

Sequence - Trails V1.5

Executive Summary

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

Type	Multichain Transactions	Documentation quality	Medium
Timeline	2026-01-26 through 2026-01-30	Test quality	High
Language	Solidity	Total Findings	3 Fixed: 1 Acknowledged: 2
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review	High severity findings ⓘ	0
Specification	wallet-v3 doc ↗	Medium severity findings ⓘ	0
Source Code	• Oxsequence/trails-contracts ↗ #ed91b5b ↗	Low severity findings ⓘ	0
Auditors	• Andy Lin Senior Auditing Engineer • Jennifer Wu Auditing Engineer • Yamen Merhi Auditing Engineer	Undetermined severity findings ⓘ	0
		Informational findings ⓘ	3 Fixed: 1 Acknowledged: 2

Summary of Findings

We reviewed the Trails Contracts project, an intent-based transaction execution framework integrated with Sequence Wallet that enables complex transaction workflows with runtime parameter resolution. The system is composed of core modules: `MalleableSapient`, which provides flexible signature validation via static and repeatable sections; `HydrateProxy`, which supports runtime value injection and batched execution; `RequireUtils`, which enforces precondition validation; and `Sweepable`, which provides fund recovery after execution. A standalone `TrailsValidator` contract provides `msg.sender`-based precondition checks.

Overall, the codebase reflects strong engineering discipline, featuring a stateless architecture, comprehensive test coverage (including unit tests, integration tests exercising real Sequence Wallet flows, and fuzz testing), as well as use of `SafeERC20` for token transfers and gas-efficient custom errors across core modules. While the project demonstrates a generally sound architectural approach and reasonable security considerations for a stateless execution model, its security posture heavily depends on the correctness of the `wallet-contracts-v3` integration and on SDK/backend components constructing payloads as intended. Our review identified some issues and improvement opportunities, and we recommend addressing all findings and suggestions to strengthen the overall security and robustness of the intent execution flow.

Fix Review Update: The team has fixed or acknowledged all of the findings and suggestions in this report.

ID	DESCRIPTION	SEVERITY	STATUS
SEQ-1	Truncated Hydration Call Index May Mutate the Wrong Call	• Informational ⓘ	Acknowledged
SEQ-2	Malformed Hydration Payload May Cause Unintended Call Mutation	• Informational ⓘ	Acknowledged
SEQ-3	RequireUtils and TrailsValidator ERC721 Approval Checks Do Not Validate Token Ownership	• Informational ⓘ	Fixed

Assessment Breakdown

i 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.

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>

Included Paths: `src/`

Extensions: `.sol`

Files Excluded

Files: `lib/*`

Operational Considerations

- The codebase is tightly coupled to Sequence `wallet-contracts-v3`. We assume the triggering flow originates from the wallet's `Calls` module and invokes `HydrateProxy` via the delegated-extension entrypoint `HydrateProxy.handleSequenceDelegateCall()`. Correct call bundling (including success/fallback calls using `onlyFallback`) is required for the intended execution semantics.
- The SDK/API is trusted to construct `packedPayload`, `hydratePayload`, and the `MalleableSapient` descriptor (static/repeat sections). Malformed or truncated hydration payloads and intentional misconfiguration are considered out of scope per client discussion. Additionally, these configurations must account for the low-level implementation details of the Trails contract, as this is required to ensure that fields and payloads are mostly static and not malleable, except where necessary.
- Intent wallet addresses are expected to be single-use per Trails transaction; even for the same action set, each Trails execution uses a new intent address.
- `MalleableSapient` repeat sections are intentionally "malleable but must match" (e.g., enforce `permit(amount)` matches `transferFrom(amount)`), and do not commit to a specific value.
- The relayer must trigger execution of intent operations to provide protocol liveness.

- `sweep()` and `hydrateExecuteAndSweep()` transfer ERC20 tokens and native ETH to a caller-specified recipient. These operations revert the entire transaction if any transfer fails (e.g., malformed token addresses in `tokensToSweep`, tokens that revert due to pausability/blocklists, ERC777/ERC1363 receiver hooks that revert, or ETH recipients that cannot receive ETH). This is by design; callers must ensure valid token contracts and compatible recipients. Failed sweeps do not result in fund loss; funds remain in the contract and can be swept in a later transaction.
- **Cross-chain replay with `noChainId`**: When `payload.noChainId` is true, the signature is chain-agnostic (uses `chainId=0` in both the EIP-712 domain and `MalleableSapient.imageHash`). This allows the same signed payload to be replayed across chains if nonce/space align. However, this risk is acceptable for Trails: intent addresses are single-use, temporary execution contexts that are not expected to hold funds beyond the transaction lifecycle. Any cross-chain replay would simply re-execute the same intent logic on a different chain (e.g., permit+transfer on Polygon after execution on Ethereum), which fails if the source chain's nonce was already consumed or if token balances differ. The `noChainId` feature is intentional for legitimate cross-chain use cases (e.g., atomic swaps, unified multi-chain intents), and the SDK/UI should warn users when this mode is enabled.

