Treasure - L2 EigenLayer Restaking Security Audit

: Treasure - L2 EigenLayer Restaking via ERC-6551 accounts

Mar 23, 2025

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.

© 2025 ChainLight, Theori. All rights reserved

Table of Contents

Executive Summary

Beginning on February 7, 2025, ChainLight of Theori conducted a two-week security audit of Treasure's L2 EigenLayer Restaking via ERC-6551 accounts. The primary goal of the audit was to identify critical security vulnerabilities and evaluate potential impacts.

Summary of Findings

The audit revealed a total of **seventeen** issues, categorized by severity as follows:

- **Critical:** 1 issue (theft of funds)
- **High:** 3 issues (limited theft of funds, incorrect accounting, etc.)
- **Medium:** 3 issues
- **Low:** 8 issues
- **Informational:** 2 issues

Audit Overview

Scope

Name	Treasure - L2 EigenLayer Restaking Security Audit
Target / Version	• Git Repository (TreasureProject/L2-eigenlayer-restaking): commit `1dcb61a8a13d625e7935b326bb9aad95683ade8a`
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.

© 2025 ChainLight, Theori. All rights reserved

Finding Breakdown by Severity

Category	Count	Findings
Critical	1	• `TREASURE-2502-003`
High	3	• `TREASURE-2502-014` • `TREASURE-2502-015` • `TREASURE-2502-016`
Medium	3	• `TREASURE-2502-001` • `TREASURE-2502-007` • `TREASURE-2502-008`
Low	8	• `TREASURE-2502-002` • `TREASURE-2502-005` • `TREASURE-2502-006` • `TREASURE-2502-009` • `TREASURE-2502-010` • `TREASURE-2502-011` • `TREASURE-2502-012` • `TREASURE-2502-013`
Informational	2	• `TREASURE-2502-004` • `TREASURE-2502-017`
Note	0	• N/A

Findings

Summary

#	ID	Title	Severity	Status
1	`TREASURE-2502-001`	Insufficient Validation of Message in `_depositWithEigenAgent()`	Medium	Patched
2	`TREASURE-2502-002`	Unvalidated `_strategy` Parameter in `_depositWithEigenAgent()`	Low	Patched
3	`TREASURE-2502-003`	Insufficient Validation of `_signer` in `_completeWithdrawalWithEigenAgent()` and `_processClaimWithEigenAgent()`	Critical	Patched
4	`TREASURE-2502-004`	Forward-Compatibility Issue in `_completeWithdrawalWithEigenAgent()`	Informational	Patched
5	`TREASURE-2502-005`	Improper Contract Address Handling in Refund and Withdrawal Functions	Low	Won't Fix
6	`TREASURE-2502-006`	Using SafeERC20 is Recommended	Low	Patched
7	`TREASURE-2502-007`	Front-Running of `executeWithSignature()` May Lead to Denial of Service	Medium	Patched
8	`TREASURE-2502-008`	Non-Compliance with EIP-721 Standards in `EigenAgent6551`	Medium	Patched
9	`TREASURE-2502-009`	Function Selector Should Be Validated in `beforeSendCCIPMessage()`	Low	Patched

#	ID	Title	Severity	Status
10	`TREASURE-2502-010`	Potential Double Refund/Payment Issue in `withdrawTokenForMessageId()`	Low	Patched
11	`TREASURE-2502-011`	Forced Refunds in `ReceiverCCIP` Can Deplete Funds	Low	WIP
12	`TREASURE-2502-012`	Lack of Excess Fee Refund in `sendMessagePayNative()`	Low	Patched
13	`TREASURE-2502-013`	Need for Per-Chain Management of `allowlistedSenders`	Low	Patched
14	`TREASURE-2502-014`	Stale `RewardsCoordinator.claimerFor[]` on `EigenAgent` Ownership Transfer	High	Patched
15	`TREASURE-2502-015`	`SenderCCIP.handleTransferToAgentOwner()` May Return Incorrect Token Amounts	High	Patched
16	`TREASURE-2502-016`	Handling Multiple Tokens Under a Single `transferRoot` Fails in `SenderCCIP`	High	Patched
17	`TREASURE-2502-017`	Minor Suggestions	Informational	Patched

