BAIL security

0x CrossChainReceiver Update

FINAL REPORT

January ‘2026

Disclaimer:

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

The content of this assessment is not an investment. The information provided in this report is for general informational purposes only and is not intended as investment, legal, financial, regulatory, or tax advice. The report is based on a limited review of the materials and documentation provided at the time of the audit, and the audit results may not be complete or identify all possible vulnerabilities or issues. The audit is provided on an "as-is," "where-is," and "as-available" basis, and the use of blockchain technology is subject to unknown risks and flaws.

The audit does not constitute an endorsement of any particular project or team, and we make no warranties, expressed or implied, regarding the accuracy, reliability, completeness, or availability of the report, its content, or any associated services or products. We disclaim all warranties, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We assume no responsibility for any product or service advertised or offered by a third party through the report, any open-source or third-party software, code, libraries, materials, or information linked to, called by, referenced by, or accessible through the report, its content, and the related services and products. We will not be liable for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract.

The contract owner is responsible for making their own decisions based on the audit report and should seek additional professional advice if needed. The audit firm or individual assumes no liability for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract. The contract owner agrees to indemnify and hold harmless the audit firm or individual from any and all claims, damages, expenses, or liabilities arising from the use or reliance on the audit report or the smart contract.

By engaging in a smart contract audit, the contract owner acknowledges and agrees to the terms of this disclaimer.

1. Project Details

Important:
Please ensure that the deployed contract matches the source-code of the last commit hash.

Project	0x - CrossChainReceiver (Update)- Audit Report
Website	0x.org
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/0xProject/0x-settler/pull/430/commits/026838c0d02d9729dca7b81f4c1e41cd9edbba0f
Resolution 1	https://github.com/0xProject/0x-settler/pull/498

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no changes made)	Failed resolution	Open
High						
Medium	1	1				
Low	1	1				
Informational	1			1		
Governance						
Total	3	2		1		

2.1 Detection Definitions

Severity	Description
High	The problem poses a significant threat to the confidentiality of a considerable number of users' sensitive data. It also has the potential to cause severe damage to the client's reputation or result in substantial financial losses for both the client and the affected users.
Medium	While medium level vulnerabilities may not be easy to exploit, they can still have a major impact on the execution of a smart contract. For instance, they may allow public access to critical functions, which could lead to serious consequences.
Low	Poses a very low-level risk to the project or users. Nevertheless the issue should be fixed immediately
Informational	Effects are small and do not post an immediate danger to the project or users
Governance	Governance privileges which can directly result in a loss of funds or other potential undesired behavior

3. Detection

MultiCallContext

MultiCallContext is a Context extension that makes a contract transparently compatible with a trusted MultiCall ERC-2771-style forwarder at EIP150_MULTICALL_ADDRESS, so that when calls are routed through MultiCall, _msgSender() and _msgData() reflect the original caller and payload, while direct calls behave normally.

- **Trusted forwarder wiring:** _MULTICALL() returns the canonical IMultiCall instance, and _isForwarded() reports true when the immediate caller is that MultiCall (or when an underlying context already considered it forwarded).
- **Sender unwrapping:** _msgSender(address multicall) detects when msg.sender == multicall and, in that case, decodes the last 20 bytes of calldata as the original sender; otherwise it just returns super._msgSender().
- **Data unwrapping:** _msgData(address multicall) returns super._msgData() with the trailing 20-byte ERC-2771 sender suffix stripped when the call is forwarded, and unchanged for direct calls.
- **Drop-in overrides:** Inheriting contracts call _msgSender() / _msgData() as usual and automatically get correct behavior for both direct calls and calls forwarded via MultiCall.

Privileged Functions

None.

Core Invariants:

INV 1: _MULTICALL() is the sole trusted forwarder and _isForwarded() is true only when the immediate caller is that address (or a forwarder trusted by a parent context).

INV 2: _msgSender(address multicall) returns super._msgSender() for non-forwarded calls and only decodes the last 20 bytes of calldata as the original sender when msg.sender == multicall and the calldata is long enough.

INV 3: _msgData(address multicall) returns super._msgData() unchanged for non-forwarded calls and, for forwarded calls, returns the same calldata but with exactly the 20-byte ERC-2771 sender suffix logically stripped.