Key Actors And Their Capabilities

The Trails Contracts are **stateless and ownerless** with no privileged admin roles. All functions are permissionless. Security relies on Sequence Wallet's signature validation and the trusted SDK/API constructing safe payloads.

1. Sequence Wallet Signer

- Signs transaction payloads off-chain that define static vs malleable sections via `MalleableSapient`
- Commits to specific parameters: static sections (immutable), repeat sections (must match), malleable sections (modifiable by relayer)
- Controls nonce consumption to prevent replay
- Cannot directly call `TrailsUtils`—must go through Sequence Wallet
- Example: signs `permit(amount) + transferFrom(amount)` where `amount` is malleable but must match

2. Relayer

- Submits pre-signed transactions via `Calls.execute()` and pays gas fees
- Can modify malleable sections without invalidating signature
- Can inject their address (`msg.sender`) into payloads via `HYDRATE_DATA_MESSAGE_SENDER`
- Cannot modify static sections or execute without valid signature

3. SDK/API (Trusted Off-chain)

- Constructs `packedPayload`, `hydratePayload`, and `MalleableSapient` descriptors
- Defines which bytes are static (committed) vs malleable (modifiable)
- Calculates byte offsets for hydration commands
- Currently hardcodes configurations—users cannot customize (per client discussion Q6)
- Trusted to mark security-critical data as static and avoid `HYDRATE_TYPE_TO` on delegatecalls

4. Sequence Wallet Contract

- Validates signatures via `MalleableSapient.recoverSapientSignature()` for Sapient signers (flag 0x9)
- Computes `imageHash` from static/repeat sections and validates against wallet config
- Consumes nonces and enforces threshold requirements
- Delegatecalls to `HydrateProxy.handleSequenceDelegateCall()` for execution
- Privileged functions (`onlySelf`): `BaseAuth.updateImageHash()`, `BaseAuth.setStaticSignature()`, `Calls.selfExecute()`

5. HydrateProxy Caller (Permissionless)

- Anyone can call `HydrateProxy.hydrateExecute()`, `sweep()`, or `RequireUtils` functions
- Hydration capabilities: inject addresses, balances (`balanceOf()`), allowances (`allowance()`), replace call targets/values
- Address sources: `SELF`, `MESSAGE_SENDER`, `TRANSACTION_ORIGIN`, `ANY_ADDRESS`
- Via wallet delegatecall: `handleSequenceDelegateCall()` validates context (`address(this) != SELF`)
- Delegatecall blocked when called directly (`HydrateProxy._hydrateExecute()` line 135-137)
- `sweep()` is permissionless—anyone can sweep funds from the contract

6. External Target Contracts (Untrusted)

- Receive calls from HydrateProxy during batch execution
- Can revert, return data, or attempt reentrancy (mitigated by stateless design)
- Cannot modify `TrailsUtils` state or steal funds (should be empty after sweep)
- Subject to error handling: `IGNORE_ERROR`, `REVERT_ON_ERROR`, `ABORT_ON_ERROR`

Findings

SEQ-1

Truncated Hydration Call Index May Mutate the Wrong Call

• **Informational** ⓘ

Acknowledged

ⓘ Update

The client confirmed this is a deliberate gas optimization trade-off. Real-world batches are expected to remain well below 255 calls. A new deployment with `uint16` indexing would be created if larger batches become necessary.

File(s) affected: src/modules/HydrateProxy.sol

Description: The packed call format in wallet-v3's `Payload.fromPackedCalls()` supports batches containing up to 65,535 calls, with `numCalls` encoded using either 8 or 16 bits. The hydration stream, however, encodes the target call index (`tindex`) using a single byte (`uint8`) in `HydrateProxy._firstHydrateCall()` and `HydrateProxy._hydrate()`, and does not enforce a maximum batch size or validate index bounds when hydration data is present.

For batches exceeding 255 calls, hydration directives cannot address calls beyond index 255. If the off-chain encoding layer truncates a target index to fit in a `uint8` (e.g., index 260 becomes 4), the hydration will be silently applied to the wrong call. Since hydration can mutate call recipients (`Payload.Call.to`), calldata (`Payload.Call.data`), and ETH value (`Payload.Call.value`) prior to execution, this would cause execution of calls with unintended parameters without reverting.

Even if the encoding layer correctly avoids truncation, calls at indices 256+ that require hydration would execute un-hydrated with their original packed payload values, also without reverting.

`HydrateProxy._hydrateExecute()` runs inside an `unchecked` block, and no bounds validation exists to detect or prevent either failure mode.

Recommendation: Either widen the hydration target index (`tindex`) to `uint16` so it can correctly address all calls in batches that support up to 65,535 entries, or add an explicit check at the start of `HydrateProxy._hydrateExecute()` that reverts when `hydratePayload.length > 0 && numCalls > 255`, making the limitation fail-safe rather than fail-silent.

