Hyperlane Security Audit

: Hyperlane

Dec 3, 2024

Revision 1.0

ChainLight@Theori

Theori, Inc. (“We”) is acting solely for the client and is not responsible to any other party. Deliverables are valid for and should be used solely in connection with the purpose for which they were prepared as set out in our engagement agreement. You should not refer to or use our name or advice for any other purpose. The information (where appropriate) has not been verified. No representation or warranty is given as to accuracy, completeness or correctness of information in the Deliverables, any document, or any other information made available. Deliverables are for the internal use of the client and may not be used or relied upon by any person or entity other than the client. Deliverables are confidential and are not to be provided, without our authorization (preferably written), to entities or representatives of entities (including employees) that are not the client, including affiliates or representatives of affiliates of the client.

Table of Contents

Hyperlane Security Audit 1
Table of Contents 2
Executive Summary 3
Audit Overview 4
- Scope 4
- Code Revision 5
- Severity Categories 5
- Status Categories 6
- Finding Breakdown by Severity 7

Findings 8
- Summary 8
- #1 HL2408-001 ISM possible fund lock/theft issue 10
- #2 HL2408-002 OPL2ToL1Ism can be bypassed (Any message can be delivered) 13
- #3 HL2408-003 ACL or replay protection must be added to AbstractMessageIdAuthHook.postDispatch() 15
- #4 HL2408-004 Users assets could be locked in the interchain account 17
- #5 HL2408-005 _sendMessageId() of OPStackHook and ArbL2ToL1Hook should consider the case where msg.value and metadata.msgValue(0) are different 19
- #6 HL2408-006 HypERC4626 does not work as a rebasing token 21
- #7 HL2408-007 Wrapped HypERC4626 should be implemented for interoperability 22
- #8 HL2408-008 Custom Hook Quote is not considered in the message dispatch process 23
- #9 HL2408-009 HypERC4626OwnerCollateral._transferFromSender() must return HypERC4626.PRECISION on the remote chain 25
- #10 HL2408-010 AbstractAggregationIsm.verify() fails when M is not equal to N 26
- #11 HL2408-011 HypERC4626._handle() should restrict out-of-order updates of the exchangeRate 28
- #12 HL2408-012 Minor Issues 30
- Revision History 33

Executive Summary

Starting on Aug 19th, 2024, ChainLight of Theori audited the smart contract of Hyperlane for three weeks. In the audit, we primarily considered the issues/impacts listed below.

- Bypass message validation
- Temporary Fund Freeze
- Incorrect rebasing token implementation

As a result, we identified issues as listed below.

- Total: 12
- Critical: 1 (Bypass message validation)
- High: 3 (Temporary Fund Freeze)
- Medium: 4 (Incorrect rebasing token implementation, ...)
- Low: 1
- Informational: 3

Audit Overview

Scope

Name	Hyperlane Security Audit
Target / Version	- Git Repository (`hyperlane-xyz/hyperlane-monorepo`): commit `469f2f34030d9539f2038df302195b6a2dbc94c6` (Apply patch commit) - `git diff v3-solidity main --numstat ./solidity/contracts`
Application Type	Smart contracts
Lang. / Platforms	Smart contracts [Solidity]

Code Revision

N/A

Severity Categories

Severity	Description
Critical	The attack cost is low (not requiring much time or effort to succeed in the actual attack), and the vulnerability causes a high-impact issue. (e.g., Effect on service availability, Attacker taking financial gain)
High	An attacker can succeed in an attack which clearly causes problems in the service's operation. Even when the attack cost is high, the severity of the issue is considered "high" if the impact of the attack is remarkably high.
Medium	An attacker may perform an unintended action in the service, and the action may impact service operation. However, there are some restrictions for the actual attack to succeed.
Low	An attacker can perform an unintended action in the service, but the action does not cause significant impact or the success rate of the attack is remarkably low.
Informational	Any informational findings that do not directly impact the user or the protocol.
Note	Neutral information about the target that is not directly related to the project's safety and security.

Status Categories

Status	Description
Reported	ChainLight reported the issue to the client.
WIP	The client is working on the patch.
Patched	The client fully resolved the issue by patching the root cause.
Mitigated	The client resolved the issue by reducing the risk to an acceptable level by introducing mitigations.
Acknowledged	The client acknowledged the potential risk, but they will resolve it later.
Won't Fix	The client acknowledged the potential risk, but they decided to accept the risk.