Issue_01	Inverted ERC-2771 _msgData trimming condition
Severity	**Medium**
Description	In _msgData(address multicall), the assembly line *r.length :=* *sub(r.length, mul(0x14, lt(0x00, shl(0x60, xor(multicall, sender)))))* reduces r.length by 20 bytes when sender != multicall and leaves it unchanged when sender == multicall. For correct ERC-2771 semantics, the opposite should occur: only forwarded calls (sender == multicall, with an appended 20-byte sender suffix) should have 20 bytes stripped, while direct calls should see the full msg.data. As written, _msgData behaves contrary to the expectations implied by _msgSender and the comments, which can cause future callers that rely on _msgData to misinterpret calldata (e.g. missing the last 20 bytes on direct calls and incorrectly including the suffix on forwarded calls).
Recommendations	It is recommended to flip the condition to strip 20 bytes when sender == multicall, e.g.: *r.length :=* *sub(r.length, mul(0x14, iszero(shl(0x60, xor(multicall, sender)))))*
Comments / Resolution	Resolved, the function is now showing the expected behavior.

Issue_02	The _msgData's length might overflow when less than 4 bytes are sent
Severity	**Informational**
Description	The MultiCall contract only adds the sender to the calldata if the data length is greater than 3 bytes, i.e. contains at least a selector. However, the _msgData function does not handle that edge case and always subtracts 20 bytes from the calldata when sender == multicall. Therefore, when sending calldata with less than 4 bytes, the length would underflow on subtraction, because the MultiCall contract does not append any data.
Recommendations	Consider implementing a calldata length check similar to the one in the _msgSender function.
Comments / Resolution	Acknowledged, a comment was added to clarify this behavior and that it shouldn't be used within a fallback function.

CrossChainReceiverFactory

CrossChainReceiverFactory is the central contract that creates and governs 0x cross-chain receiver wallets. It ensures each user gets a deterministic receiver address per chain, wires those receivers behind a trusted MultiCall forwarder (for ERC-2771-style meta-transactions) and an optional wrapped native token, and enforces robust authorization and replay protection.

- **Receiver lifecycle:** Deploys minimal proxy receivers at predictable CREATE2 addresses based on a Merkle root and owner, and provides setOwner and cleanup hooks to manage and retire them safely.
- **Cross-chain integration:** Binds receivers to a chain-specific MultiCall and, if configured, a chain-specific IWrappedNative, with constructor-time code-hash and behavior checks to ensure correct deployment.
- **Secure execution & signatures:** Exposes metaTx and isValidSignature to support both Merkle-based delegation and EIP-712/ERC-7739 signatures, embedding the owner into nonces and using Permit2's unordered nonces to prevent replay or cross-proxy reuse.
- **Utility calls & fund recovery:** Provides call helpers for owner-authorized arbitrary calls and getFromMulticall to drain stuck ERC-20 or native assets from the shared MultiCall contract back into the receiver.

Privileged Functions

- setOwner
- call (both overloads)
- metaTx (in case of optional relayer)
- cleanup

Core Invariants:

INV 1: Each proxy address is uniquely determined by (factory, root, owner, _proxyInitHash) via CREATE2, and _verifyDeploymentRootHash must confirm that mapping.

INV 2: The configured MultiCall, WNATIVE, and immutable storage contracts must match expected code hashes and behavior or the constructor reverts.

INV 3: Every metaTx nonce (including the embedded owner bits) is single-use, enforced by Permit2's unordered nonce bitmap.

INV 4: All privileged operations (metaTx, approvals, arbitrary calls, cleanup) are only meaningful on proxies and must resolve the caller through _msgSender() (MultiCall-aware).

INV 5: Signature verification must always bind to the correct chain and proxy (via chainid, verifyingContract, or deployment root), preventing cross-proxy or cross-chain replay.

Issue_03	Off-by-4 length coupling between _hashMultiCall and metaTx calldata forwarding
Severity	**Low**
Description	In _hashMultiCall, the data buffer's length word is set to msgData.length (selector + args), but only msgData.length - 4 bytes of arguments are actually copied starting at data + 0x20. Later, metaTx reads that word as dataLength and calls MultiCall from data + 0x1c with exactly dataLength bytes. Bit-for-bit this means the call forwards 4 bytes of selector plus msgData.length - 4 bytes of arguments, so MultiCall sees correct (calls, contextdepth, nonce, deadline, signature, ...) prefix, but the bytes data object in memory advertises a length that doesn't match its actual initialized region. The current behavior works because these two off-by-4 mismatches cancel at the observable layer but are brittle and misleading.
Recommendations	It is recommended to update _hashMultiCall to store argsLength = msgData.length - 4 as data.length, and in metaTx change the call to use add(0x04, dataLength) as the length (still starting from add(0x1c, data)), so that data's length matches the copied arguments region.
Comments / Resolution	Resolved, the data object's length is now correctly set and used.