLayerRouter.sol SHA3: 95b8bf70fb0635988258cdd5d27c1c91ba8d6bdce784b8f1bb456e2cbd377fa5 File: ./solidity/contracts/OwnableMulticall.sol SHA3: 586923b834484a399fff21e7078149df06cf2b303b49ebc02a13307e2e4303d6 File: ./solidity/contracts/PausableReentrancyGuard.sol SHA3: f21a36f320243422aaf896d1127d7a2ad44c9e04610ba79bc23ac68ddaacf849 File: ./solidity/contracts/Router.sol SHA3: 91742a671942082210cb0f805abcb1e6ac7bc5b2c99166861494ce8c6c7bca12 File: ./solidity/contracts/upgrade/ProxyAdmin.sol SHA3: e986bb64e43d1929a57096e06456ab29c007ca3df4d95cebc02a809bdaef9eb5 File: ./solidity/contracts/upgrade/TransparentUpgradeableProxy.sol SHA3: d7f2f0942f57e0c6dafe7ba9be364e50f2381082f6de7d70ce2b04c4c627fb8e File: ./solidity/contracts/upgrade/Versioned.sol SHA3: 7b1e8357094ec9676a04cf8b6b6e022155c12f9949688dd9a6479e5640716a60 File: ./solidity/interfaces/IInterchainAccountRouter.sol SHA3: bd6fa3bc80ec1faec01f41f744cae3323ee14d469ecd537995f457da4aa07ddf File: ./solidity/interfaces/IInterchainGasPaymaster.sol SHA3: 2684fb8703fc8b29c3e7ad6c8e617322c8912eaadb400aad88255781a2621024 File: ./solidity/interfaces/IInterchainQueryRouter.sol SHA3: 42283382b3e8a82969972b25334d486d91edbed4e5bfa44c0adfbadf8d634984 File: ./solidity/interfaces/IInterchainSecurityModule.sol SHA3: abd2e0879ec812a8ae5f1975977128ca1dd19f7a2af239682096b05e82ea7f4f File: ./solidity/interfaces/ILiquidityLayerMessageRecipient.sol SHA3: 2c4b4723e71a016152f7ddd66ab55cea8ab6beca88d9f39c885f4c2e97bb177c File: ./solidity/interfaces/ILiquidityLayerRouter.sol SHA3: 5dc89eab3cf04fe17e3d56c24d860935e8e62294d329e296e423ea46e46d25bd File: ./solidity/interfaces/IMailbox.sol SHA3: c75094d68ec0a1ad5d92d62629c77e37c555995a35b16311309099c3f59e1329 File: ./solidity/interfaces/IMessageRecipient.sol

	SHA3: df770d6ff439c2300c4ba039ecdc5eb5757db06d8df98d64a47659204c4bfd1b File: ./solidity/interfaces/IMultisigIsm.sol SHA3: b24df09465064fbe3daf54b77fcaa2e86dd1f6b7152e35c95a240d28c7703070

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor gas optimization. These issues won't have a significant impact on code execution but affect code quality

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- The documentation is overall clear and complete

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.

Test coverage

Code coverage of the project is **100%** (branch coverage).

- Deployment and basic user interactions are covered with tests.

Security score

As a result of the audit, the code contains **2** medium and **2** low severity issues. The security score is **8** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **8.6**.

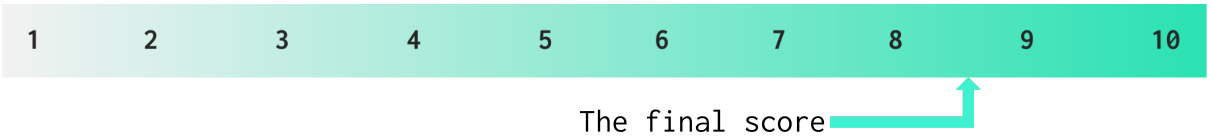


Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
28 January 2023	10	6	5	0
03 April 2023	2	2	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Failed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed

Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Passed
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Not Relevant
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed

Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply Manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Passed
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