© 2025 ChainLight, Theori. All rights reserved

#1 `TREASURE-2502-001` Insufficient Validation of Message in `_depositWithEigenAgent()`

ID	Summary	Severity
`TREASURE-2502-001`	The `_depositWithEigenAgent()` function in `RestakingConnector.sol` does not verify that the `token` and `amount` parameters match `destTokenAmounts[0].token` and `destTokenAmounts[0].amount`. This could lead to theft of funds if tokens are unexpectedly sent to `ReceiverCCIP`.	Medium

Description

In its current implementation, `_depositWithEigenAgent()` receives a message containing `token` and `amount` fields but does not cross-check them against the values in `destTokenAmounts[0]`. Consequently, an attacker could craft a malicious message and bypass validation, allowing the arbitrary transfer of tokens. Additionally, the existing condition `if (destTokenAmounts.length > 1) { ... }` should be replaced with `if (destTokenAmounts.length != 1) { ... }` for stricter validation.

Impact

Medium

By manipulating the `token` and `amount` fields, an attacker could steal tokens stored in `ReceiverCCIP`, leading to potential financial losses. However, since tokens are generally not held in `ReceiverCCIP`, the losses may be limited.

Recommendation

1. Add strict checks to confirm that `token == destTokenAmounts[0].token` and `amount == destTokenAmounts[0].amount`.
2. Replace `if (destTokenAmounts.length > 1) { ... }` with `if (destTokenAmounts.length != 1) { ... }` to ensure only a single token transfer is processed at a time.

Remediation

Patched

It has been patched as recommended.

#2 `TREASURE-2502-002` Unvalidated `_strategy` Parameter in `_depositWithEigenAgent()`

ID	Summary	Severity
`TREASURE-2502-002`	The `_strategy` parameter passed to `_depositWithEigenAgent()` is not validated against the `strategy` configured in `setEigenlayerContracts()`.	Low

Description

Within `_depositWithEigenAgent()`, the contract decodes `_strategy` from the message but does not verify that it matches a trusted address. As a result, funds could be directed to an unrecognized strategy by mistake.

Impact

Low

Failing to confirm a valid strategy could cause deposits to flow into unintended addresses, creating a risk of loss.

Recommendation

- Validate `_strategy` against a known, trusted address or addresses.
- If multiple strategy addresses should be supported, use a mapping (e.g., `mapping(address => bool)`) instead of a single address variable.

Remediation

Fixed

Multiple strategies are allowed, but they follow the built-in validation of the EigenLayer strategy vault. Therefore, a patch has been made to remove the `_strategy` storage variable set by `setEigenlayerContracts()`.

© 2025 ChainLight, Theori. All rights reserved

#3 `TREASURE-2502-003` Insufficient Validation of `_signer` in `_completeWithdrawalWithEigenAgent()` and `_processClaimWithEigenAgent()`

ID	Summary	Severity
`TREASURE-2502-003`	In `RestakingConnector`, the `_signer` decoded from `messageWithSignature` is not validated against the owner of `eigenAgent`. An attacker may front-run a legitimate CCIP message with modified `_signer` field to steal tokens for withdrawal or rewards.	Critical

Description

Both `_completeWithdrawalWithEigenAgent()` and `_processClaimWithEigenAgent()` rely on a `_signer` derived from the CCIP message but do not verify it against `eigenAgent.owner()`. This allows an attacker to modify the `_signer` value in a legitimate user's CCIP message to their own address and resend a new CCIP message with a higher gas limit.

Because CCIP messages are processed sequentially, if the legitimate user's CCIP message is processed successfully first on L1, the attacker's message fails due to signature reuse. However, if the legitimate user's CCIP message fails or is front-run, the attacker's CCIP message could execute successfully, diverting funds (withdrawn tokens or claimed rewards) intended for the legitimate user.

Impact

Critical

An attacker can intercept withdrawn tokens or claimed rewards by modifying the `_signer` in a valid CCIP message, causing the legitimate user's funds to be diverted if the attacker's message executes first.