Finding Breakdown by Severity

Category	Count	Findings
Critical	1	• HL2408-002
High	3	• HL2408-001 • HL2408-003 • HL2408-009
Medium	4	• HL2408-004 • HL2408-006 • HL2408-010 • HL2408-011
Low	1	• HL2408-005
Informational	3	• HL2408-007 • HL2408-008 • HL2408-012
Note	0	• N/A

Findings

Summary

#	ID	Title	Severity	Status
1	`HL2408-001`	ISM possible fund lock/theft issue	High	Won't Fix
2	`HL2408-002`	OPL2ToL1Ism can be bypassed (Any message can be delivered)	Critical	Patched
3	`HL2408-003`	ACL or replay protection must be added to `AbstractMessageIdAuthHook.postDispatch()`	High	Patched
4	`HL2408-004`	Users assets could be locked in the interchain account	Medium	Patched
5	`HL2408-005`	`_sendMessageId()` of `OPStackHook` and `ArbL2ToL1Hook` should consider the case where `msg.value` and `metadata.msgValue(0)` are different	Low	Patched
6	`HL2408-006`	`HypERC4626` does not work as a rebasing token	Medium	Patched
7	`HL2408-007`	Wrapped `HypERC4626` should be implemented for interoperability	Informational	Patched
8	`HL2408-008`	Custom Hook Quote is not considered in the message dispatch process	Informational	Won't Fix
9	`HL2408-009`	`HypERC4626OwnerCollateral._transferFromSender()` must return `HypERC4626.PRECISION` on the remote chain	High	Patched

#	ID	Title	Severity	Status
10	`HL2408-010`	`AbstractAggregationIsm.verify()` fails when M is not equal to N	Medium	Won't Fix
11	`HL2408-011`	`HypERC4626._handle()` should restrict out-of-order updates of the `exchangeRate`	Medium	Patched
12	`HL2408-012`	Minor Issues	Informational	Patched

#1 HL2408-001 ISM possible fund lock/theft issue

ID	Summary	Severity
HL2408-001	Publicly accessible functions (i.e., `verify`, `releaseValueToRecipient`) on the ISM contracts allow any user to prematurely trigger the transfer of funds intended to a recipient, which disrupts the atomicity of the message processing. This premature release can lead to a scenario where funds transferred from various chain-specific portal contracts are locked and become irretrievable.	High

Description

When the recipient uses one of the chain's portal contract (e.g., Portal for Optimism, Outbox for Arbitrum, ULN/Endpoint for LayerZero), the `verifyMessageId` function, which handles the message verification, is called by the portal contract and stores the `msg.value`. This `msg.value` is intended to be transferred to the recipient when the Mailbox calls `verify` function.

Under normal operation, the Mailbox calls `ISM.verify` messages through the `process` function. If the message is successfully verified, the specified recipient receives the `msg.value` via the `releaseValueToRecipient` function on the ISM:

```
// solidity/contracts/isms/hook/AbstractMessageIdAuthorizedIsm.sol
    function verifyMessageId(bytes32 messageId) public payable virtual {
        require(
            _isAuthorized(),
            "AbstractMessageIdAuthorizedIsm: sender is not the hook"
        );
        require(
            msg.value < 2 ** VERIFIED_MASK_INDEX,
            "AbstractMessageIdAuthorizedIsm: msg.value must be less than 2
^255"
        );

        verifiedMessages[messageId] = msg.value.setBit(VERIFIED_MASK_INDEX
);
        emit ReceivedMessage(messageId);
    }
```

However `releaseValueToRecipient` is publicly accessible function that allows any user to prematurely trigger the release of funds to the recipient of the message. This premature release can lead to a scenario where funds transferred from various chain-specific portal contracts are locked and become irretrievable.

```
// solidity/contracts/isms/hook/AbstractMessageIdAuthorizedIsm.sol
    function releaseValueToRecipient(bytes calldata message) public {
        bytes32 messageId = message.id();
        uint256 _msgValue = verifiedMessages[messageId].clearBit(
            VERIFIED_MASK_INDEX
        );
        if (_msgValue > 0) {
            verifiedMessages[messageId] -= _msgValue;
            payable(message.recipientAddress()).sendValue(_msgValue);
        }
    }
```

Impact

High

It can lead to the user funds losses as the funds sent from various chain-specific portal contracts could be locked or exploited.

