Quantstamp

Sequence - Trail Contracts

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Multichain Transactions
Timeline	2025-10-15 through 2025-10-23
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• 0xsequence/trails-contracts #ca3916e
Auditors	• Tim Sigl Auditing Engineer • Leonardo Passos Senior Research Engineer • Andy Lin Senior Auditing Engineer

Documentation quality	Medium
Test quality	Medium
Total Findings	5 Fixed: 5
High severity findings	1 Fixed: 1
Medium severity findings	1 Fixed: 1
Low severity findings	3 Fixed: 3
Undetermined severity findings	0
Informational findings	0

Summary of Findings

The Trails protocol enables cross-chain transactions through three contracts operating within the Sequence v3 wallet context. The audit covered only the contracts within this audit's defined scope.

The `TrailsRouter` implements three main functionalities:

1. Batching DEX and bridge operations via `Multicall3`.
2. Injecting runtime token balances into calldata for protocols requiring exact amounts post-swap.
3. Sweeping tokens for fee collection or refunds.

The `TrailsRouterShim` contract is a wrapper for operations in the `TrailsRouter` which sets a success sentinel flag to conditionally collect fees only when the wrapped operation succeeds, or triggers a refund on failure. The `TrailsIntentEntrypoint` provides optional gasless token transfers for intent execution.

The protocol creates unique wallet addresses for each intent, executes operations through relayer-triggered batched calls, and handles two execution paths: cross-chain flows with bridging and same-chain with swaps and deposits into DeFi protocols such as Aave or Morpho. During every step of the process, the user should be able to recover tokens if operations fail. Necessary calls to fulfill the intent are constructed by the backend and signed by the user. The relayer can only execute operations that are signed by the user.

During the audit, five issues were identified, two of which were classified as high and medium severity. Finding SEQ-1 is concerned with unsigned parameters in the gasless token transfer function, which could allow malicious actors to drain user tokens. SEQ-2 addresses compatibility issues with common ERC-20 tokens caused by the lack of `safeTransfer()` usage. The remaining three finding are of low severity and further explained in the report below.

The repository demonstrates moderate test coverage, which should be further improved. Code documentation was enhanced during the audit and now includes sequence diagrams illustrating all intent flows.

Fix-Review Update 2025-11-03:
The client addressed all identified issues and implemented all recommendations. We are very satisfied with both the quality of the fixes and the clear communication throughout the fix process.

ID	DESCRIPTION	SEVERITY	STATUS
SEQ-1	Unsigned Fee Parameters Let Any Signature Holder Drain Funds	High	Fixed
SEQ-2	`transferFrom()` Usage Reverts on Non-Standard ERC-20 Tokens	Medium	Fixed
SEQ-3	Unenforced `allowFailure` Flag Could Cause Silent Execution Failures	Low	Fixed
SEQ-4	Mismatch Between `IMulticall3` Interface and Encoded Function	Low	Fixed
SEQ-5	Redundant ERC-20 Self-Approval in `delegatecall` -only Functions	Low	Fixed

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

The Sequence v3 wallet in which intent flows are executed and the off-chain backend responsible for constructing calldata were out of scope. Since these components were not reviewed, our security assessment necessarily relies on assumptions informed by discussions with the team and our high-level understanding of the Sequence v3 wallet's design.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

Repo: https://github.com/0xsequence/trails-contracts/(ca3916ed7d0a40bd085f0d301e5267a00f331f92) Files: src/*

Repo: `https://github.com/0xsequence/trails-contracts`

Included Paths: `src/`

Operational Considerations

- The relayer needs to trigger execution of intent operations to guarantee liveness of the protocol.
- The codebase is heavily coupled with the `wallet-contract-v3` project. We assume the triggering flow comes from the `Calls` contract that allows embedding multiple calls to the `TrailsRouterShim.handleSequenceDelegateCall()` in a single transaction. To correctly use the `TrailsRouter`, the calls need to be correctly constructed. For instance, a valid call sequence would be a main execution, a `validateOpHashAndSweep()`, and, `refundAndSweep()` as the fallback call.
- The backend is trusted to calculate correct `amountOffset` values for balance injection. Incorrect offsets will cause injection to revert.
- All intent wallet addresses are expected to be used only once. The intent wallet is the caller of the `TrailsRouterShim` and the `TrailsRouter`. Even for the same action sets, the intent wallet address will be unique.
- All pulled funds are consumed entirely so that the `TrailsRouter` does not hold any funds after executions.
- Callers of intent wallets can only executed predefined operations in the Merkle tree of actions.
- Cross-chain bridging operations are expected to succeed or fail and trigger the expected fallback flow.