Recommendation

Enforce `_signer == eigenAgent.owner()` checks within `_completeWithdrawalWithEigenAgent()` and `_processClaimWithEigenAgent()`. Reject any CCIP message if the signer does not match the actual `eigenAgent` owner.

Remediation

Patched

The logic has been updated so that tokens are sent to the agent's owner address instead of the signer address included in the message.

#4 `TREASURE-2502-004` Forward-Compatibility Issue in `_completeWithdrawalWithEigenAgent()`

ID	Summary	Severity
`TREASURE-2502-004`	EigenLayer's slashing update will change the `DelegationManager.completeQueuedWithdrawals()` interface. The current implementation of `_completeWithdrawalWithEigenAgent()` does not account for potential slashing and needs updating prior to the upgrade.	Informational

Description

`DelegationManager.completeQueuedWithdrawals()` no longer accepts the `uint256[] calldata middlewareTimesIndexes` parameter after the slashing update. Therefore, The `EigenLayerMsgDecoders` and `EigenLayerMsgEncoders` contracts need to be updated to align with the interface changes. Additionally, `RestakingConnector._completeQueuedWithdrawal()` should be modified to account for slashing.

Impact

Informational

Failing to align with the new struct changes could break `RestakingConnector`'s functionality or improperly handle slash events, leading to unexpected outcomes or potentially lost funds.

Recommendation

1. The `EigenLayerMsgDecoders` and `EigenLayerMsgEncoders` contracts need to be updated to align with the interface changes in the `DelegationManager.completeQueuedWithdrawals()` function.
2. Modify `_completeWithdrawalWithEigenAgent()` to handle slash scenarios correctly once the slashing update is deployed.

Remediation

Patched

It has been patched as recommended.

#5 `TREASURE-2502-005` Improper Contract Address Handling in Refund and Withdrawal Functions

ID	Summary	Severity
`TREASURE-2502-005`	If `signer` is an L1 contract that does not exist on L2, sending tokens to the same address on L2 can result in fund loss. To avoid this, tokens should either be directed to `signer` on L1 or handled through `EigenAgent`.	Low

Description

When `_completeWithdrawalWithEigenAgent()` or `_processClaimWithEigenAgent()` is invoked, the contract assumes the `signer` address exists on L2. However, if the `signer` is a contract deployed only on L1, transferring tokens to the same address on L2 can lead to a permanent loss. One alternative solution is to use `msg.sender` in L2 `SenderCCIP` for the `signer` field.

Impact

Low

If the signer address of `EigenAgent` (owner of EigenAgent) is a contract deployed only on L1 and not on L2, sending tokens to the signer address on L2 could result in fund loss.

Recommendation

When sending a message from L2 to L1, `msg.sender` is used as the `signer` value. This ensures that the `signer` exists on L2, allowing tokens to be sent to the `signer` address on L2. Additionally, for tokens that cannot be bridged from L1 to L2, they should be sent to `EigenAgent` instead of `EigenAgent` 's owner.

Remediation

Won't Fix

The team has stated that they will not fix this issue but will provide users with an appropriate risk warning.

#6 `TREASURE-2502-006` Using SafeERC20 is Recommended

ID	Summary	Severity
`TREASURE-2502-006`	Direct calls to `approve`, `transfer`, and `transferFrom` may fail with certain tokens (e.g., USDT). Using OpenZeppelin's `SafeERC20` wrapper ensures more robust token handling.	Low

Description

In functions like `EigenAgent6551.approveByWhitelistedContract()`, a direct `approve` call may fail if a non-zero allowance already exists, as is the case with USDT. By using the `SafeERC20` wrapper, the contract remains functional even under these circumstances, making it more robust.

Impact

Low

Directly invoking ERC20 methods can expose the contract to unexpected errors or silent failures, potentially locking funds or causing transaction failures under specific token implementations.

Recommendation

Replace direct ERC20 function calls with `SafeERC20`-wrapped functions such as `safeApprove`, `safeTransfer`, and `safeTransferFrom`. These methods handle non-standard token behavior and revert on failure.