- ideal scenario
  - relayer (or someone) finalize the L2 transaction
  - relayer/user calls process
    - process calls ism.verify
      - ism.verify releases the funds to the message's recipient if the bridged transaction has value
      - call handle function of the recipient
        - recipient uses `address(this).balance`, since ism transfers the fund to the recipient.

- attack scenario
  - relayer (or someone) finalize the L2 transaction

    - attacker calls ism.verify or ism.releaseValueToRecipient
        - it release the funds to the message's recipient if the bridged transaction has value.
        - and subtract the value to 0.
    - attacker calls process
        - recipient uses victim's value (balance)
    - relayer/user calls process
        - process calls ism.verify
            - ism.verify returns success, but it does not transfer any funds since there isn't remained fund.
            - call handle function of the recipient
                - recipient uses `address(this).balance`, but it's 0 at this moment.

Recommendation

The monkey patch could be add ACL on the both functions. (only callable by Mailbox).

Remediation

Won't Fix

The team responded that they cannot add ACL to the verify function due to the ISM composability requirement.

#2 `HL2408-002` OPL2ToL1Ism can be bypassed (Any message can be delivered)

ID	Summary	Severity
`HL2408-002`	`OPL2ToL1Ism._verifyWithPortalCall()` does not check `metadata`, which allows any messages to be delivered.	Critical

Description

The `OPL2ToL1Ism._verifyWithPortalCall()` does not check `metadata`. While `OPL2ToL1Ism.verify()` is intended to check messages delivered by the `OPL2ToL1Hook`, the lack of `metadata` check enables this process to be bypassed. Consequently, an attacker can bypass the `OPL2ToL1Ism` verification process by sending a transaction directly to the OP portal.

Impact

Critical

An attacker can craft messages that bypass `OPL2ToL1Ism` verification.

Recommendation

1. it mitigates these kind of issues.

```
  function verify(
    bytes calldata metadata,
    bytes calldata message
  ) external override returns (bool) {
    bool verified = isVerified(message);
    if (!verified) {
      _verifyWithPortalCall(metadata, message);
    }
+   require(isVerified(message));
    releaseValueToRecipient(message);
    return true;
  }
```

2. properly check the L2's sender like ArbL2ToL1Ism does.

```
        // check if the sender of the l2 message is the authorized hook
        require(
            l2Sender == TypeCasts.bytes32ToAddress(authorizedHook),
            "ArbL2ToL1Ism: l2Sender != authorizedHook"
        );
```

Remediation

Patched

The issue has been resolved as recommended.

#3 `HL2408-003` ACL or replay protection must be added to `AbstractMessageIdAuthHook.postDispatch()`

ID	Summary	Severity
`HL2408-003`	When a user sends a message using an ISM that inherits `AbstractMessageIdAuthorizedIsm` with funds, an attacker can replay the message, causing the user's funds to be permanently locked within the ISM.	High

Description

The `AbstractMessageIdAuthHook.postDispatch()` does not verify whether the `msg.sender` is a Mailbox or wether the message is being replayed. Consequently, an attacker can resend the same interchain message immediately after `Mailbox.dispatch()` is called.

For example, suppose Alice sends an interchain message with 1 ether using the OP Stack ISM from L1. The `Mailbox.dispatch()` calls `OPStackHook.postDispatch()`, followed by the `OPStackISM.verifyMessageId(messageId)` being triggered in L2 by the deposit transaction. Under normal circumstances, when `releaseValueToRecipient()` is invoked, the `verifiedMessages[messageId]` (1 ether) will be transferred to the recipient.

```
// solidity/contracts/isms/hook/AbstractMessageIdAuthorizedIsm.sol
    function verifyMessageId(bytes32 messageId) public payable virtual {
        require(
            _isAuthorized(),
            "AbstractMessageIdAuthorizedIsm: sender is not the hook"
        );
        require(
            msg.value < 2 ** VERIFIED_MASK_INDEX,
            "AbstractMessageIdAuthorizedIsm: msg.value must be less than 2^255"
        );

        verifiedMessages[messageId] = msg.value.setBit(VERIFIED_MASK_INDEX);
        emit ReceivedMessage(messageId);
    }
```

However, an attacker can replay the message with 0 `msg.value` by calling `OPStackHook.postDispatch()` immediately after the user calls `Mailbox.dispatch()`. This causes `OPStackISM.verifyMEssageId(messageId)` to be called again, leading to `verifiedMessages[messageId]` being set `0 + VERIFIED_MASK`. Since the `verifiedMessages` value is overwritten with 0, the user's funds become permanently locked within the ISM.

