OpenZeppelin | security

OneSig Audit

USD₮0

June 2, 2025

Table of Contents

Summary

Type	Stablecoins	Total Issues	5 (0 resolved)
Timeline	From 2025-05-28 To 2025-05-30	Critical Severity Issues	0 (0 resolved)
Languages	Solidity	High Severity Issues	0 (0 resolved)
		Medium Severity Issues	0 (0 resolved)
		Low Severity Issues	0 (0 resolved)
		Notes & Additional Information	5 (0 resolved)

Scope

OpenZeppelin audited the LayerZero-Labs/onesig repository at commit a5ed4b2.

In scope were the following files:

```
packages/onesig-evm
└── contracts
    ├── MultiSig.sol
    └── OneSig.sol
```

System Overview

This audit covers the LayerZero implementation of the OneSig protocol, which builds upon the previous codebase audited under the Everdawn-Labs/OneSig repository. The current version only introduces minor improvements and refinements, with no significant changes to the overall architecture or core logic.

As in the previous version, the system enables secure cross-chain execution of pre-authorized transactions. A proposer can send a batch of transactions to an off-chain server for review. A group of designated signers collectively approves the batch, producing a Merkle root representing the authorized set. On-chain, an executor can execute transactions from the batch by submitting a valid Merkle proof along with sufficient signatures.

Each transaction batch is uniquely identified using a nonce and a deployment-specific identifier to enforce ordering and prevent replay. This model continues to be well-suited for scalable, multi-chain deployments where frequent updates are required, such as in the LayerZero ecosystem.

Security Model and Trust Assumptions

The same assumptions outlined in the previous audit apply to this version of the system. However, this audit is limited to the on-chain components and does not include the off-chain infrastructure responsible for constructing the Merkle tree. It is assumed that this off-chain system correctly implements the double-hashing and encoding logic as defined in the `encodeLeaf` function.

All configured signers are assumed to be trustworthy, and the signature threshold is expected to be set according to best practices. More specifically, the signature threshold is high enough relative to the total number of signers to prevent compromise.

The system allows signatures to be reused across different chains due to the use of a static EIP-712 domain. This behavior is acknowledged and accepted by the project team as intentional, based on the protocol's documented design choices.

Privileged Roles

The set of privileged roles remains consistent with the previous audit. The designated signers are responsible for authorizing transaction batches by signing Merkle roots off-chain. They also have the authority to update the signer set and the contract's seed value, both of which are integral to the signature verification process. These privileged operations, like transaction execution, require the same threshold of signatures and cannot be performed unilaterally by any individual signer.

Acknowledged Issues

The LayerZero Labs team acknowledges a number of previously reported issues raised by OpenZeppelin and other independent audit firms. These issues have been documented in past audits and can be found in the appendix.

Notes & Additional Information

N-01 State Variable Visibility Not Explicitly Declared

Within `MultiSig.sol`, the `SIGNATURE_LENGTH` state variable lacks an explicitly declared visibility.

For improved code clarity, consider always explicitly declaring the visibility of state variables, even when the default visibility matches the intended visibility.

N-02 Incomplete Docstring

Within `OneSig.sol`, the `SeedSet` event has an incomplete docstring. More specifically, the `seed` parameter is not documented.

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of a contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

N-03 Possible Duplicate Event Emissions

Some setter functions in the codebase emit events without verifying whether the new value differs from the current one. This can result in duplicate event emissions when the same value is set repeatedly, causing the event to be emitted even though no actual change has occurred.

While the functions are permissioned and not prone to external spamming, redundant event emissions can still lead to confusion or inefficiencies for off-chain clients or indexers that rely on event data to detect meaningful state changes.

Throughout the codebase, multiple instances of potential duplicate event emissions were identified:

- The `_setThreshold` sets the `threshold` and emits an event without checking if the value has changed.
- The `_setSeed` sets the `seed` and emits an event without checking if the value has changed.

Consider adding a check statement to revert the transaction if the value being set is identical to the existing value.