SEQ-2

Malformed Hydration Payload May Cause Unintended Call Mutation

• **Informational** ⓘ

Acknowledged

ⓘ Update

The client confirmed that well-formed encoding is the SDK/API's responsibility, and intentional misconfiguration is out of scope.

File(s) affected: src/modules/HydrateProxy.sol

Description: `HydrateProxy._hydrate()` reads the hydration stream using wallet-v3's `LibBytes` helpers that do not bounds-check. If `hydratePayload` is truncated, reads will continue past the payload into ABI-padded zeroes or subsequent tail data, and the contract will treat those bytes as valid hydration inputs. This can silently mutate `Payload.Call.to`, `Payload.Call.value`, or `Payload.Call.data` fields (e.g., `HYDRATE_DATA_ANY_ADDRESS` resolving to `address(0)`), leading to unintended recipients or values instead of a clean revert.

Under the intended usage model, `hydratePayload` is constructed by trusted off-chain tooling (the SDK/API) and committed as a static section in the `MalleableSapient` signature. This means a relayer cannot tamper with or truncate the hydration payload without invalidating the user's signature. The only trigger for this issue is a bug in the SDK that produces a malformed payload which the user then signs.

Recommendation: Document hydration parsing as a trusted-input limitation. If the hydration payload is malformed or truncated, hydration should not continue in a way that treats missing bytes as valid values and mutates calls to unintended recipients or values. In such cases, hydration should fail closed and rely on a recovery mechanism, such as resubmission with a corrected payload or an owner-authorized recovery path, so the wallet can continue to execute future transactions safely.

SEQ-3

RequireUtils and TrailsValidator ERC721 Approval Checks Do Not Validate Token Ownership

• **Informational** ⓘ

Fixed

ⓘ Update

Marked as "Fixed" by the client.

Addressed in: 1264d8e31e9dc4eaf8b24e5ee4b31c889d7a33c3 , 0609ff8c6e415f2432952df55893750ba7a12a58 .

The client provided the following explanation:

Added ERC721 ownership functions. Also added owner+approved and balance+approved functions to support this common pattern.

We verified that the team introduced new functions, `requireERC721OwnerApproval()` and `requireERC721OwnerApprovalSelf()`, which include the "owner" check as pointed out in this issue. We want to highlight that the old functions, `requireERC721Approval()` and `requireERC721ApprovalSelf()`, were intentionally kept so users can have flexibility and control over what to validate. The old functions do not include the "owner" check.

File(s) affected: src/modules/RequireUtils.sol , src/TrailsValidator.sol

Description: Both contracts contain ERC721 approval validation functions that do not verify actual token ownership before checking approvals. In `RequireUtils._requireERC721Approval()`, the function accepts an `owner` parameter but trusts it without verification:

```
function _requireERC721Approval(address token, address owner, address spender, uint256 tokenId) private view {
    address approved = IERC721(token).getApproved(tokenId);
    if (approved != spender && !IERC721(token).isApprovedForAll(owner, spender)) {
        revert ERC721NotApproved(token, tokenId, owner, spender);
    }
}
```

In `TrailsValidator.requireERC721Approval()`, the function uses `msg.sender` in the `isApprovedForAll` check without verifying that `msg.sender` actually owns the token. Both functions check two conditions: (1) is `spender` specifically approved for `tokenId`, or (2) is `spender` an operator approved for all of the presumed owner's tokens. If the presumed owner has `isApprovedForAll(presumedOwner, spender) == true` but does not actually own the `tokenId`, the validation passes incorrectly. The subsequent transfer operation will then fail with a confusing error. For example, if NFT tokenId=42 is owned by Alice, but Bob has called `setApprovalForAll(spender, true)` for his own NFTs, calling `requireERC721Approval(nft, Bob, spender, 42)` will pass (Bob approved spender) but `nft.transferFrom(Bob, recipient, 42)` will fail (Bob doesn't own tokenId=42). This is not a security vulnerability as no funds can be stolen—the batch will fail at the actual transfer. However, it degrades user experience by allowing a precondition to pass when it should fail early with a clear error message.

Recommendation: Consider adding ownership verification, or document this limitation clearly. For `RequireUtils`:

```
function _requireERC721Approval(address token, address owner, address spender, uint256 tokenId) private view {
    // Verify ownership first
    address actualOwner = IERC721(token).ownerOf(tokenId);
    if (actualOwner != owner) {
        revert ERC721WrongOwner(token, tokenId, owner, actualOwner);
    }

    address approved = IERC721(token).getApproved(tokenId);
    if (approved != spender && !IERC721(token).isApprovedForAll(owner, spender)) {
        revert ERC721NotApproved(token, tokenId, owner, spender);
    }
}
```

Alternatively, if the additional external call is undesirable for gas optimization, add documentation warning users to ensure correct ownership context.