Impact

High

User funds sent via interchain messages can become temporarily locked in the ISM (e.g., OPL2ToL1Ism, ArbL2ToL1Ism, LayerZeroV2Ism).

Recommendation

It is recommended to add the `onlyMailbox()` modifier to `AbstractMessageIdAuthHook._postDispatch()` and to implement replay protection in `AbstractMessageIdAuthorizedIsm.verifyMessageId()`.

Remediation

Patched

Replay protection has been implemented by adding the check `require(verifiedMessages[messageId] == 0)`.

#4 HL2408-004 Users assets could be locked in the interchain account

ID	Summary	Severity
HL2408-004	The `InterchainAccountRouter` must transfer `msg.value` during multicall execution.	Medium

Description

```
// File: https://github.com/hyperlane-xyz/hyperlane-monorepo/blob/b1d8bb87
77684cfc863020d9f2eb170f166c112c/solidity/contracts/middleware/InterchainA
ccountRouter.sol#L293-L310
    function handle(
        uint32 _origin,
        bytes32 _sender,
        bytes calldata _message
    ) external payable override onlyMailbox { // (A)
        (
            bytes32 _owner,
            bytes32 _ism,
            CallLib.Call[] memory _calls
        ) = InterchainAccountMessage.decode(_message);

        OwnableMulticall _interchainAccount = getDeployedInterchainAccount
(
            _origin,
            _owner,
            _sender,
            _ism.bytes32ToAddress()
        );
        _interchainAccount.multicall(_calls); // (B)
    }
```

An interchain account supports to receive a native value from other chains `(A)` , however the interchain account contract does not transfer the received value to the multicall contract.

Impact

Medium

Recommendation

`handle()` should send the value when it calls to the `OwnableMulticall.multicall` on `(B)`. (i.e. add `{value: msg.value}`)

Remediation

Patched

The issue has been resolved as recommended.

#5 `HL2408-005` `_sendMessageId()` of `OPStackHook` and `ArbL2ToL1Hook` should consider the case where `msg.value` and `metadata.msgValue(0)` are different

ID	Summary	Severity
`HL2408-005`	The `_sendMessageId()` in `OPStackHook` and `ArbL2ToL1Hook` should consider scenarios where `msg.value` and `metadata.msgValue(0)` do not match.	**Low**

Description

The `_sendMessageId()` in `OPStackHook` and `ArbL2ToL1Hook` does not return the excess amount to the user when `msg.value` exceeds `metadata.msgValue(0)`. In such cases, the excess remains in the Hook contract. Moreover, it does not validate the case where `msg.value` is less than `metadata.msgValue(0)`, allowing an attacker to set `msg.value` to `0` and `metadata.msgValue(0)` to the ETH balance held by the Hook contract, thereby enabling the theft of ETH left in the contract.

Impact

Low

If `msg.value` is greater than `metadata.msgValue(0)`, the excess ETH will be retained in the contract, which an attacker could later extract by setting a higher `metadata.msgValue(0)`.

Recommendation

1. Add a check `require(msg.value >= metadata.msgValue(0));` in the `_sendMessageId()` of `OPStackHook` and `ArbL2ToL1Hook`.
2. Implement logic to return any excess amount to the user (`message.sender()`) if msg.value exceeds `metadata.msgValue(0)`.

Remediation

Patched

The issue has been resolved as recommended.

#6 `HL2408-006` `HypERC4626` does not work as a rebasing token

ID	Summary	Severity
`HL2408-006`	`HypERC4626` was intended to be a rebasing token; however, some function overrides are missing.	Medium

Description

A rebasing token is designed to have its total supply fluctuate dynamically. The `HypERC4626` contract is a rebasing token that reflects the yields generated on the origin chain to the token balance. However, the contract does not fully support rebasing functionality because the `transferFrom()` and `totalSupply()` functions were not overridden to support the rebasing token function.

Impact

Medium

Due to inconsistent handling between underlying amounts and share amounts, users may experience confusion. Some functions operate based on the underlying amount (`transfer()`), while others rely on the share amount (`transferFrom()`).