Remediation

Patched

It has been patched as recommended.

#7 `TREASURE-2502-007` Front-Running of `executeWithSignature()` May Lead to Denial of Service

ID	Summary	Severity
`TREASURE-2502-007`	An attacker could intercept and reuse the signature from an L2-to-L1 CCIP message to front-run `EigenAgent6551.executeWithSignature()`. This could lock the legitimate transaction or skip subsequent logic in the `RestakingConnector`.	Medium

Description

When L2 `SenderCCIP` sends a CCIP message to L1's `ReceiverCCIP`, it eventually calls `EigenAgent6551.executeWithSignature()`. A malicious actor monitoring L2 transactions can intercept the signature and call `executeWithSignature()` on L1 first, preventing the CCIP message from being executed properly. If the CCIP message is not processed properly due to frontrunning, the withdrawal and reward tokens may not be correctly delivered to the L2 user and could become stuck in `EigenAgent`.

Impact

Medium

- Denial of service for valid CCIP transactions.
- Possible blocking of legitimate withdrawals or reward claims.

Recommendation

Restrict `EigenAgent6551.executeWithSignature()` so that only the `RestakingConnector` contract can invoke it. By limiting permitted callers, attackers cannot bypass the CCIP flow and front-run the function.

Remediation

Patched

It has been patched as recommended.

#8 `TREASURE-2502-008` Non-Compliance with EIP-721 Standards in `EigenAgent6551`

ID	Summary	Severity
`TREASURE-2502-008`	`EigenAgent6551` departs from EIP-721 conventions in the hashing of `data` and the calculation of `domainSeparator()`. The current approach allows signature reuse across different `EigenAgent` instances sharing the same owner.	Medium

Description

EigenAgent6551 currently violates two aspects of the EIP-721 standard.

1. in `createEigenAgentCallDigestHash()`, the `data` should be hashed using `keccak256(data)`.
2. The `domainSeparator()` function incorrectly uses `keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256(bytes("EigenLayer")), chainid, contractAddr))`. The EIP-712 specification requires `verifyingContract` to be the address performing signature verification (`address(this)`), not a user-provided contract address. By deviating from this standard, signatures might be reused in multiple `EigenAgent` contracts if they share an owner.

Impact

Medium

Violating the EIP-721 can lead not only to compatibility issues but also to security risks. An incorrect calculation of `domainSeparator()` introduces the risk of signature reuse, allowing the same owner's signature to be used across multiple `EigenAgent` contracts.

Recommendation

- Use `keccak256(data)` for hashing `data` in `createEigenAgentCallDigestHash()`.
- Update the `domainSeparator()` logic to `keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256(bytes("EigenLayer")), chainid, address(this)))` to align with EIP-712 standards.

© 2025 ChainLight, Theori. All rights reserved

Remediation

Patched

It has been patched as recommended.

#9 `TREASURE-2502-009` Function Selector Should Be Validated in `beforeSendCCIPMessage()`

ID	Summary	Severity
`TREASURE-2502-009`	`SenderHooks.beforeSendCCIPMessage()` does not confirm that `functionSelector` corresponds to a recognized function within `RestakingConnector`. An unvalidated selector could lead to unintended operations.	Low

Description

`SenderHooks.beforeSendCCIPMessage` is responsible for validating CCIP messages sent to L1, but it currently allows users to specify an arbitrary function selector. It should be modified to reject function selectors that are not supported by `RestakingConnector`.

Impact

Low

Allowing arbitrary function selectors does not pose an issue at the moment since the L1 `RestakingConnector` does not execute any unintended actions. However, as new functionalities are added in the future, this could introduce unforeseen risks.

Recommendation

It is recommended that `SenderHooks.beforeSendCCIPMessage()` reverts if an unauthorized function selector is used.

Remediation

Patched

It has been patched as recommended.