Key Actors And Their Capabilities

Fee Collector

Responsibilities

- Receives protocol fees from successful intent executions via `validateOpHashAndSweep()` and `refundAndSweep()` calls.

Trust Assumptions

The fee collector address is provided by the backend off-chain. There are no on-chain restrictions on who can be set as the fee collector.

Relayer

Responsibilities

- Monitors user deposits to intent wallet addresses.
- Triggers execution of intent operations by submitting signed transactions.
- Provides liveness guarantees for the protocol by ensuring timely execution of pending intents.

Trust Assumptions

Relayers are trusted to execute operations signed by users. While they cannot execute unauthorized operations, they control execution timing and could delay or selectively process intents, potentially causing deadline expiration or unfavorable market conditions for users.

Intent Address (Sequence v3 Wallet)

Responsibilities

- Holds user assets deposited for intent execution.
- Executes delegatecall operations to `TrailsRouter` and `TrailsRouterShim`.
- A new wallet is created for every intent.

Trust Assumptions

The Sequence v3 wallet implementation is out of scope for this audit but critical to protocol security. The wallet must ensure that the user's EOA retains access to recover tokens in all scenarios, including failed operations or protocol malfunctions.

User (Signer)

Responsibilities

- Reviews and signs EIP-712 intents authorizing deposits and downstream protocol interactions.
- Provides ERC-20 allowances or ERC-2612 permits that fund intents.

Trust Assumptions

Users must validate calldata and fee parameters before signing; falsely signed calldata or excessive allowances can be abused.

Multicall3 (External Contract)

Responsibilities

- Executes batched calls on behalf of the intent wallet via `delegatecall`, preserving context.

Trust Assumptions

The protocol relies on the canonical `Multicall3` deployment at `0xcA11bde05977b3631167028862bE2a173976CA11`. Any calldata misconfiguration (for example, setting `allowFailure = true`) will cause inconsistencies.

Backend

Responsibilities

- Constructs intent calldata including DEX swaps, bridge calls, and protocol interactions.
- Calculates `amountOffset` values for balance injection in `injectAndCall()` operations.
- Encodes `Multicall3` calldata.
- Provides reasonable fee amounts relative to deposit amounts in `TrailsIntentEntrypoint` operations.

Trust Assumptions

The backend is trusted to construct valid and user-favorable calldata. Incorrect `amountOffset` calculations cause reverts. Malicious or compromised backend could encode operations that drain user assets. Users should validate the constructed calldata before signing it.

Findings

SEQ-1
Unsigned Fee Parameters Let Any Signature Holder Drain Funds

High Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `f36f3e8a1e00c7227db86f45ba2bff2df440d1ca` although the main fix is in commit `bfc4459`.
The client provided the following explanation:

```
Fixed in two parts:
1. CRITICAL FIX ( bfc4459 ): Added  feeAmount  and  feeCollector  to  TRAILS_INTENT_TYPEHASH .
     Fee parameters are now part of the EIP-712 typed data signature, preventing malicious
     relayers from injecting arbitrary fees or draining user funds.
2. ADDITIONAL VALIDATION ( 79ff75f ,  cad0793 ,  171b2c3 ): Added explicit validation to ensure
      permitAmount  exactly matches amount +  feeAmount  in  depositToIntentWithPermit() .
     This prevents DoS scenarios where permits are insufficient and provides clear error
     messages with the  PermitAmountMismatch()  custom error.
```

File(s) affected: `TrailsIntentEntrypoint.sol`

Description: Both `depositToIntentWithPermit()` and `depositToIntent()` functions in the `TrailsIntentEntrypoint` contract accept `feeAmount` and `feeCollector` as parameters without requiring user signature, creating two critical vulnerabilities that expose users to fund drainage.

The `feeAmount` and `feeCollector` parameters are not included in the signed intent defined by `TRAILS_INTENT_TYPEHASH`, allowing any relayer or even actors observing the mempool to specify arbitrary values when executing a user's intent. Since users typically grant maximum or excess token allowances for convenience, malicious relayers can frontrun legitimate transactions and drain the user's entire token balance through inflated fees. The contract will successfully transfer both the intended deposit amount and the malicious fee amount, as long as the user has sufficient balance and allowance.

Additionally, the `depositToIntentWithPermit()` function contains a validation gap where `permitAmount` is independent from `amount` and `feeAmount`. The contract does not enforce that `permitAmount == amount + feeAmount`, creating edge cases where permits may be insufficient or users unknowingly over-sign, exposing them to malicious fee extraction.