Recommendation

1. Override `transferFrom()` to function similarly to the redefined `transfer()` .
2. Override `totalSupply()` to return `sharesToAssets(super.totalSupply())` .

Remediation

Patched

The issue has been resolved as recommended.

#7 `HL2408-007` Wrapped `HypERC4626` should be implemented for interoperability

ID	Summary	Severity
`HL2408-007`	Given that `HypERC4626` is a rebasing token, it's clear that integrating it into DeFi protocols presents significant challenges. To address this, we recommend the creation of a wrapper contract to facilitate its use.	Informational

Description

Rebasing tokens dynamically adjusts token balances, which can present difficulties when used in DeFi protocols. For example, if a pair on UniswapV2 includes a rebasing token, the yield generated can be claimed by anyone via `skim()`. This scenario forces liquidity providers who supply rebasing tokens to forgo their yield while maintaining their liquidity positions, which is unfavorable.

As a result, many rebasing tokens utilize wrapped rebasing token contracts. These contracts transfer rebasing tokens based on share units and return balances in share units, preventing automatic balance adjustments.

Impact

Informational

Recommendation

Developing wrapped rebasing token contracts like `wstETH` to support `HypERC4626` in DeFi protocols is recommended.

Remediation

Patched

The issue has been resolved as recommended.

#8 `HL2408-008` Custom Hook Quote is not considered in the message dispatch process

ID	Summary	Severity
`HL2408-008`	The Message Dispatch Process doesn't factor in Custom Hook Quotes when calculating the Relay Fee Quote. The relay process will always fail if the system includes an expensive Custom Hook.	Informational

Description

The `Mailbox.dispatch()` function contains the following logic to handle underpayments:

```
// solidity/contracts/Mailbox.sol#L298-L305
// ...
uint256 requiredValue = requiredHook.quoteDispatch(metadata, message);
if (msg.value < requiredValue) {
    msg.value = requiredValue;
}
requiredHook.postDispatch{value: requiredValue}(metadata, message);
hook.postDispatch{value: msg.value - requiredValue}(metadata, message);
// ...
```

This causes a revert in the `requiredHook` when the `msg.value` is less than the required amount. However, the calculation of `requiredValue` only considers the result from `requiredHook.quoteDispatch()` and does not consider the value from `hook.quoteDispatch()` when determining the total amount needed for dispatch.

Impact

Informational

The relay process will fail under these conditions:

1. The `requiredHook.quoteDispatch()` function returns 0.

2. `msg.value` is 0 during dispatch.

3. A computationally expensive Custom Hook is used.

Recommendation

Modify the `requiredValue` calculation in the `Mailbox.dispatch()` function to include both `requiredHook.quoteDispatch(metadata, message)` and `hook.quoteDispatch(metadata, message)`.

This adjustment ensures that the costs from both `requiredHook` and `hook` are considered, aligning the logic with the intended design of the `Mailbox.quoteDispatch()` function. (`requiredValue = requiredHook.quoteDispatch(metadata, message) + hook.quoteDispatch(metadata, message)`)

Remediation

Won't Fix

The client said that this is a known gas optimization issue and stated that callers must handle quoting outside of the mailbox, ideally off-chain.

#9 `HL2408-009`

`HypERC4626OwnerCollateral._transferFromSender()` must return `HypERC4626.PRECISION` on the remote chain

ID	Summary	Severity
`HL2408-009`	`HypERC4626OwnerCollaeteral` sends a message with incorrect token metadata, which causes the sender's funds to be temporarily frozen in the contract.	High

Description

The `HypERC4626OwnerCollateral` contract sends a message with `_tokenMetadata` as `bytes("")` to `HypERC4626` on the remote chain. `HypERC4626._handle()`, which is called during message processing in the remote chain, tries to `abi.decode()` the passed empty token metadata into `uint256`. This always results in a revert.

As a result, the message sender cannot receive the message on the remote chain, and funds become temporarily locked in the `HypERC4626OwnerCollateral` contract.

Impact

High

The sender's deposited assets at the origin chain to be temporarily frozen because of inability to receive messages on the remote chain. (Since the `HypERC4626OwnerCollateral` contract is a proxy implementation, the owner can recover the locked assets through a contract upgrade.)

Recommendation

It is recommended that `HypERC4626OwnerCollateral._transferFromSender()` be modified to return the `HypERC4626.PRECISION` value from the remote chain.