Auditor Suggestions

S1 Missing Input Validation for `recoverSapientSignature`

Acknowledged

i Update

Marked as "Acknowledged" by the client.

The client provided the following explanation:

These are deliberate optimisations.

As per SEQ-1. Real world batches will use far fewer than 128 calls.

As per SEQ-2. Intentionally configurations are out of scope.

File(s) affected: `src/modules/MalleableSapient.sol`

Description: `MalleableSapient.recoverSapientSignature()` parses section descriptors from `signature` and uses the decoded values to index into `Payload.Call.data`. The function relies on several implicit assumptions about the encoded input that are not explicitly validated:

- Call index range limited by encoding design:** `tindex` uses its most significant bit as the repeat/static flag and masks the remaining 7 bits (`(tindex & 0x7F)`), limiting static and repeat-source sections to call indices 0--127. Meanwhile, `wallet-v3`'s `Payload.fromPackedCalls()` supports up to 65,535 calls. Calls at index 128+ cannot have their data committed as static sections and are implicitly treated as fully malleable.
- Asymmetric index handling:** `tindex` is masked to 7 bits (max 127), while `tindex2` (the repeat-target index) is read as a full `uint8` (max 255) without masking. This means a repeat section's source and target have different addressable ranges. Both indices are bounds-checked by Solidity's calldata array access, so out-of-range values revert with `Panic(0x32)` rather than causing silent misbehavior.
- No signature length validation per iteration:** The `while (rindex < signature.length)` loop guard in `MalleableSapient.recoverSapientSignature()` does not ensure sufficient bytes remain for a complete section descriptor (5 bytes

for static, 8 bytes for repeat). `wallet-v3`'s `LibBytes.readUInt8()` and `LibBytes.readUInt16()` perform no bounds checks (documented: "do not check if the input index is within the bounds of the data array"). If a truncated `signature` is provided, partial reads return zero-padded values, producing an incorrect `imageHash`. However, this incorrect hash will fail wallet signature validation, so the failure mode is transaction rejection rather than unintended execution.

Recommendation: Refactor the section descriptor encoding in `MalleableSapient.recoverSapientSignature()` to separate call indexing from control flags and align index width with the maximum supported payload size:

1. **(Sub-issue 1)** Widen the call index representation so `tindex` supports `uint16` when payloads may contain more than 255 calls, matching the payload's call indexing semantics. Move the repeat/static flag out of the index field by introducing a dedicated flag byte (or bitfield), instead of overloading the most significant bit of `tindex`.
2. **(Sub-issue 2)** Apply consistent decoding for all call indices (`tindex` and `tindex2`).
3. **(Sub-issue 3)** Validate signature length during parsing, reverting if insufficient bytes remain to decode a complete section descriptor.

This prevents index truncation, avoids ambiguous decoding, and ensures malformed signatures fail predictably.

S2 RequireUtils Self Checks Can Be Misapplied Under delegatecall Context Acknowledged

i Update

Marked as "Acknowledged" by the client.

The client provided the following explanation:

This is a deliberate optimisation to reduce both gas costs and complexity.

File(s) affected: `src/modules/RequireUtils.sol`

Description: `RequireUtils` provides multiple `*Self(...)` helpers (e.g., `requireMinBalanceSelf`, `requireMinERC20BalanceSelf`, `requireMinERC20AllowanceSelf` ...) that interpret "Self" as `msg.sender`, which is the Sequence Wallet instance. In the Trails design, execution frequently occurs through nested `delegatecall` (e.g., via `HydrateProxy.handleSequenceDelegateCall`), and `msg.sender` in a delegated context may differ from what is expected (e.g., sequence wallet). While this behavior is correct per EVM semantics, it is an operational footgun: a misconfigured bundle could unintentionally execute `RequireUtils` via `delegatecall` and end up validating balances/allowances/approvals for the wrong actor, resulting in false positives/negatives and unexpected liveness failures.

Adding a lightweight guard to each `*Self(...)` function to ensure it is not executed in a `delegatecall` context by validating `address(this)` equals an immutable `SELF` set at deployment.

Recommendation: If more security is deemed around such scenario, consider implementing the suggested mitigation. Otherwise, ignore the recommendation to have flexible implementation.

S3

README delegatecall surface statement is misleading relative to actual execution behavior Fixed

✓ Update

Marked as "Fixed" by the client.

Addressed in: [56753b8e404613cb8c58b27b98965277abc82477](#).

The client provided the following explanation:

README updated.

File(s) affected: `README.md`, `src/modules/HydrateProxy.sol`

Description: The README claims that `delegatecall` via the intent address is only able to access `hydrateExecute`. However, `HydrateProxy.handleSequenceDelegateCall(...)` forwards arbitrary data to `SELF.delegatecall(data)` without selector gating. In practice, any function exposed on the deployed `TrailsUtils` address is reachable when properly encoded into data (e.g., `hydrateExecuteAndSweep`, and any `RequireUtils` external function).