Exploit Scenario:

1. Alice signs an intent to deposit 100 USDC with a legitimate relayer advertising 1 USDC fee.
2. Alice grants maximum approval or permits a large amount (1000 USDC) for convenience.
3. Bob, a malicious relayer, observes Alice's transaction in the mempool.
4. Bob frontruns Alice's transaction by calling `depositToIntent()` with `feeAmount=900` and his own `feeCollector` address.
5. The contract successfully transfers 100 USDC to the intent address and 900 USDC to Bob's fee collector, as the signature only validates the deposit amount.
6. Alice loses 900 USDC despite only intending to pay 1 USDC in fees, with no recourse since the transaction was technically valid according to her signed intent.

Recommendation: Extend `TRAILS_INTENT_TYPEHASH` to include `feeAmount` and `feeCollector` parameters, ensuring users explicitly authorize the complete fee structure as part of their intent signature. Update the type hash definition to include these fields and modify the `_verifyAndMarkIntent()` function to verify these additional parameters match the signed intent.

Also, add validation in `depositToIntentWithPermit()` to ensure the permit covers both the deposit and fee amounts:

```
require(permitAmount == amount + feeAmount, "Insufficient permit amount");
```

These changes ensure users explicitly authorize both the deposit amount and the fee structure, preventing unauthorized fee extraction while maintaining the flexibility of the intent-based architecture.

SEQ-2
`transferFrom()` Usage Reverts on Non-Standard ERC-20 Tokens

Medium Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `57ef66a4f13f4f0b1e7e58b355150d71d05f70f8`.
The client provided the following explanation:

> Replaced all raw `transferFrom()` calls with OpenZeppelin's `SafeERC20.safeTransferFrom()` in `TrailsIntentEntrypoint`. This ensures compatibility with non-standard ERC-20 tokens (like USDT) that don't return boolean values. The SafeERC20 library properly handles both standard and non-standard token implementations.

File(s) affected: `TrailsIntentEntrypoint.sol`

Description: The contract uses raw `IERC20.transferFrom()` calls with manual boolean return value checking, which assumes full ERC-20 compliance and fails with popular non-standard tokens like USDT that do not return boolean values. When such tokens are used, the ABI decoder cannot parse the empty return payload, causing the `require` statements to revert and preventing users from depositing these assets.

The issue affects four transfer locations in the contract. In the `depositToIntentWithPermit()` function, the main transfer from user to `intentAddress` occurs at line 99, and an optional fee transfer to `feeCollector` happens at line 103. Similarly, the `depositToIntent()` function performs the main transfer at line 126 and the optional fee transfer at line 130. All four locations use the same vulnerable pattern: `require(IERC20(token).transferFrom(...));`

This implementation limits the protocol's compatibility with widely-used tokens and may exclude a significant portion of potential users who hold assets in non-standard ERC-20 implementations.

Recommendation: Import OpenZeppelin's `SafeERC20` library and replace all raw `transferFrom()` calls with `safeTransferFrom()`.

SEQ-3
Unenforced `allowFailure` Flag Could Cause Silent Execution Failures

Low | Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `815cd10540580768dd1e61b4883610313abad441`.
> The client provided the following explanation:
>
> > Added on-chain validation in `_validateRouterCall()` to decode the `Multicall3` calldata and verify that all `allowFailure` flags are set to `false`. This prevents silent failures where individual calls could fail without reverting the entire transaction. If any call has `allowFailure=true`, the transaction reverts with `AllowFailureMustBeFalse(uint256 callIndex)` error.

File(s) affected: `TrailsRouter.sol`

Description: `TrailsRouter` accepts arbitrary calldata for the `Multicall3.aggregate3Value()` function without validating the `Call3Value` struct contents. The backend currently encodes `allowFailure = false` to ensure subcall failures propagate, but nothing on-chain enforces this. A misconfigured backend could set `allowFailure = true` for critical operations (swaps, bridges), causing the multicall to succeed despite underlying reverts and leaving funds in an inconsistent state.

Recommendation: There are two possible fixes for this issue:

1. Construct the calldata to the `Multicall3` contract on-chain by encoding the `Call3Value[]` array inside `TrailsRouter` (e.g., accept a simpler struct and set `allowFailure = false` unconditionally).
2. Verify the passed calldata by the backend by decoding the calldata into `IMulticall3.Call3Value[]` and iterate through all entries to ensure `allowFailure` equals `false` for every call. Revert immediately if any entry has `allowFailure` set to `true`, preventing silent partial execution scenarios. This validation ensures that all operations must succeed or the entire transaction reverts, maintaining atomic execution semantics.