Remediation

Patched

The issue has been resolved as recommended.

#10 `HL2408-010` `AbstractAggregationIsm.verify()` fails when M is not equal to N

ID	Summary	Severity
`HL2408-010`	`AbstractAggregationIsm.verify()` will only succeed when exactly the required number of ISM verifications pass, due to multiple issues.	Medium

Description

`AbstractAggregationIsm.verify()` reverts the entire transaction if any ISM verification fails or throws an error, preventing partial success in the intended `m-of-n` verification model. Additionally, the function decrements a threshold for each successful verification, which can lead to integer underflow if more ISMs pass than required, causing the transaction to fail unexpectedly.

In the case of `RateLimitedIsm.verify()`, the verification may fail even if a transaction simulation by a relayer was successful, due to state changes by other transactions included earlier in the same block.

Impact

Medium

These issues make `AbstractAggregationIsm.verify()` prone to failure in most cases unless M equals N.

Suppose a relayer simulates ISM verification and only submits the required data. In that case, failures will be rare but still possible when there is a discrepancy between the simulation and actual execution.

Recommendation

Use `try-catch` to handle ISM verification errors, and do not revert when `false` is returned. If less trusted ISMs are included, consider using an assembly call with limited gas and return data size.

Successful verifications should be counted and compared with the threshold rather than decrementing the threshold.

Remediation

Won't Fix

The client said that relayers aim to minimize costs and prefer not to verify more ISMs than necessary.

#11 `HL2408-011` `HypERC4626._handle()` should restrict out-of-order updates of the `exchangeRate`

ID	Summary	Severity
`HL2408-011`	The `HypERC4626._handle()` can decrease the exchange rate if messages are processed out of order, potentially leading to user fund losses in certain situations.	**Medium**

Description

The `HypERC4626._handle()` parses the exchange rate from the token metadata in process messages and saves it in the `exchangeRate` variable. However, an outdated exchange rate may be saved since the message process order from the remote chain is not guaranteed. In a typical ERC4626, the exchange rate monotonically increases unless there has been a loss. However, if messages are processed out of order, the exchange rate could decrease, potentially reducing user token balances and causing losses.

Impact

Medium

When `HypERC4626` is used with DeFi protocols, users' assets could be lost due to the unordered message execution affecting the exchange rate. However, when assets are transferred back to the origin chain, they are sent based on shares, preventing direct loss in such cases.

Recommendation

The following measures are recommended:

1. Include the `nonce` when sending messages from the `HypERC4626Collateral` contract to the remote chain.
2. Record the last `nonce` each time the exchange rate is updated in `HypERC4626._handle()`.
3. Only update the exchange rate if the previous `nonce` is less than the `nonce` in the token metadata.

Remediation

Patched

The issue has been resolved as recommended.

#12 `HL2408-012` Minor Issues

ID	Summary	Severity
`HL2408-012`	The description includes multiple suggestions for preventing incorrect settings caused by operational mistakes, mitigating potential issues, improving code maturity and readability, and other minor issues.	Informational

Description

Operational Risk Mitigation / Sanity Check

- `RateLimited.setRefillRate()` allows unrestricted setting of the `_capacity` value. If `_capacity` is set to a value smaller than `DURATION`, the `refillRate` may be set to zero, causing a revert in `calculateCurrentLevel()`. It is recommended to enforce the condition `_capacity >= DURATION` to ensure a minimum `refillRate`.

- In the `TrustedRelayerIsm` constructor, verify that the `mailbox` and `trustedRelayer` addresses are not zero to prevent configuration errors.

- `AttributeCheckpointFraud`, `Mailbox`, `MailboxClient`, and `ProtocolFee` are recommended to use `Ownable2Step` to avoid the risk of losing ownership.

- In `TypeCasts.bytes32ToAddress()`, add check `require(uint256(_buf) <= uint256(type(uint160).max));` to ensure valid address conversion.

- For `ArbL2ToL1Ism`, `OPL2ToL1Ism._verifyWithPortalCall()`, and `LayerZeroV2Ism.lzReceive()`, it is recommended to verify the function signature using `require(AbstractMessageIdAuthorizedIsm.verifyMessageId == data[0:4]);`.

- In `HypERC4626Collateral` and `HypERC4626._transferRemote()`, add `require(address(hook) == _hook);` to prevent potential user fund losses by ensuring that the correct hook is specified.