Recommendation: Consider updating the README to accurately describe the reachable selector surface through `handleSequenceDelegateCall`.

S4 General Suggestions

Mitigated

✓ Update

Marked as "Fixed" by the client.

Addressed in: fe405fae05633a691829373227637ef9a5d9a0ba , ab2a091638018871e3156a368009e143f5681b7a , eab05323f795de8339921c1c6ce47b9e7fd5d39e .

The client provided the following explanation:

Removed unchecked blocks, unused library and deprecated contract.

We verified that the team has removed `TrailsValidator`, dropped the unchecked block from `HydrateProxy`, and removed the import and usage of `CalldataDecode` from `HydrateProxy`. Slight nitpicks are that there are still some unchecked usages across the codebase (e.g., `MalleableSapient.recoverSapientSignature()`), and for better maintenance, the team might want to move the `CalldataDecode` contract to the test folder instead.

File(s) affected: `src/modules/HydrateProxy.sol`, `src/modules/MalleableSapient.sol`, `src/utils/CalldataDecode.sol`, `src/TrailsValidator.sol`

Description: The codebase contains some opportunities for cleanup that would improve maintainability:

1. Unnecessary unchecked blocks: Loop increments in `HydrateProxy._hydrateExecute()` and `MalleableSapient.recoverSapientSignature()` are wrapped in `unchecked`, but modern Solidity (0.8.22+) can automatically optimize provably safe operations like array-bounded loop counters. These blocks reduce readability with minimal gas savings.
2. Unused CalldataDecode dependency: `HydrateProxy.sol` imports and declares using `CalldataDecode` for `bytes`; , but no functions from this library are called in production code. The client confirmed this is a leftover from earlier development.
3. Deprecated TrailsValidator contract: `TrailsValidator.sol` is no longer used—its functionality has been replaced by `RequireUtils`. Keeping deprecated code creates confusion and increases the codebase maintenance surface.

Recommendation: Apply the following cleanup:

1. Remove unchecked blocks around loop increments where overflow is provably impossible
2. Delete `src/utils/CalldataDecode.sol` (or move to `test/utils/`), and remove its import and using directive from `HydrateProxy.sol`
3. Delete `src/TrailsValidator.sol` and all references in tests/documentation

S5

Unknown `behaviorOnError` Values Cause Failed Calls to Emit Success Events

Acknowledged

i Update

Marked as "Acknowledged" by the client.

The client provided the following explanation:

This pattern follows the upstream Calls logic in the Sequence v3 wallet contracts

File(s) affected: `src/modules/HydrateProxy.sol`

Description: In `HydrateProxy._hydrateExecute()` , the contract handles three known `behaviorOnError` values: `0` (`IGNORE_ERROR`), `1` (`REVERT_ON_ERROR`), and `2` (`ABORT_ON_ERROR`). However, if a call fails and `behaviorOnError` has an unknown value (e.g., `3`), none of the conditional branches match:

```
if (!success) {  
    if (call.behaviorOnError == Payload.BEHAVIOR_IGNORE_ERROR) {  
        errorFlag = true;  
        emit Calls.CallFailed(hydratableOpHash, i, LibOptim.returnData());  
        continue;  
    }  
    if (call.behaviorOnError == Payload.BEHAVIOR_REVERT_ON_ERROR) {  
        revert Calls.Reverted(decoded, i, LibOptim.returnData());  
    }  
    if (call.behaviorOnError == Payload.BEHAVIOR_ABORT_ON_ERROR) {  
        emit Calls.CallAborted(hydratableOpHash, i, LibOptim.returnData());  
        break;  
    }  
}  
// Falls through - no errorFlag set  
emit Calls.CallSucceeded(hydratableOpHash, i);
```

Execution falls through without setting the `errorFlag`, and the function emits `CallSucceeded` despite the call having failed. While `behaviorOnError` is typically set by trusted SDK/API code, values `>= 3` are achievable via packed payload encoding where `behaviorOnError = (flags & 0xC0) >> 6`. This does not cause direct fund loss but breaks observability: monitoring systems relying on events will incorrectly treat failed calls as successful, and downstream logic depending on `onlyFallback` semantics may execute incorrectly. The test `testFuzz_hydrateExecute_behavior3_fallthrough_emitsSucceeded` confirms this behavior occurs.