As a defense-in-depth measure, after the `delegatecall` returns, iterate through the `Result[]` array and verify that all entries have success equal to true. Revert on any failure to catch edge cases where validation might be bypassed.

SEQ-4
Mismatch Between `IMulticall3` Interface and Encoded Function

Low | Fixed

> **Update**
> Marked as "Fixed" by the client.
> The commit provided by the client (`fd495505f4de965461adc0a4ce66d674c7634e2d`) does not fully address the issue. However, the problem is correctly resolved in the final commit (`f36f3e8a1e00c7227db86f45ba2bff2df440d1ca`), which will be referenced here instead.
>
> The client provided the following explanation:

Updated the `IMulticall3` interface to include the `aggregate3Value()` function signature and the `Call3Value` struct definition. The interface now correctly matches the canonical `Multicall3` implementation deployed at `0xcA11bde05977b3631167028862bE2a173976CA11`.

File(s) affected: `TrailsRouter.sol`, `IMulticall3.sol`

Description: The `TrailsRouter`'s `execute()`, `pullAndExecute()` and `pullAmountAndExecute()` functions are intended to forward ETH through the `Multicall3` contract. However, the `IMulticall3` interface only declares the `aggregate3()` function, which never forwards `msg.value`. The client clarified that the backend actually encodes the `aggregate3Value()` function and therefore ETH forwarding works as expected. The mismatch between the interface and the encoded function could cause confusion for developers.

Recommendation: First, extend the `IMulticall3` interface with the `aggregate3Value()` function signature (and its matching struct).

Following that, similar to SEQ-3, there are two possible solutions:

1. Either construct the calldata to the `Multicall3` contract on-chain in the `TrailsRouter` contract via `abi.encodeWithSelector(IMulticall3.aggregate3Value.selector, calls)`.
2. Or extract and validate that the function selector in the first 4 bytes of data matches `IMulticall3.aggregate3Value.selector`. Reject any other selectors to prevent invocation of unintended functions.

SEQ-5
Redundant ERC-20 Self-Approval in `delegatecall`-only Functions

Low · Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `3121e54fee93703f235f4c06a41438fb139dbc73`.
The client provided the following explanation:

Removed redundant self-approvals in functions that are only called via `delegatecall` context. In delegatecal l context, address(this) refers to the wallet itself, making self-approvals unnecessary and wasteful. This change improves gas efficiency without affecting functionality.

File(s) affected: `TrailsRouter.sol`

Description: The `_ensureERC20Approval()` helper is called in `sweep()`, `refundAndSweep()`, and `validateOpHashAndSweep()` functions, but these functions are restricted by the `onlyDelegatecall` modifier. When executed via delegatecall through the wallet, `address(this)` resolves to the wallet contract while `_SELF` (the router implementation address) remains unchanged. This creates unnecessary approvals from the wallet to the router implementation contract.

Since `_transferERC20()` executes in the wallet's context, the wallet itself is the token holder and can transfer directly without needing any approval. The router implementation never calls `transferFrom()`, making these approvals completely redundant. The `sweep()` function at line 154 approves `_SELF` before every ERC20 sweep, while `refundAndSweep()` at line 169 pre-approves `_SELF` for the full balance. The `validateOpHashAndSweep()` function inherits the issue through its internal call to `sweep()`.

The main security concern is that wallets are intended for single-use. However, if a wallet is reused or if `_SELF` (the router implementation) is compromised or upgraded maliciously, these lingering approvals could allow unexpected token drainage.

Recommendation: Remove the `_ensureERC20Approval()` function and the calls from delegatecall-only functions.

Auditor Suggestions

S1 Replace String Reverts with Custom Errors

Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `8ef811b1ad4f04427eadd7798625fbe3b2aa91e9`.
At the time of reviewing the client's fixes the commit was no longer available, therefore the final commit will be referenced here: `f36f3e8a1e00c7227db86f45ba2bff2df440d1ca`.
The client provided the following explanation:

By implementing SafeERC20 in SEQ-2, the string-based require statements for transfer validation were replaced with OpenZeppelin's custom error handling. SafeERC20 uses custom errors internally, which are more gas-efficient than string-based reverts.