System Overview

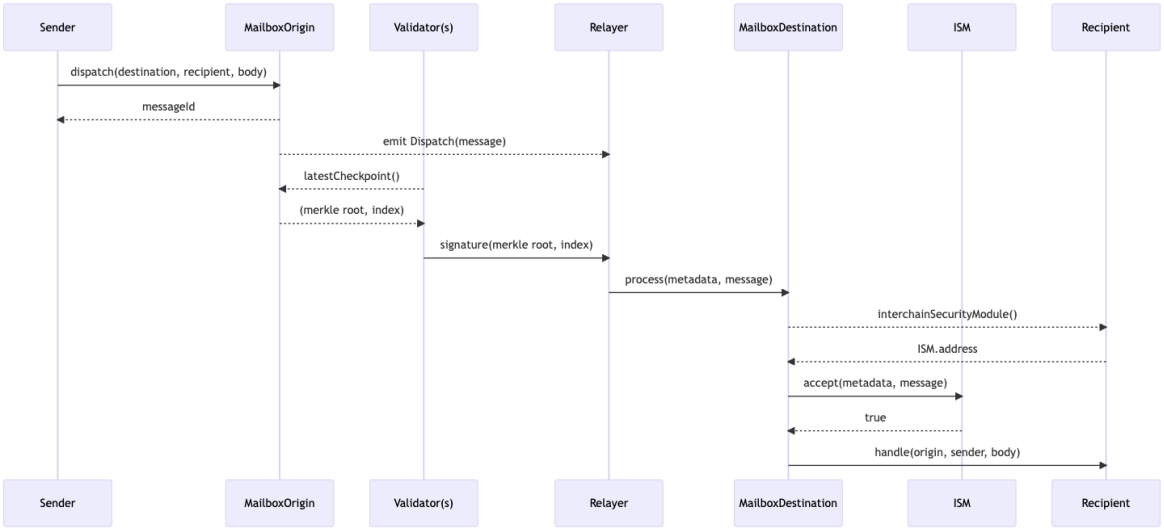
There are two different repositories in the project, the repository for the main protocol and the repository for the token standard.

1. Token repository - Warp API
   Developers can use Hyperlane's Warp API to permissionlessly deploy "Warp Routes", contracts that allow ERC20 and ERC721 tokens to move effortlessly between chains.
   Unlike other token wrapping protocols, Warp Routes are secured by a sovereign consensus.
2. Hyperlane Monorepo
   Hyperlane is a modular interchain protocol, with various parts:
   - Mailbox: Mailbox is the low-level part of the messaging protocol. It is the place where the message arrives and departs from. Relayers are watching the mailbox at all times to look for new messages to route to another chain. Relayers will invoke the Mailbox contract on the target chain to deliver an interchain message.
   - Router: The router is the high-level part of the messaging protocol. It is used to interact with the *Mailbox* contract, and can be extended to offer many capabilities like messaging, bridging tokens and making calls to another chain. The protocol offers *Router* implementations providing these functionalities: *InterchainQueryRouter*, *InterchainAccountRouter*, *LiquidityLayerRouter*. Routers are built according to a structure that allows them to be deployed to multiple chains without code changes.
     - *InterchainQueryRouter*: used to query the state on remote chains via interchain view calls.
     - *IterchainAccountRouter*: used to create an account on a remote chain, and use that account to call smart contracts. Remote accounts are owned by the protocol.
     - *LiquidityLayerRouter*: used to wrap around several token bridges to allow developers to send tokens alongside their message.
   - HyperlaneConnectionClient: mix-in contract that maintains pointers to the three contracts Hyperlane developers may need to interact with: *Mailbox* (required), *InterchainGasPaymaster* (optional), *InterchainSecurityModule* (optional).
   - Validators: Validators create the signMailbox merkle roots and make their signatures available to relayers.
   - ISM: Interchain security modules (ISMs) are smart contracts that define the security model for an application. They can be

user defined or the protocol default ones can be used. At the moment the only ISM provided by the protocol is *MultisigIsm*, a *t-of-n* security module. ISMs must implement the *verify()* interface, which gets called by the *Mailbox* before delivering a message. If *verify()* does not return true, the transaction will revert.