Recommendation: Add explicit handling for unknown `behaviorOnError` values to prevent misleading events:

```
if (!success) {
    if (call.behaviorOnError == Payload.BEHAVIOR_IGNORE_ERROR) {
        errorFlag = true;
        emit Calls.CallFailed(hydratableOpHash, i, LibOptim.returnData());
        continue;
    }
    if (call.behaviorOnError == Payload.BEHAVIOR_REVERT_ON_ERROR) {
        revert Calls.Reverted(decoded, i, LibOptim.returnData());
    }
    if (call.behaviorOnError == Payload.BEHAVIOR_ABORT_ON_ERROR) {
        emit Calls.CallAborted(hydratableOpHash, i, LibOptim.returnData());
        break;
    }
    // Handle unknown values explicitly
    revert UnknownBehaviorOnError(call.behaviorOnError);
}
```

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>

- 69a...1c9 ./src/TrailsUtils.sol
- 84f...75c ./src/TrailsValidator.sol
- 426...728 ./src/modules/HydrateProxy.sol
- 8d9...db0 ./src/modules/MalleableSapient.sol
- 504...0a7 ./src/modules/RequireUtils.sol
- 439...22c ./src/modules/Sweepable.sol
- 0ce...f50 ./src/utils/CalldataDecode.sol
- 129...fd5 ./src/utils/ReplaceBytes.sol

Test Suite Results

We ran the tests by `forge test`.

```
Ran 12 tests for test/CalldataDecode.t.sol:CalldataDecodeTest
[PASS] testFuzz_decodeBytesBytes_roundTrip(bytes32,bytes32,uint256,uint256) (runs: 256, μ: 20367, ~: 20464)
[PASS] testFuzz_decodeBytesBytes_roundTrip_direct(bytes,bytes) (runs: 256, μ: 13928, ~: 13880)
[PASS] test_decodeBytesBytes_reverts_aDataRel_outOfBounds() (gas: 11585)
[PASS] test_decodeBytesBytes_reverts_aDataRel_overflow() (gas: 11651)
[PASS] test_decodeBytesBytes_reverts_aEndRel_outOfBounds() (gas: 11815)
[PASS] test_decodeBytesBytes_reverts_bDataRel_outOfBounds() (gas: 11640)
[PASS] test_decodeBytesBytes_reverts_bEndRel_outOfBounds() (gas: 12018)
[PASS] test_decodeBytesBytes_reverts_lengthOutOfBounds() (gas: 12361)
[PASS] test_decodeBytesBytes_reverts_misalignedOffset() (gas: 12268)
[PASS] test_decodeBytesBytes_reverts_offsetsTooSmall() (gas: 11407)
[PASS] test_decodeBytesBytes_reverts_short() (gas: 15595)
[PASS] test_decodeBytesBytes_roundTrip() (gas: 12753)
Suite result: ok. 12 passed; 0 failed; 0 skipped; finished in 434.18ms (350.26ms CPU time)
```

```
Ran 2 tests for test/Sweepable.t.sol:SweepableTest
[PASS] testFuzz_sweep(address,uint256[],uint256,bool) (runs: 256, μ: 1246192, ~: 1255039)
[PASS] testFuzz_sweepZeroBalances(address,uint256,bool) (runs: 256, μ: 600379, ~: 591567)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 565.24ms (531.91ms CPU time)
```

```
Ran 12 tests for test/ReplaceBytes.t.sol:ReplaceBytesTest
[PASS] testFuzz_replaceAddress(bytes32,uint256,address) (runs: 256, μ: 40740, ~: 42426)
[PASS] testFuzz_replaceUint256(bytes32,uint256,uint256) (runs: 256, μ: 41355, ~: 42918)
[PASS] test_replaceAddress_exactEndOffsets() (gas: 59829)
[PASS] test_replaceAddress_inPlace_aliasReflectsMutation() (gas: 31791)
[PASS] test_replaceAddress_offsets_allAlignments() (gas: 1422408)