File(s) affected: `TrailsIntentEntrypoint.sol`

Description: The `TrailsIntentEntrypoint` contract uses string-based `require()` statements (e.g., `require(condition, "Error message")`), which consume more gas than custom errors and provide less structured debugging information. Custom errors introduced in Solidity 0.8.4 offer cheaper reverts and enable typed parameters for richer context. Note that Solidity 0.8.26 introduced `require(bool, Error)` syntax to use custom errors directly with `require()`.

Recommendation: Replace all string-based reverts with custom error declarations.

S2 Remove Unused Internal Helper Functions in `TrailsRouter`

Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `5745f14acdaedf389947353a57198dcf12780f92`.
At the time of reviewing the client's fixes the commit was no longer available, therefore the final commit will be referenced here: `f36f3e8a1e00c7227db86f45ba2bff2df440d1ca`.
The client provided the following explanation:

```
Removed three unused internal helper functions from TrailsRouter: _nativeBalance(), _erc20Balance(), and _erc20BalanceOf(). These functions were not being used anywhere in the codebase. Removing them improves code maintainability and provides minor gas savings during deployment.
```

File(s) affected: `TrailsRouter.sol`

Description: The `TrailsRouter` contract contains three internal helper functions that are defined but never used anywhere in the codebase. The `_nativeBalance()` function at line 306, `_erc20Balance(address _token)` at line 311, and `_erc20BalanceOf(address _token, address _account)` at line 316 were likely intended to provide balance checking utilities. However, the contract instead uses the more flexible `_getBalance(address token, address account)` and `_getSelfBalance(address token)` helper functions throughout its implementation.

Recommendation: Remove the three unused internal helper functions from `TrailsRouter`: `_nativeBalance()`, `_erc20Balance()`, and `_erc20BalanceOf()`.

S3 Functions `pullAndExecute()` and `pullAmountAndExecute()` Can Be Refactored to Eliminate Redundancy

Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `5745f14acdaedf389947353a57198dcf12780f92`.
At the time of reviewing the client's fixes the commit was no longer available, therefore the final commit will be referenced here: `f36f3e8a1e00c7227db86f45ba2bff2df440d1ca`.
The client provided the following explanation:

```
Refactored pullAndExecute() to call pullAmountAndExecute() after calculating the balance, eliminating code duplication. Now there is a single source of truth for the execution logic, making the codebase easier to maintain and less prone to bugs from inconsistent implementations.
```

File(s) affected: `TrailsRouter.sol`

Description: Both functions are too much alike. Redundancy can be removed for easier maintenance.

Recommendation: Have `pullAndExecute()` call `pullAmountAndExecute()` passing in the user's balance.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.

- **Informational** – The issue does not pose an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Files

Repo: `https://github.com/0xsequence/trails-contracts`

- `1ee...dfd ./src/TrailsIntentEntrypoint.sol`
- `694...583 ./src/TrailsRouter.sol`
- `84f...44c ./src/TrailsRouterShim.sol`
- `e61...857 ./src/guards/DelegatecallGuard.sol`
- `7a5...ae6 ./src/interfaces/IMulticall3.sol`
- `d0b...d41 ./src/interfaces/ITrailsIntentEntrypoint.sol`
- `535...7d8 ./src/interfaces/ITrailsRouter.sol`
- `af9...74e ./src/interfaces/ITrailsRouterShim.sol`
- `5f6...2f4 ./src/libraries/TrailsSentinelLib.sol`