- Relayer: aggregates the off-chain metadata needed to deliver messages (e.g. validator signatures and merkle proofs) and submits them to Mailboxes by calling *Mailbox.process()*, which in turn will invoke *IMessageRecipient.handle()* on the destination chain Router.
- Watch towers: observe the network for validator fraud. If detected, watchtowers submit evidence to the source chain, slashing the fraudulent validator(s). The slashing process is frictionless because each validator only validates on the chain they are staking on.

This is the execution flow of a general message between chains:



Privileged roles

- Protocol deployer - *onlyOwner*: the owner of the protocol can set the mailbox, interchain gas payment and interchain security module in the hyperlane connect client, set the default interchain security module in the mailbox, and also pause and unpause the mailbox, enroll a new router in the router contract, enroll, unenroll and set thresholds for validators in the interchain security module, set a liquidity layer adapter in the liquidity layer router contract, and add a domain and add/remove a token in the adapter contract.

- Mailbox - *onlyMailbox*: can call the handle function in the router.
- Router - *onlyRemoteRouter*: can be the original sender of a message dispatched to the handle function in the router.
- LiquidityLayerRouter - *onlyLiquidityLayerRouter* in CircleBridgeAdapter.sol: can send and receive tokens.

Risks

- The Token repository makes use of the unaudited package @hyperlane-xyz.
- Contracts are upgradable.
- Part of the protocol is off-chain and out of the scope of this audit.
- InterchainAccountRouter does not implement a callback functionality to inform the origin chain about state changing operations performed on the remote chain.
- LiquidityLayerRouter can only send tokens to contracts implementing the receiveTokens() function.

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

H02. Upgradeability Errors

CircleBridgeAdapter.sol is inheriting Router, but is not initializing it. Issue M09 is related to this situation.
OwnableMulticall.sol is inheriting OwnableUpgradable, but is not initializing it.

Path: monorepo/solidity/contracts/middleware/CircleBridgeAdapter.sol: initialize()

Recommendation: Initialize the inherited upgradable contracts.

Status: Fixed (Revised commit: def4031)

H03. Upgradeability Errors

In *InterchainGasPaymaster.sol* the *initialize()* function is inside the constructor, the constructor in the upgradable contract should only contain immutable variable declarations or the *_disableInizializer()* function.

Path:monorepo/solidity/contracts/InterchainGasPaymaster.sol

Recommendation: Remove the constructor to follow best practices for upgradable contracts.

Status: Fixed (Revised commit: def4031)

H05. Data Inconsistency

InterchainQueryRouter is the Router contract used to poll view functions on a remote chain. By design, *InterchainQueryRouter* on the remote chain will perform a callback on the origin chain to communicate the results of the calls. The calls on the remote chain are performed through the function *OwnableMulticall._call()*, called by *InterchainQueryRouter._handle()*.

In case of a failed call, *OwnableMulticall._call()* will revert, so the flow of *InterchainQueryRouter._handle()* will stop before calling *_dispatch()* which would take care of sending the callback to the origin chain.

In this case, the origin chain will receive no callbacks, and will not be informed of the failed call on the remote chain. Even if the

call failed and there is no value to communicate to the origin chain, a flag informing of the failure should be sent back.

Path: monorepo/solidity/contracts/middleware/InterchaiQueryRouter.sol : query(), _handle()
monorepo/solidity/contracts/Mailbox.sol : dispatch(), process()
monorepo/solidity/contracts/OwnableMulticall.sol : _call()

Recommendation: Instead of reverting, a failure flag should be sent to the origin chain.

Status: Mitigated (Documentation updated to reflect this behavior)

H07. Compilation Error

The *Create2Factory* contract implements custom errors, a solidity functionality introduced on version 0.8.4.

The pragma version defined in the contract is *^0.8.0*, making it possible to raise compilation errors.

Path: monorepo/solidity/contracts/Create2Factory.sol

Recommendation: Adjust pragma version

Status: Fixed (Revised commit: def4031)

Medium

M02. Missing SafeERC20

The project does not implement SafeERC20 for ERC20 transfers. This may expose the protocol to denial of service vulnerabilities or data inconsistencies during interactions with non-standard tokens.

This issue has been acknowledged by developers through *TODO* comments in contracts *LiquidityLayerRouter.sol* and *CircleBridgeAdapter.sol*

Path:
monorepo/solidity/contracts/middleware/liquidity-layer/LiquidityLayerRouter.sol : dispatchWithTokens()

monorepo/solidity/contracts/middleware/liquidity-layer/adapters/CircleBridgeAdapter.sol : receiveTokens()

Recommendation: Implement the SafeERC20 library to safely interact with tokens.