[PASS] test_replaceAddress_reverts_outOfBounds() (gas: 15402)
[PASS] test_replaceAddress_reverts_overflowEnd() (gas: 6987)
[PASS] test_replaceUint256_exactEndOffsets() (gas: 59703)
[PASS] test_replaceUint256_inPlace_aliasReflectsMutation() (gas: 34621)
[PASS] test_replaceUint256_offsets_allAlignments() (gas: 1780094)
[PASS] test_replaceUint256_reverts_outOfBounds() (gas: 11148)
[PASS] test_replaceUint256_reverts_overflowEnd() (gas: 6907)
Suite result: ok. 12 passed; 0 failed; 0 skipped; finished in 928.20ms (467.57ms CPU time)
```

```
Ran 23 tests for test/HydrateProxy.t.sol:HydrateProxyTest
[PASS] testFuzz_handleSequenceDelegateCall_decodesAndExecutes(bytes4) (runs: 256, μ: 2361438, ~: 2361438)
[PASS] testFuzz_hydrateExecuteAndSweep_explicitTarget_andNoBalances(address) (runs: 256, μ: 1962692, ~: 1962692)
[PASS] testFuzz_hydrateExecuteAndSweep_revertsWhenEthSweepFails(uint96) (runs: 256, μ: 2043320, ~: 2043213)
[PASS] testFuzz_hydrateExecuteAndSweep_sweepsEthAndTokens_toMsgSenderWhenZeroTarget(uint96,uint128)
(runs: 256, μ: 2578376, ~: 2578800)
[PASS] testFuzz_hydrateExecute_abortOnError_aborts(bytes,bytes4) (runs: 256, μ: 1963153, ~: 1963150)
[PASS] testFuzz_hydrateExecute_behavior3_fallthrough_emitsSucceeded(bytes,bytes4) (runs: 256, μ: 2030989,
~: 2030984)
[PASS] testFuzz_hydrateExecute_delegateCallAllowedInCallerContext_execute(bytes) (runs: 256, μ: 1990617,
~: 1990465)
[PASS] testFuzz_hydrateExecute_delegateCallNotAllowed_reverts(bytes) (runs: 256, μ: 1891100, ~: 1891095)
[PASS] testFuzz_hydrateExecute_emptyHydratePayload_executes(bytes4) (runs: 256, μ: 1962309, ~: 1962309)
[PASS] testFuzz_hydrateExecute_hydratesAllDataFlags_andExecutes(bytes32) (runs: 256, μ: 5039831, ~:
5039831)
[PASS] testFuzz_hydrateExecute_ignoreError_enablesOnlyFallback(bytes4,bytes4,bytes4) (runs: 256, μ:
3332124, ~: 3332124)
[PASS] testFuzz_hydrateExecute_notEnoughGas_reverts(uint256) (runs: 256, μ: 1896095, ~: 1895963)
[PASS] testFuzz_hydrateExecute_revertOnError_reverts(bytes) (runs: 256, μ: 1553581, ~: 1553576)
[PASS] testFuzz_hydrateExecute_unknownHydrateFlag_reverts(uint8) (runs: 256, μ: 1891323, ~: 1891323)
[PASS] testFuzz_receive_acceptsEth(uint96) (runs: 256, μ: 1491899, ~: 1491845)
[PASS] test_hydrateExecute_hydrateTo_ANY_ADDRESS_setsToProvidedAddress() (gas: 2366908)
[PASS] test_hydrateExecute_hydrateTo_MESSAGE_SENDER_setsToCaller() (gas: 2294275)
[PASS] test_hydrateExecute_hydrateTo_SELF_setsToProxy() (gas: 1896165)
[PASS] test_hydrateExecute_hydrateTo_TRANSACTION_ORIGIN_setsToOrigin() (gas: 2767137)
```

```
[PASS] test_hydrateExecute_hydrateValue_ANY_ADDRESS_setsToProvidedBalance() (gas: 1991238)
[PASS] test_hydrateExecute_hydrateValue_MESSAGE_SENDER_setsToCallerBalance() (gas: 2388501)
[PASS] test_hydrateExecute_hydrateValue_SELF_setsToProxyBalance() (gas: 1996237)
[PASS] test_hydrateExecute_hydrateValue_TRANSACTION_ORIGIN_setsToOriginBalance() (gas: 2390335)
Suite result: ok. 23 passed; 0 failed; 0 skipped; finished in 1.20s (3.21s CPU time)
```