© 2025 ChainLight, Theori. All rights reserved

#10 `TREASURE-2502-010` Potential Double Refund/Payment Issue in `withdrawTokenForMessageId()`

ID	Summary	Severity
`TREASURE-2502-010`	If the administrator refunds the user via `ReceiverCCIP.withdrawTokenForMessageId()`, but the message is later successfully processed, the user could receive a double payment.	Low

Description

If a CCIP message fails to process correctly, the administrator calls `ReceiverCCIP.withdrawTokenForMessageId()` to issue a refund. However, if the CCIP message is later successfully processed, there is a risk of a double payment.

Impact

Low

If a `messageId` that has already been refunded via `withdrawTokenForMessageId()` is later processed successfully, it may result in a double payment.

Recommendation

It is recommended to modify `_ccipReceive()` so that if the `messageId` has already been refunded via `withdrawTokenForMessageId()`, the function terminates immediately without executing any additional logic.

Remediation

Patched

The patch removes `withdrawTokenForMessageId()`, eliminating the risk of double payments.

#11 `TREASURE-2502-011` Forced Refunds in `ReceiverCCIP` Can Deplete Funds

ID	Summary	Severity
`TREASURE-2502-011`	Attackers can deliberately set a low `expiry` or gas limit to trigger `ReceiverCCIP._refundToSignerAfterExpiry()` repeatedly. This tactic can drain the `ReceiverCCIP` contract’s funds used for covering L1 → L2 refunds.	Low

Description

By crafting CCIP messages with parameters that are guaranteed to fail or expire quickly, attackers can force repeated refund attempts. Although there is no direct financial gain for the attacker, the repeated execution cost depletes `ReceiverCCIP` resources over time.

Impact

Low

- Incremental loss of funds for `ReceiverCCIP`.
- Potential disruption if `ReceiverCCIP` runs out of funds to process legitimate refunds.

Recommendation

- Require an additional refund fee for messages likely to fail or expire.
- Deduct the transaction cost from the token balance in `_refundToSignerAfterExpiry()` on L1 to avoid an open-ended subsidy.

Remediation

WIP

The team said they would proceed with the patch as recommended.

#12 `TREASURE-2502-012` Lack of Excess Fee Refund in `sendMessagePayNative()`

ID	Summary	Severity
`TREASURE-2502-012`	In `BaseMessengerCCIP.sendMessagePayNative()`, users are not refunded if `msg.value` exceeds the actual required fee. This can lead to overpayment when users do not pre-check fees with `RouterClient.getFee()`.	Low

Description

If a user sends a `msg.value` higher than required, the contract does not refund the excess amount. While the contract owner can withdraw these funds using `withdraw()`, there is no direct mechanism to return the excess amount to a specific user, making the process more complicated.

Impact

Low

Users risk losing excess funds. The contract accumulates unnecessary balances, causing potential grievances and operational overhead in returning overpayments.

Recommendation

Implement an immediate refund mechanism that calculates `excess = msg.value - fee` and returns `excess` to the sender within the same transaction. This approach ensures precise fee collection and improved user experience.

Remediation

Patched

It has been patched as recommended.

© 2025 ChainLight, Theori. All rights reserved

#13 `TREASURE-2502-013` Need for Per-Chain Management of `allowlistedSenders`

ID	Summary	Severity
`TREASURE-2502-013`	`allowlistedSenders` does not differentiate addresses across chains, which could be exploited under certain conditions.	Low

Description

Currently, the contract uses a single mapping `mapping(address => bool) allowlistedSenders` for sender allowlisting. Once an address is approved as a sender, it can relay CCIP messages from any chain. This can be exploited if an approved address is a contract address deployed from a shared factory and not yet deployed on a chain (e.g., a smart contract wallet).

Impact

Low

If a certain type of contract address is used as an allowlisted sender and specific conditions are met, an attacker may relay CCIP messages. However, this scenario is not very likely.

Recommendation

Use `mapping(uint64 => mapping(address => bool))` so each chain has its own set of permitted sender addresses.

Remediation

Patched

It has been patched as recommended.