Test Suite Results

Test data output was obtained with `forge test`. Deployment related tests were skipped. All the 8 test suites with 136 tests in total executed successfully.

```
Ran 56 tests for test/TrailsRouter.t.sol:TrailsRouterTest
[PASS] testDelegateCallWithETH() (gas: 429585)
[PASS] testExecute_WithFailingMulticall() (gas: 556795)
[PASS] testHandleSequenceDelegateCall_InjectAndCall() (gas: 85553)
[PASS] testHandleSequenceDelegateCall_RefundAndSweep() (gas: 99051)
[PASS] testHandleSequenceDelegateCall_Sweep() (gas: 58326)
[PASS] testHandleSequenceDelegateCall_ValidateOpHashAndSweep() (gas: 77706)
[PASS] testInjectAndCall_NoReplacementNeeded() (gas: 1396741)
[PASS] testInjectAndCall_WithReplacement() (gas: 1404981)
[PASS] testInjectAndCall_WithTokenZeroBalance() (gas: 822845)
[PASS] testInjectAndCall_WithZeroBalance() (gas: 15178)
[PASS] testInjectSweepAndCall() (gas: 1416396)
[PASS] testInjectSweepAndCall_WithETH_TargetCallFails() (gas: 65505)
[PASS] testInjectSweepAndCall_WithETH_ZeroBalance() (gas: 16491)
[PASS] testInjectSweepAndCall_WithToken_TargetCallFails() (gas: 1403016)
[PASS] testInjectSweepAndCall_WithToken_ZeroBalance() (gas: 823057)
[PASS] testInsufficientEthValidation() (gas: 28762)
[PASS] testNativeTransferFailure() (gas: 100963)
[PASS] testRefundAndSweep_FullRefund() (gas: 66129)
[PASS] testRefundAndSweep_PartialRefundERC20() (gas: 142736)
[PASS] testRefundAndSweep_ZeroRefundAmount() (gas: 64769)
[PASS] testRevertWhen_injectAndCall_InsufficientEth() (gas: 16052)
[PASS] testRevertWhen_injectAndCall_NoEthAvailable() (gas: 16529)
[PASS] testRevertWhen_injectSweepAndCall_InsufficientAllowance() (gas: 63667)
[PASS] testRevertWhen_injectSweepAndCall_NoEthSent() (gas: 20046)
[PASS] testSweepAndCallETH() (gas: 87338)
[PASS] testValidateOpHashAndSweep_WithoutSentinel() (gas: 19895)
```

```
[PASS] test_Execute_FromContract_ShouldPreserveContractAsSender() (gas: 30567)
[PASS] test_Execute_FromEOA_ShouldPreserveEOAAsSender() (gas: 38321)
[PASS] test_Execute_WithMultipleCalls() (gas: 40747)
[PASS] test_Multicall3Address_IsCorrect() (gas: 7675)
[PASS] test_ReceiveETH_ShouldAcceptETH() (gas: 19961)
[PASS] test_RevertWhen_pullAmountAndExecute_InsufficientAllowance() (gas: 34817)
[PASS] test_RevertWhen_pullAndExecute_InsufficientAllowance() (gas: 40791)
[PASS] test_amount_offset_out_of_bounds() (gas: 498044)
[PASS] test_direct_sweep_reverts_not_delegatecall() (gas: 14652)
[PASS] test_handleSequenceDelegateCall_dispatches_to_sweep_native() (gas: 61021)
[PASS] test_handleSequenceDelegateCall_invalid_selector_reverts() (gas: 15962)
[PASS] test_native_transfer_failed() (gas: 104482)
[PASS] test_no_tokens_to_pull() (gas: 820715)
[PASS] test_no_tokens_to_sweep() (gas: 1297125)
[PASS] test_placeholder_mismatch() (gas: 499137)
[PASS] test_pullAmountAndExecute_WithETH_InsufficientEthSent() (gas: 30776)
[PASS] test_pullAmountAndExecute_WithETH_ShouldTransferAndExecute() (gas: 46296)
[PASS] test_pullAmountAndExecute_WithToken_ShouldTransferAndExecute() (gas: 95681)
[PASS] test_pullAmountAndExecute_WithValidToken_ShouldTransferAndExecute() (gas: 95336)
[PASS] test_pullAndExecute_WithETH_NoEthSent() (gas: 23405)
[PASS] test_pullAndExecute_WithETH_ShouldTransferAndExecute() (gas: 47067)
[PASS] test_pullAndExecute_WithFailingMulticall() (gas: 616587)
[PASS] test_pullAndExecute_WithValidToken_ShouldTransferFullBalanceAndExecute() (gas: 95874)
[PASS] test_refundAndSweep_erc20_partialRefund() (gas: 169387)
[PASS] test_refundAndSweep_native_partialRefund() (gas: 104649)
[PASS] test_success_sentinel_not_set() (gas: 19228)
[PASS] test_sweep_erc20Token() (gas: 125605)
[PASS] test_sweep_nativeToken() (gas: 57555)
[PASS] test_validateOpHashAndSweep_native_success() (gas: 80829)
[PASS] test_validateOpHashAndSweep_native_success_tstore() (gas: 176877)
Suite result: ok. 56 passed; 0 failed; 0 skipped; finished in 235.26ms (77.22ms CPU time)

Ran 18 tests for test/TrailsRouterShim.t.sol:TrailsRouterShimTest
[PASS] testConstructorValidation() (gas: 72582)
[PASS] testForwardToRouterReturnValue() (gas: 696707)
[PASS] testRouterAddressImmutable() (gas: 1841368)
[PASS] test_constructor_revert_zeroRouter() (gas: 72214)
[PASS] test_delegatecall_forwards_and_sets_sentinel_sstore_inactive() (gas: 2125960)
[PASS] test_delegatecall_forwards_and_sets_sentinel_tstore_active() (gas: 42781)
[PASS] test_delegatecall_router_revert_bubbles_as_RouterCallFailed() (gas: 90623)
[PASS] test_delegatecall_sets_sentinel_with_sstore_when_no_tstore() (gas: 2114325)
[PASS] test_delegatecall_sets_sentinel_with_tstore_when_supported() (gas: 30985)
[PASS] test_direct_handleSequenceDelegateCall_reverts_not_delegatecall() (gas: 13680)
[PASS] test_forwardToRouter_return_data_handling() (gas: 710061)
[PASS] test_forwardToRouter_revert_with_custom_error() (gas: 639238)
[PASS] test_handleSequenceDelegateCall_empty_calldata() (gas: 23566)
[PASS] test_handleSequenceDelegateCall_large_calldata() (gas: 8237184)
[PASS] test_handleSequenceDelegateCall_max_call_value() (gas: 30621)