Status: Fixed (Revised commit: def4031)

M03. Best Practice Violation

The functions do not use the *SafeTransferFrom()* function or the *SafeMint()* function for checking if the recipient address can receive ERC721 token transfers. The recipient might not be able to handle ERC721 tokens and to transfer them back.

Issue M06 is related to this situation.

Path: token/contracts/HypERC721Collateral.sol: _transferTo(), _transferFromSender();

token/contracts/HypERC721.sol: _transferTo();

Recommendation: use *SafeTransferFrom()* and *SafeMint()* functions to interact with tokens safely.

Status: Fixed (Revised commit: def4031)

M04. Unfinalized code

The code contains multiple *TODO* comments about functionalities yet to be implemented or code sections to be rewritten.

Some of these situations have already been addressed in the identified issues in this audit.

Path: monorepo/solidity/contracts/Router.sol : handle()

monorepo/solidity/contracts/OwnableMulticall.sol : proxyCallBatch()

monorepo/solidity/contracts/middleware/InterchaiQueryRouter.sol : query(), _handle()

monorepo/solidity/contracts/middleware/liquidity-layer/LiquidityLayerRouter.sol : dispatchWithTokens()

Recommendation: Finalize functionalities implementations.

Status: Fixed (Revised commit: def4031)

M05. Copy of well-known contract

The project uses *PausableReentrancyGuard.sol*, a merged version of OpenZeppelin's *ReentrancyGuardUpgradeable* and *PausableUpgradeable* contracts.

While this does not pose a direct security threat, well-known contracts from projects like OpenZeppelin should be imported directly from the source as the projects are in development and may update the contracts in the future.

Path: monorepo/solidity/contracts/PausableReentrancyGuard.sol

Recommendation: Import ReentrancyGuardUpgradeable and PausableUpgradeable from the source and use them separately.

Status: Mitigated (The merged contract implements gas optimization that saves users of Hyperlane >2k gas on every Mailbox.process invocation)

M08. Data Consistency

hyperlaneDomainToCircleDomain mapping can be ambiguous, as in Circle domain the Ethereum chain is encoded as *'0'*.

This issue has been already identified by the developers and highlighted in a TODO comment on line 25.

Path:
monorepo/solidity/contracts/middleware/liquidity-layer/adapters/CircleBridgeAdapter.sol

Recommendation: An easy solution would be to add a *+1* offset to the mapping to make it base 1 instead of base 0.

Status: Reported

M09. Best Practices

CircleBridgeAdapter.sendTokens() performs a check on *_destinationDomain*, to verify that there is a valid router on the destination domain.

As stated in a comment on lines 162-163, *CircleBridgeAdapter* inherits *Router* only to perform this check.

This is inefficient as all remote router addresses shall be added and kept updated in the *_router* Map variable on this contract, only to perform this check.

It would suffice to move this check to the invoker contract *LiquidityLayerRouter*, which is a *Router* and already has the *_router* variable filled and updated.

This piece of code contains another issue: the check is performed without using the dedicated function *Router._mustHaveRemoteRouter()*, the code is duplicated instead.

Path:
monorepo/solidity/contracts/middleware/liquidity-layer/adapters/CircleBridgeAdapter.sol : sendTokens()

Recommendation: Move the check to *LiquidityLayerRouter.dispatchWithTokens()* and make use of the function *Router._mustHaveRemoteRouter()*

Status: Reported

M10. Upgradeability Errors

In *InterchainAccountRouter.sol* the constructor is combined with the initialize function, which is not following best practices for upgradability contracts.

Path:
monorepo/solidity/contracts/middleware/InterchainAccountRouter.sol

Recommendation: Use only the initialize function instead of the constructor.

Status: Mitigated (The constructor allows for consistent gas savings on transactions)

Low

L01. Inefficient Gas Model

Length computation for arrays is often included in the for loops condition instead of being catched in a local variable.

In these cases, the Solidity compiler will always read the length of the array during each iteration.