#14 `TREASURE-2502-014` Stale `RewardsCoordinator.claimerFor[]` on `EigenAgent` Ownership Transfer

ID	Summary	Severity
`TREASURE-2502-014`	When `EigenAgent` ownership changes, the `claimerFor[earner]` value in `RewardsCoordinator` remains set to the previous owner. This prevents the new owner from claiming rewards and allows the old owner to continue claiming.	High

Description

If `claimerFor[earner]` is set for a given `EigenAgent`, it persists even if the `EigenAgent` ownership transfers. The new owner cannot claim rewards, while the old owner maintains the right to do so via the existing `claimerFor` assignment.

Impact

High

The new owner of `EigenAgent` cannot claim rewards until `claimerFor[earner]` is reset to `address(0)`, and the previous owner can unfairly claim rewards despite not having ownership of `EigenAgent`.

Recommendation

It is recommended to apply one of the following two measures:

- Convert `EigenAgent` into a soulbound token so it cannot be transferred.
- In the `EigenAgent` contract, call `RewardsCoordinator.setClaimerFor(address(0))` upon ownership transfer.

Remediation

Patched

It has been modified so that the token transfer fails if `claimerFor[eigenAgent]` is not `address(0)`.

#15 `TREASURE-2502-015`

`SenderCCIP.handleTransferToAgentOwner()` May Return Incorrect Token Amounts

ID	Summary	Severity
`TREASURE-2502-015`	`SenderHooks.handleTransferToAgentOwner()` returns token amounts that do not necessarily match the actual tokens received by the `EigenAgent` on L1. The logic incorrectly assumes `withdrawal.shares` or `cumulativeEarnings` equals the final token amount transferred.	High

Description

`SenderHooks.handleTransferToAgentOwner` returns the stored values of `withdrawal shares` or `cumulativeEarnings`, but these may not necessarily match the actual number of tokens sent from L1.

- **Withdrawals**: When L1 `EigenAgent` calls `DelegationManager.completeQueuedWithdrawal()`, `shares` are converted into token amounts, which may differ from the original `shares` value.
- **Rewards**: `cumulativeEarnings` includes all historical earnings, but only the difference from `cumulativeClaimed[earner][tokenLeaf.token]` is actually claimed.

As a result, there may be a discrepancy between the actual number of tokens `EigenAgent` sends from L1 to L2 and the number of tokens sent to the agent owner on L2 based on `SenderHooks.transferCommitmentsAmount[transferRoot]`.

Impact

High

Incorrect token accounting can lead to overpayment or underpayment on L2. If there are not enough tokens in `SenderCCIP` during an overpayment scenario, the CCIP message may fail to be processed.

Recommendation

It is recommended to apply all of the following measures:

1. In `RestakingConnector`, calculate the difference in the `EigenAgent`'s token balance before and after interacting with EigenLayer.
2. Send only that difference via CCIP to L2.
3. On L2, rely on `any2EvmMessage.destTokenAmounts[]` for the actual amount being transferred, rather than `transferCommitmentsAmount[transferRoot]`.

Remediation

Patched

It has been patched as recommended.

#16 `TREASURE-2502-016` Handling Multiple Tokens Under a Single `transferRoot` Fails in `SenderCCIP`

ID	Summary	Severity
`TREASURE-2502-016`	`ReceiverCCIP` sends separate CCIP messages for each token in `transferTokensArray`, even though they share the same `transferRoot`. However, `SenderHooks.handleTransferToAgentOwner()` expects a single message containing all tokens for that `transferRoot`, leading to a failure when individual tokens arrive separately.	High

Description

`ReceiverCCIP._ccipReceive()` dispatches one CCIP message per token. Conversely, `SenderHooks.handleTransferToAgentOwner()` attempts to process all tokens associated with a `transferRoot` at once. When the first message arrives, it contains only one token, causing the combined token processing to break.

Impact

High

If each token is sent as a separate CCIP message, `SenderCCIP` will only hold a single token when the first message arrives on L2. However, `SenderHooks.handleTransferToAgentOwner()` attempts to transfer all tokens associated with the `transferRoot`, leading to a shortage of tokens and causing the message processing to fail.

Recommendation

It is recommended to apply one of the following two measures:

1. Modify `ReceiverCCIP` to send all tokens within a single CCIP message.
2. Remove the `transferRoot`-based grouping on L2 and rely on `any2EvmMessage.destTokenAmounts` in `SenderCCIP._afterCCIPReceiveMessage()` to transfer tokens individually.