[PASS] test_handleSequenceDelegateCall_with_eth_value() (gas: 31044)
[PASS] test_handleSequenceDelegateCall_zero_call_value() (gas: 24580)
[PASS] test_sentinel_setting_with_different_op_hashes() (gas: 45086)
Suite result: ok. 18 passed; 0 failed; 0 skipped; finished in 325.61ms (118.31ms CPU time)

Ran 8 tests for test/guards/DelegatecallGuard.t.sol:DelegatecallGuardTest
[PASS] testOnlyDelegatecallInternalFunction() (gas: 158000)
[PASS] testSelfImmutableVariable() (gas: 153916)
[PASS] test_delegatecall_context_succeeds() (gas: 20867)
[PASS] test_delegatecall_nested_context() (gas: 415303)
[PASS] test_direct_call_reverts_NotDelegateCall() (gas: 10312)
[PASS] test_multiple_delegatecall_guards() (gas: 392466)
[PASS] test_onlyDelegatecall_modifier_usage() (gas: 149502)
[PASS] test_self_address_immutable() (gas: 3987)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 325.94ms (82.18ms CPU time)

Ran 3 tests for test/script/TrailsRouterShim.s.t.sol:TrailsRouterShimDeploymentTest
[PASS] test_DeployRouterShim_SameAddress() (gas: 9329412)
[PASS] test_DeployRouterShim_Success() (gas: 6000393)
[PASS] test_DeployedContract_HasCorrectConfiguration() (gas: 6001078)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 325.60ms (362.48ms CPU time)

Ran 3 tests for test/script/TrailsRouter.s.t.sol:TrailsRouterDeploymentTest
[PASS] test_DeployTrailsRouter_SameAddress() (gas: 2149219)
```

```
[PASS] test_DeployTrailsRouter_Success() (gas: 2131122)
[PASS] test_DeployedRouter_HasCorrectConfiguration() (gas: 2130589)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 325.88ms (361.64ms CPU time)

Ran 3 tests for test/script/TrailsIntentEntrypoint.s.t.sol:TrailsIntentEntrypointDeploymentTest
[PASS] test_DeployIntentEntrypoint_SameAddress() (gas: 1415930)
[PASS] test_DeployIntentEntrypoint_Success() (gas: 1399706)
[PASS] test_DeployedIntentEntrypoint_HasCorrectConfiguration() (gas: 1410863)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 326.11ms (389.00ms CPU time)

Ran 32 tests for test/TrailsIntentEntrypoint.t.sol:TrailsIntentEntrypointTest
[PASS] testAssemblyCodeExecution() (gas: 143306)
[PASS] testConstructor() (gas: 4003)
[PASS] testConstructorAndDomainSeparator() (gas: 12408)
[PASS] testDepositToIntentAlreadyUsed() (gas: 172299)
[PASS] testDepositToIntentCannotReuseDigest() (gas: 152605)
[PASS] testDepositToIntentExpiredDeadline() (gas: 73985)
[PASS] testDepositToIntentReentrancyProtection() (gas: 172276)
[PASS] testDepositToIntentRequiresNonZeroAmount() (gas: 40158)
[PASS] testDepositToIntentRequiresValidToken() (gas: 44321)
[PASS] testDepositToIntentTransferFromFails() (gas: 100472)
[PASS] testDepositToIntentWithPermitAlreadyUsed() (gas: 210751)
[PASS] testDepositToIntentWithPermitExpiredDeadline() (gas: 57396)
[PASS] testDepositToIntentWithPermitReentrancyProtection() (gas: 193713)
[PASS] testDepositToIntentWithPermitRequiresNonZeroAmount() (gas: 54592)
[PASS] testDepositToIntentWithPermitRequiresPermitAmount() (gas: 167689)
[PASS] testDepositToIntentWithPermitRequiresValidToken() (gas: 56052)
[PASS] testDepositToIntentWithPermitTransferFromFails() (gas: 166815)
[PASS] testDepositToIntentWithPermitWrongSigner() (gas: 63193)
[PASS] testDepositToIntentWithoutPermit_RequiresIntentAddress() (gas: 45761)
[PASS] testDepositToIntentWrongSigner() (gas: 77204)
[PASS] testExactApprovalFlow() (gas: 275601)
[PASS] testExecuteIntentWithFee() (gas: 235537)
[PASS] testExecuteIntentWithPermit() (gas: 192654)
[PASS] testExecuteIntentWithPermitExpired() (gas: 57225)
[PASS] testExecuteIntentWithPermitInvalidSignature() (gas: 60294)
[PASS] testFeeCollectorReceivesFees() (gas: 312837)
[PASS] testInfiniteApprovalFlow() (gas: 268989)
[PASS] testIntentTypehashConstant() (gas: 6820)
[PASS] testInvalidNonceReverts() (gas: 71537)
[PASS] testNonceIncrementsOnDeposit() (gas: 144154)
[PASS] testUsedIntentsMapping() (gas: 145729)
[PASS] testVersionConstant() (gas: 13516)
Suite result: ok. 32 passed; 0 failed; 0 skipped; finished in 326.50ms (598.73ms CPU time)

Ran 13 tests for test/libraries/TrailsSentinelLib.t.sol:TrailsSentinelLibTest
[PASS] test_Constants_DoNotChange() (gas: 2067)
[PASS] test_SentinelNamespace_Computation() (gas: 638)
[PASS] test_SentinelNamespace_Constant() (gas: 868)
[PASS] test_SuccessSlot_AssemblyCorrectness() (gas: 2208)
[PASS] test_SuccessSlot_Computation() (gas: 1854)
[PASS] test_SuccessSlot_Deterministic() (gas: 1195)
[PASS] test_SuccessSlot_DifferentOpHashes() (gas: 1295)
[PASS] test_SuccessSlot_MaxOpHash() (gas: 1555)
[PASS] test_SuccessSlot_UsesCorrectNamespace() (gas: 1531)
[PASS] test_SuccessSlot_VariousOpHashes() (gas: 23670)
[PASS] test_SuccessSlot_ZeroOpHash() (gas: 1621)
[PASS] test_SuccessValue_Constant() (gas: 932)
[PASS] test_SuccessValue_IsOne() (gas: 1619)
Suite result: ok. 13 passed; 0 failed; 0 skipped; finished in 326.39ms (349.47ms CPU time)

Ran 8 test suites in 370.31ms (2.52s CPU time): 136 tests passed, 0 failed, 0 skipped (136 total tests)
```