Ran 7 tests for test/MalleableSapient.t.sol:MalleableSapientTest

```
[PASS] testFuzz_recoverSapientSignature_chainId_vs_noChainId(bytes32,uint256,uint256,uint8) (runs: 256, μ: 479822, ~: 480807)
[PASS] testFuzz_recoverSapientSignature_emptySignature_matchesExpected(bytes32,uint256,uint256,uint8) (runs: 256, μ: 469248, ~: 469532)
[PASS] testFuzz_recoverSapientSignature_reverts_nonTransactions(uint8) (runs: 256, μ: 451966, ~: 451966)
[PASS] testFuzz_recoverSapientSignature_withSections_matchesExpected(bytes32,uint256,uint256,uint8,uint8) (runs: 256, μ: 508193, ~: 504603)
[PASS] test_recoverSapientSignature_chainId_included() (gas: 460138)
[PASS] test_recoverSapientSignature_invalidRepeatSection_reverts(bytes32,uint256,uint256,uint8,uint8) (runs: 256, μ: 502297, ~: 498498)
[PASS] test_recoverSapientSignature_noChainId_zeroUsed() (gas: 457593)
Suite result: ok. 7 passed; 0 failed; 0 skipped; finished in 3.08s (1.17s CPU time)
```

Ran 18 tests for test/RequireUtils.t.sol:RequireUtilsTest

```
[PASS] testFuzz_requireERC1155Approval(address,address,bool) (runs: 256, μ: 1227603, ~: 1220232)
[PASS] testFuzz_requireERC1155ApprovalSelf(address,address,bool) (runs: 256, μ: 1230858, ~: 1239345)
[PASS] testFuzz_requireERC721Approval(address,address,uint256,address,bool) (runs: 256, μ: 1100926, ~: 1108437)
[PASS] testFuzz_requireERC721ApprovalSelf(address,address,uint256,address,bool) (runs: 256, μ: 1101766, ~: 1093134)
[PASS] testFuzz_requireMinBalance(address,uint96,uint96) (runs: 256, μ: 880151, ~: 880151)
[PASS] testFuzz_requireMinBalanceSelf(address,uint96,uint96) (runs: 256, μ: 880839, ~: 881224)
[PASS] testFuzz_requireMinERC1155Balance(address,uint256,uint128,uint128) (runs: 256, μ: 1237523, ~: 1236784)
[PASS] testFuzz_requireMinERC1155BalanceBatchSelf_passes(address,uint256[],bytes32) (runs: 256, μ: 1297536, ~: 1296026)
[PASS] testFuzz_requireMinERC1155BalanceBatchSelf_reverts_firstIndex(address,uint256[],bytes32) (runs: 256, μ: 1295767, ~: 1295082)
[PASS] testFuzz_requireMinERC1155BalanceBatch_passes(address,uint256[],bytes32) (runs: 256, μ: 1296938, ~: 1295728)
[PASS] testFuzz_requireMinERC1155BalanceBatch_reverts_firstIndex(address,uint256[],bytes32) (runs: 256, μ: 1294636, ~: 1294151)
[PASS] testFuzz_requireMinERC1155BalanceSelf(address,uint256,uint128,uint128) (runs: 256, μ: 1239190, ~: 1239361)
[PASS] testFuzz_requireMinERC20Allowance(address,address,uint128,uint128) (runs: 256, μ: 1467513, ~: 1468411)
[PASS] testFuzz_requireMinERC20AllowanceSelf(address,address,uint128,uint128) (runs: 256, μ: 1468411, ~: 1468280)
[PASS] testFuzz_requireMinERC20Balance(address,uint128,uint128) (runs: 256, μ: 1466008, ~: 1464773)
[PASS] testFuzz_requireMinERC20BalanceSelf(address,uint128,uint128) (runs: 256, μ: 1467342, ~: 1467090)
[PASS] testFuzz_requireNonExpired(uint48,uint48) (runs: 256, μ: 880059, ~: 880435)
[PASS] test_requireMinERC1155BalanceBatch_reverts_lengthMismatch() (gas: 1216647)
Suite result: ok. 18 passed; 0 failed; 0 skipped; finished in 3.29s (11.54s CPU time)
```

Ran 6 test suites in 3.36s (9.49s CPU time): 74 tests passed, 0 failed, 0 skipped (74 total tests)

Code Coverage

We ran the coverage by: `forge coverage --ir-minimum`. For those contracts in production use: `HydrateProxy`, `MalleableSapient`, `RequireUtils`, and, `Sweepable`, the coverage is very high.

File	% Lines	% Statements	% Branches	% Funcs
src/TrailsValidator.sol	0.00% (0/22)	0.00% (0/20)	0.00% (0/16)	0.00% (0/8)
src/modules/HydrateProxy.sol	97.03% (98/101)	97.22% (105/108)	83.33% (30/36)	100.00% (7/7)

File	% Lines	% Statements	% Branches	% Funcs
src/modules/MalleableSapient.sol	97.22% (35/36)	97.44% (38/39)	100.00% (6/6)	100.00% (3/3)
src/modules/RequireUtils.sol	98.48% (65/66)	98.25% (56/57)	100.00% (9/9)	100.00% (22/22)
src/modules/Sweepable.sol	100.00% (16/16)	100.00% (18/18)	100.00% (4/4)	100.00% (2/2)
src/utils/CalldataDecode.sol	4.17% (1/24)	0.00% (0/30)	0.00% (0/7)	100.00% (1/1)
src/utils/ReplaceBytes.sol	14.29% (2/14)	0.00% (0/14)	0.00% (0/2)	100.00% (2/2)
test/CalldataDecode.t.sol	100.00% (3/3)	100.00% (3/3)	100.00% (0/0)	100.00% (1/1)
test/HydrateProxy.t.sol	80.00% (8/10)	87.50% (7/8)	100.00% (0/0)	75.00% (3/4)
test/helpers/Mocks.sol	67.27% (37/55)	63.41% (26/41)	0.00% (0/8)	81.25% (13/16)
test/helpers/PackedPayloads.sol	100.00% (16/16)	94.44% (17/18)	50.00% (4/8)	100.00% (2/2)
Total	77.41% (281/363)	75.84% (270/356)	55.21% (53/96)	82.35% (56/68)

Changelog

- 2026-01-30 - Initial report
- 2026-02-05 - Fix review 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 1100 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, Solana, Polygon, TON, Cardano, Binance Smart Chain, Avalanche, Arbitrum, Flow
- DeFi: Lido, Ethena, Compound, Curve, Venus, PancakeSwap, Polymarket
- Infra/Staking: TrustWallet, Alchemy, Liquid Collective, Kiln, Galxe, API3, ssv.network, Luganodes
- Immersive: Virtuals Protocol, Wayfinder, OpenSea, Square Enix, Parallel, Camp, Decentraland
- Institutional: Visa, Circle, Revolut, Anthropic, OpenAI, Canton, Republiccc, Arkham, Sequoia

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

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.