Remediation

Patched

It has been patched as recommended.

#17 `TREASURE-2502-017` Minor Suggestions

ID	Summary	Severity
`TREASURE-2502-017`	The description includes multiple suggestions for preventing incorrect settings caused by operational mistakes, mitigating potential issues, and improving code maturity and readability.	Informational

Description

Operational Risk Mitigation / Sanity Checks

- `RestakingConnector` contract's owner should be a multisig wallet since it has privileges such as deleting items in `bridgeTokensL1toL2`.
- `ReceiverCCIP` should remove its dependency on `BaseSepolia` scripts. Instead of using `BaseSepolia.ChainSelector` in `_ccipReceive()`, `any2EvmMessage.sourceChainSelector` should be used.
- In L1 `ReceiverCCIP.sendMessagePayNative()`, users can send arbitrary text to L2 `SenderCCIP`. Currently, this does not pose a problem because `SenderCCIP` processes token transfer details related to a specific `transferRoot` (i.e., `transferCommitmentsAmount`, `transferCommitmentsTokenL2`, and `transferCommitmentsAgentOwner`) when `_afterCCIPReceiveMessage()` is executed. However, since it is impossible to predict at the time of sending the message from L2 how many tokens will actually be received (Issue 015), `_afterCCIPReceiveMessage()` could be modified to include the token amount sent from L1 within the text field. If such change is implemented, an attacker could exploit it by calling `ReceiverCCIP.sendMessagePayNative()` with arbitrary text to steal funds from L2 `SenderCCIP`. To mitigate this risk, it is recommended to enforce a validation in `ReceiverCCIP._buildCCIPMessage()` ensuring that `_receiver` cannot be `SenderCCIP` when `msg.sender` is not `address(this)`, or that the `decodeFunctionSelector(_text)` return value cannot be `ISenderHooks.handleTransferToAgentOwner.selector`.
- If the `msg.sender` on L2 is a contract, and the owner of the contract at the same address on L1 is different, the `EigenAgent` funds could be controlled by the L1 contract owner. Therefore, it is recommended to either add a warning message in the documentation or frontend, or restrict CCIP message transmission when `msg.sender` is a contract.

Missing / Confusing Events

- In `RestakingConnector.dispatchMessageToEigenAgent()`, if `functionSelector` does not match any expected condition, an event should be emitted for the ease of exception monitoring.
- `RestakingConnector._completeWithdrawalWithEigenAgent()` should emit an event similar to `SendingRewardsToAgentOwnerOnL1` from `_processClaimWithEigenAgent()` when withdrawing non-bridgeable assets to L1.
- Emit event in `BaseMessengerCCIP`'s `allowlistDestinationChain()`, `allowlistSourceChain()`, `allowlistSender()`.
- Emit event in `RestakingConnectorStorage`'s `setReceiverCCIP()`, `setAgentFactory()`, `setEigenlayerContracts()`, `setBridgeTokens()`, `clearBridgeTokens()`.
- Emit event in `SenderCCIP`'s `setSenderHooks()`.
- Emit event in `SenderHook`'s `setSenderCCIP()`.
- Emit event in `AgentFactory`'s `setRestakingConnector()`, `set6551Registry()`, `setEigenAgentOwner721()`.
- Emit event in `EigenAgentOwner721`'s `setAgentFactory()`, `addToWhitelistedCallers()`, `removeFromWhitelistedCallers()`.

Code Maturity Improvements

- The comment `InvalidSignature(string)` in `FunctionSelectorDecoder.decodeEigenAgentExecutionError()` should be updated to `SignatureInvalid(string)` to match the actual function name.

Impact

Informational

Recommendation

Consider applying the suggestions in the description above.

Remediation

Patched

Most items were patched as recommended.

Revision History

Version	Date	Description
1.0	Mar 23, 2025	Initial version

© 2025 ChainLight, Theori. All rights reserved

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.

© 2025 ChainLight, Theori. All rights reserved