N-04 Misleading Documentation

The internal documentation of the `verifyNSignatures` function in `MultiSig.sol` states that the function reverts if the number of signatures does not match the threshold, suggesting that an exact match is required. However, the implementation only checks that the number of signatures is not less than the threshold.

This means that any number of signatures equal to or greater than the threshold is accepted, which does not align with the stated behavior. This mismatch may lead developers or auditors to assume that the function enforces stricter validation than it actually does. While this does not introduce a direct vulnerability, it can cause confusion or unintended assumptions about how the signature verification works.

To improve code clarity and prevent misunderstanding, consider updating the documentation to clarify that the function accepts at least the threshold number of signatures. Alternatively, consider modifying the logic to require an exact match.

N-05 Ambiguous Error Handling

The `verifyNSignatures` function in `MultiSig.sol` reverts with `SignatureError` for two distinct failure cases: when the total signature data size is not a multiple of 65 bytes, and when the number of extracted signatures is less than the required threshold. These are logically separate conditions but are reported using the same error. Using the same error for both scenarios can make debugging more difficult and reduce clarity for developers and off-chain consumers, who may not be able to easily distinguish which condition caused the revert.

Consider introducing a separate error for each case to clearly signal the specific reason for failure. Doing so will help improve error traceability and diagnosis.

Conclusion

The LayerZero implementation of the OneSig protocol maintains the original design principles while introducing minor refinements for improved clarity and usability. This version preserves the core architecture, including off-chain batching, multi-signer authorization, and on-chain Merkle proof-based execution.

The system continues to provide a secure and scalable solution for cross-chain transaction execution, with strong protections against replay and misordering through the use of nonces and chain-specific identifiers. The overall design remains appropriate for high-throughput, multi-chain environments such as those enabled by LayerZero.

The codebase remains clean and well-structured, with no critical issues identified. The audit findings were limited to note-level observations and suggestions aimed at improving code clarity and developer experience.

Appendix

Acknowledged Issues

Issue Title	Severity	Description
Seed Invalidation (Guardian L-01)	Low	Malicious actors can frontrun a `setSeed` call to execute transactions (via `executeTransaction`) from a Merkle root that signers intend to invalidate, leading to unexpected outcomes and nonce increments.
Lacking Gas Validations Allows Censoring (Guardian L-03)	Low	The executeTransaction function lacks gas validation for individual external calls, allowing any address to invoke it and potentially manipulate execution by providing insufficient gas, which can alter behavior, especially in try/catch blocks, and cause earlier calls to deplete gas needed for later ones.
Network Fork Replay Attacks (Guardian L-04)	Low	On chain forks, signatures can be replayed, risking unintended execution.
Stale Transaction Execution on L2 (Guardian L-05)	Low	On L2s with sequencers (e.g., Arbitrum), a sequencer outage >24 hours can allow stale signatures to execute with outdated `block.timestamp`. Requires extreme conditions.
Multiple Signed Merkle Roots (Guardian L-06)	Low	If multiple signed Merkle roots with overlapping nonces exist for the same OneSig safe, an arbitrary user can mix transactions across them via `executeTransaction`, leading to unexpected execution outcomes.