Code Coverage

The code coverage output was obtained by running `forge coverage --ir-minimum --optimizer-runs 300 --no-match-coverage "test/|script/"`.

The repository shows moderate branch coverage of 83%, meaning that while many paths are tested, some branches remain untested. Test coverage should be improved to ensure more complete validation of conditional logic. In particular, the following contracts could benefit from additional testing: `TrailsIntentEntrypoint` and `TrailsRouter`.

File	% Lines	% Statements	% Branches	% Funcs
src/TrailsIntentEntrypoint.sol	65.96% (31/47)	70.37% (38/54)	70.59% (12/17)	100.00% (4/4)
src/TrailsRouter.sol	91.97% (126/137)	91.56% (141/154)	86.67% (39/45)	85.00% (17/20)
src/TrailsRouterShim.sol	85.71% (12/14)	87.50% (14/16)	100.00% (2/2)	100.00% (3/3)
src/guards/DelegatecallGuard.sol	75.00% (3/4)	66.67% (2/3)	100.00% (1/1)	100.00% (2/2)
src/libraries/TrailsSentinelLib.sol	40.00% (2/5)	25.00% (1/4)	100.00% (0/0)	100.00% (1/1)
Total	84.06% (174/207)	84.85% (196/231)	83.08% (54/65)	90.00% (27/30)

Changelog

- 2025-10-23 - Initial report
- 2025-11-03 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.