- It is advised to adjust the `PRECISION` value in `HypERC4626` and `HypERC4626Collateral` from `1e10` to `IERC20(vault.asset()).decimals()` to avoid precision loss.

- In `ArbL2ToL1Ism.verify()`, after executing `_verifyWithOutboxCall()`, add `require(isVerified(message));` for added security, similar to recommendations from issue [HL2408-01].

- In `HypERC20Collateral.constructor()`, verify that the `erc20` address is not zero to prevent issues.

- (EigenLayer) The `ECDSAStakeRegistry._updateMinimumWeight()` function should validate that the `_newMinimumWeight` value is greater than the minimum threshold.

- (EigenLayer) In the constructor of `ECDSAStakeRegistry`, the `_disableInitializers()` function is commented out. This should be re-enabled in the production environment to ensure initialization safety.

Code Maturity

- The condition `metadata.msgValue(0) < 2 ** 255` in `OPStackHook._sendMessageId()` is redundant since it is already validated in `_postDispatch()`. Removing this condition is recommended to improve code readability.

Other Recommendations

- In `OPL2ToL1Hook._sendMessageId()`, replace `require(msg.value >= metadata.msgValue(0) + GAS_QUOTE)` with `require(msg.value >= metadata.msgValue(0))`, as `GAS_QUOTE` is already accounted for in L2 execution.

- The `exchangeRate` calculation in `HypERC4626Collateral` may differ from the ERC4626 standard. Modify the calculation to `vault.convertToAssets(PRECISION)` for consistency.

- `HypERC4626Collateral.rebase()` should allow specifying hook metadata and hook addresses to prevent failures during message processing on the remote chain. Modify the function to accept these parameters.

- In `InterchainAccountRouter.handle()`, it is crucial to verify that `_sender` is an authorized InterchainAccountRouter from the origin chain to prevent manipulation of `_owner` and `_ism`. This validation should be added.

- `InterchainAccountRouter` lacks a dispatch function to specify the `_hook` address. Add this functionality to ensure proper message handling, especially if `isms` are set to non-default ISM addresses.

- `HypERC4626Collateral` could be vulnerable to inflation attacks when using ERC4626 contracts prior to version 4.9. Developers should use ERC4626 version 4.9 or higher or account for inflation risks.

- (EigenLayer) The `ECDSAStakeRegistry.isValidSignature()` function does not fully comply with the ERC-1271 standard. It should handle failure cases in `_checkSignatures()` using a try-catch block and return the function selector `0xffffffff` when an exception occurs.

Missing / Confusing Events

- In `RateLimited.validateAndConsumeFilledLevel()`, it is recommended to emit events for changes in `filledLevel` and `lastUpdated` values for operational transparency.
- For `MailboxClient.setHook()` and `MailboxClient.setInterchainSecurityModule()`, emit events when state values are modified to ensure transparency.
- Emit events in `GasRouter.setDestinationGas()` whenever destination gas settings change to provide visibility.
- In `HyperlaneServiceManager`, emit events for changes in `freezeoperator` and `setSlasher` to improve operational clarity.
- `ProtocolFee.setProtocolFee` and `ProtocolFee.setBeneficiary` should emit events when state values are changed to ensure transparency.
- (EigenLayer) For transparency in operations, it's recommended that `setPaymentCoordinator` and `updateAVSMetadataURI` in `ECDSAServiceManagerBase` emit events reflecting their changed state values.

Impact

Informational

Recommendation

Consider applying the suggestions in the description above.

Remediation

Patched

Most of the issues have been resolved.

Revision History

Version	Date	Description
1.0	Dec 3, 2024	Initial version

Theori, Inc. (“We”) is acting solely for the client and is not responsible to any other party. Deliverables are valid for and should be used solely in connection with the purpose for which they were prepared as set out in our engagement agreement. You should not refer to or use our name or advice for any other purpose. The information (where appropriate) has not been verified. No representation or warranty is given as to accuracy, completeness or correctness of information in the Deliverables, any document, or any other information made available. Deliverables are for the internal use of the client and may not be used or relied upon by any person or entity other than the client. Deliverables are confidential and are not to be provided, without our authorization (preferably written), to entities or representatives of entities (including employees) that are not the client, including affiliates or representatives of affiliates of the client.

ChainLight