Instead, creating a new variable *uint256 _length* and iterating *for (uint256 i = 0; i < _length; ++i)* will be much cheaper in terms of Gas.

This is most impactful for storage variable read operations, but it is advised as a best practice in any case.

Path: monorepo/solidity/contracts/Router.sol : enrollRemoteRouters()

monorepo/solidity/contracts/OwnableMulticall.sol : proxyCalls(), _call(), proxyCallBatch()

monorepo/solidity/contracts/MultisigIsm.sol : enrollValidators(), setThresholds()

Recommendation: Load the length of the arrays in a new local variable and use it in the loop iteration.

Status: Fixed (Revised commit: def4031)

L02. Unemitted Events

InterchainAccountRouter and *LiquidityLayerRouter* contracts do not emit events on send and receive situations while *InterchainQueryRouter* does.

Path: monorepo/contracts/middleware/InterchainAccountRouter.sol;

monorepo/contracts/middleware/liquidity-layer/InterchainAccountRouter.sol;

token/contracts/HypERC20Collateral.sol: _transferFromSender(), _transferTo();

token/contracts/HypeERC721Collateral.sol: _transferFromSender(), _transferTo();

Recommendation: Add, send and receive events to *InterchainAccountRouter* and *LiquidityLayerRouter* to keep the flow consistent among routers.

Status: Mitigated (The InterchainQueryRouter emits events on dispatch so that they can be matched with QueryResolved events)

L03. Function Visibility

public functions that are never called by the contract should be declared *external* to save gas.

Path: monorepo/solidity/contracts/Mailbox.sol : latestCheckpoint()

monorepo/solidity/contracts/isms/MultisigIsm.sol : verify()

monorepo/solidity/contracts/middleware/InterchainAccountRouter.sol : initialize(), getInterchainAccount()

monorepo/solidity/contracts/middleware/InterchainQueryRouter.sol : initialize()

monorepo/solidity/contracts/middleware/liquidity-layer/LiquidityLayerRouter.sol : initialize()

monorepo/solidity/contracts/middleware/liquidity-layer/adapters/CircleBridgeAdapter.sol : initialize()

Recommendation: Change function visibility to external for functions never called internally by the contract.

Status: Fixed (Revised commit: def4031)

L04. Boolean Equality

Boolean constants can be used directly and do not need to be compared to true or false.

Path: monorepo/contracts/mailbox: process()

Recommendation: Remove boolean equality.

Status: Mitigated (The comparison to false is done for readability purposes)

L06. Floating Pragma

The project uses floating pragmas.

Path: monorepo/contracts; token/contracts;

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Mitigated (To be permissive with devs implementing the protocol)

L07. Outdated Solidity Version

Using an outdated compiler version can be problematic, especially if publicly disclosed bugs and issues affect the current compiler version.

Path: monorepo/contracts; token/contracts;

Recommendation: Use a recent compiler version.

Status: Reported

L08. Misleading Require Message

Some messages in the require conditions are not descriptive.

This makes the code harder to test and debug, as well as making the User Experience worse.

Path: monorepo/contracts; token/contracts;

Recommendation: Refactor messages in the require conditions to fit code behavior.

Status: Reported

L09. Empty Constructor

In contract *MultisigIsm* the constructor is empty, which makes it redundant due to default Solidity behavior to create an empty constructor if it's not included in the code.
This makes parts of the code redundant.

Path: monorepo/solidity/contracts/isms/MultisigIsm.sol

Recommendation: Remove the constructor.

Status: Mitigated (This constructor is included for readability purposes invoking the super Ownable contract's constructor. It is very easy to miss this invocation when it is appended to the contract header.)

L10. Missing zero address validation

Address parameters are used without checking against the possibility of 0x0.

Path: monorepo/solidity/contracts/HyperlaneConnectionClient.sol : _setMailbox(), setInterchainSecurityModule()

monorepo/solidity/contracts/OwnableMulticall.sol : proxyCallBatch()

Recommendation: Add zero address checks.

Status: Fixed (Revised commit: def4031)

L11. Style Guide Violation

The provided projects should follow official guidelines.

Path: monorepo/contracts;

Recommendation: Folrw the official Solidity guidelines.

Status: Fixed (Revised commit: def4031)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.