Issue Title	Severity	Description
executeTransaction treats EOAs and Contracts the same when calling (SpearBit L-01)	Low	Although `executeTransaction` correctly checks for call success, expected behaviors, like sending value to EOAs, can mask silent failures, especially in multichain setups where missing contracts on some chains or future EIP-7702 changes may cause unintended execution or reverts without clear errors.
Unexpected Balance (Guardian I-01)	Informational	A signature with non-zero `value` can drain OneSig's balance if called without `msg.value`.
Unnecessary Return Data (Guardian I-04)	Informational	`.call` in `executeTransaction` loads unused return data, wasting gas and affecting efficiency.
Smart Contract Signer Exclusion (Guardian I-05)	Informational	`ECDSA.recover` only supports EOA signatures, excluding contract-based signers (e.g., DAO wallets), limiting use cases.
Missing Event in Receive Function (Guardian I-06)	Informational	The `receive` function lacks an event for tracking native token deposits, reducing transparency.
Time Drift Across Networks May Be Unexpected (Guardian I-07)	Informational	Because the expiration time applies globally to the Merkle root, differing `block.timestamp` values across chains can allow transactions intended to be expired everywhere to remain valid on some chains.
Low-Level Calls Do Not Validate Contract Code (Guardian I-08)	Informational	The `executeTransaction` function uses low-level calls without verifying that the target is a valid contract, allowing calls to EOAs or non-existent contracts to silently succeed and misleadingly indicate successful execution.
Stuck Nonce (Guardian I-09)	Informational	If a transaction fails, the `nonce` doesn't increment, blocking subsequent calls until resubmitted or invalidated via `setSeed`

Issue Title	Severity	Description
Unexpected Reinstatement Of Transactions (Guardian I-10)	Informational	Resetting `seed` to a previously used value can reinstate invalidated signatures, allowing execution if signatures and `nonce` are valid. Requires signer error.
Useful Error Data (ID-06, Guardian)	Informational	`execution error` lacks data due to transaction’s `nonce`, blocking subsequent debugging and UX.
Version Compatibility (Guardian I-13)	Informational	Cross-chain OneSig instances must share the same `VERSION`, complicating deployments if mismatched.
Token Transfer May Silently Fail (Guardian I-14)	Informational	Failure of external calls (e.g., USDT transfers) returning `false` don’t revert, emitting `true` despite failure.
Missing EIP-5267 (Guardian I-15)	Informational	The OneSig contract does not expose functionality to read the signing domain. This functionality was standardized with EIP5267 which introduces the eip712Domain function.
Unwieldy Seed Behavior (Guardian I-18)	Informational	Although the documentation claims seeds prevent signature replay, they must remain identical across OneSig instances, making it impractical to invalidate a Merkle root on just one chain without complex coordination across all chains.
Arbitrary Transaction Ordering (Guardian I-19)	Informational	Since `executeTransaction` can be called by any address once signatures are available, malicious actors can manipulate execution order leading to potential protocol disruption (e.g. front-running swaps).
Threshold and Signer Consistency (Guardian I-21)	Informational	Varying thresholds/signers across cross-chain deployments complicates UX and increases error risk.

Issue Title	Severity	Description
Typographical and Documentation Issues (Guardian I-17, I-20; Paladin #02)	Informational	Issues include magic numbers, incomplete NatSpec, grammatical errors, and misleading comments in `PROTOCOL.md`, affecting readability.
Math optimization in **`verifyNSignatures`** (Spearbit I-02)	Informational	It is not necessary to perform two MUL operations for each signature
Missing Docstrings (OpenZeppelin N-02)	Informational	`LEAF_ENCODING_VERSION` lacks a NatSpec docstring, reducing code clarity.
Lack of Security Contact (OpenZeppelin N-03)	Informational	No security contact (e.g., email, ENS) is specified, complicating vulnerability disclosures.
Function Visibility Overly Permissive (OpenZeppelin N-04)	Informational	Several functions use overly permissive `public` visibility (e.g., `getSigners`, `setSeed`, `executeTransaction`) and could be marked `external` to better reflect intended usage and reduce gas costs.
Floating Pragma (OpenZeppelin N-05, Guardian I-16)	Informational	Using `^0.8.22` risks compilation differences.
Lack of Indexed Event Parameters (OpenZeppelin N-06)	Informational	Events (`SignerSet`, `ThresholdSet`, `SeedSet`, `TransactionExecuted`) lack indexed parameters, hindering off-chain filtering.
EVM Version Compatibility (OpenZeppelin N-08)	Informational	No `evmVersion` specified, defaulting to `cancun`, risks deployment failures on chains not supporting new opcodes (e.g., `PUSH0`, `MCOPY`).