BAIL security

Kelp DAO

verify-onchain-config.ts Script Review Report

FINAL REPORT

May ‘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.

Scope Limitation - Script Review Only. This assessment is limited exclusively to the reviewed verifier script, verify-onchain-config.ts, and only to the script logic, checks, report rendering, and directly referenced configuration necessary to evaluate that script. This report is not a smart contract audit, protocol audit, bridge audit, infrastructure audit, formal verification, penetration test, deployment review, or continuous monitoring engagement. No opinion is expressed on the security, correctness, solvency, accounting, economics, governance, operational controls, private-key management, deployment process, or production behavior of the KernelDAO, rsETH, KERNEL, LayerZero, Movement, Ethereum, or any other related protocol, chain, service, or third-party system, except to the limited extent explicitly described as part of the reviewed script behavior.

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.

Review Type Notice. This engagement is a script review, not a protocol or smart contract audit. The reviewed artifact is the verifier script identified in this report. Any protocol contract, deployed bytecode, LayerZero endpoint, message library, DVN, executor, Movement module, RPC endpoint, REST API, governance action, admin action, or production configuration is outside the scope of this review unless the report expressly states that the script reads or validates that specific item. A successful review or resolution of findings in this report must not be interpreted as confirmation that the underlying protocol, bridge pathway, cross-chain configuration, or any related production system is secure.

1. Project Details

Important: Please ensure that the reviewed script, supporting configuration files, dependency versions, and commit hash match the materials listed in this report. This review does not verify deployed contract bytecode, contract source-code equivalence, deployment correctness, or any production protocol state, except where such state is read by the reviewed script for the limited purpose of evaluating the script's behavior.

Project	Kelp DAO – verify-onchain-config.ts – Script Review Report
Website	kerneldao.com
Language	TypeScript
Methods	Manual Analysis
Reviewed Artifact	verify-onchain-config.ts
Review Type	Script review / verifier logic review
Github repository	---
Resolution 1	---

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no changes made)	Failed resolution	Open
High	5	5				
Medium	2	2				
Low	1	1				
Informational						
Governance						
Total	8	8				

2.1 Detection Definitions

Severity	Description
High	A defect in the verifier that allows a broken on-chain configuration to be reported as correct. The script's output cannot be trusted as evidence on the affected dimension; a misconfiguration would pass through unflagged.
Medium	A check the script claims to perform but does not, or performs only partially. The report is incomplete on the affected dimension: a misconfiguration will not be falsely attested as correct, but neither will it be flagged.
Low	Cosmetic, stale, or efficiency-only issues that do not affect the correctness of the attestation but could mislead a future reader of the script.
Informational	Design notes, scope acknowledgements, or suggestions for future hardening that are not bugs in the current implementation.

Scope of Review

This report evaluates whether the reviewed script performs and reports the checks it purports to perform. The review is limited to the correctness, completeness, and consistency of the script's verification logic as applied to the invariants expressly encoded in the script and described in this report.

The report does not provide assurance that the underlying protocol, smart contracts, bridge design, LayerZero configuration, Movement-side modules, DVN behavior, executor behavior, governance processes, admin controls, private keys, deployments, or production operations are secure or correct. Any incident, loss, exploit, outage, misconfiguration, governance action, third-party failure, chain-level issue, or state change that arises outside the reviewed script or outside the expressly reviewed script invariants is outside the scope of this assessment.

A passing script output or resolved finding status means only that the script was assessed against the stated review scope. It must not be treated as a certification, warranty, or guarantee of protocol security, cross-chain safety, production correctness, or future system behavior.

3. Detection

Script: verify-onchain-config.ts

The verifier verify-onchain-config.ts is a standalone TypeScript program that reads on-chain state via public RPC endpoints and emits a per-chain markdown report intended to attest that the post-incident hardening (V1-V5 default-config swap closures, hub-and-spoke topology, owner / delegate co-location, executor / max-message-size / enforced-gas constraints) holds for every active rsETH and KERNEL pathway. It targets 27 chains for rsETH (Ethereum hub + 25 EVM spokes + Movement) and 3 chains for KERNEL (Ethereum + Arbitrum + BSC). It has a single external dependency (ethers v5) and calls only view functions, it does not sign or send transactions.
For each token (rsETH, KERNEL) and each chain in its topology, the initial script reads and renders the following:

1) OFT.owner(): the OApp owner address, displayed in the contracts table

2) EndpointV2.delegates(oapp): the LayerZero delegate, displayed in the contracts table

3) EndpointV2.getReceiveLibrary(oapp, srcEid): returns (lib, isDefault). The script renders pinned ✓ when isDefault == false

4) EndpointV2.getSendLibrary(oapp, dstEid): returns the send library address. The script renders pinning by comparing the returned address to the chain's hardcoded canonical SendUln302

5) SendUln302.getAppUlnConfig(oapp, dstEid): read against the hardcoded canonical SendUln302 address. Returns requiredDVNs, confirmations, optionalDVNCount, optionalDVNThreshold, optionalDVNs. The script renders DVN quorum and confirmation cells

6) ReceiveUln302.getAppUlnConfig(oapp, srcEid): read against the hardcoded canonical ReceiveUln302 address. Same shape and rendering as above

7) SendUln302.executorConfigs(oapp, dstEid): read against the hardcoded canonical SendUln302 address. Returns (maxMessageSize, executor). The script renders the executor cell as pinned ✓ when the executor is non-zero.

The hub-and-spoke topology is enforced by an iteration filter inside the script (deployment.chain === token.hubChain || peer.slug === token.hubChain), not by reading any on-chain peer state. Cells that are not generated by the iteration are not included in the report.

For the Movement spoke (eid 30325), only one read is performed: an HTTP GET to the public Movement REST API for the OAppStore resource at the rsETH OFT module address, surfacing admin and delegate. No Movement-side ULN config, executor, enforced options, or peer-mapping reads are performed.

Correctness of the verifier implementation was validated against live on-chain state at **2026-05-08 17:02:29** by re-running the script and confirming a byte-identical report (modulo the `Generated:` timestamp). Any subsequent change to either verify-onchain-config.ts or the on-chain configuration falls outside the scope of this attestation and would require a fresh review.

No Continuous Monitoring. This report reflects a point-in-time review of the reviewed script and the information available during the assessment. The review does not include continuous monitoring of the script, repositories, deployments, on-chain state, protocol configuration, governance actions, admin actions, RPC responses, REST API responses, third-party dependencies, or chain-level behavior. Any change after the reviewed commit or stated validation timestamp requires a separate review before this report may be relied upon for the changed state.

Data Source Reliance. The reviewed script depends on external data sources, including public RPC endpoints, REST APIs, ABIs, hardcoded addresses, and chain responses. This review does not guarantee the availability, correctness, finality, consistency, or non-manipulation of those data sources. A correct script may still produce incomplete, stale, inconsistent, or misleading output if an external data source is unavailable, incorrect, forked, censored, rate-limited, manipulated, or otherwise unreliable.

Furthermore, any additional context which is part of the RSETH protocol but not reflected within this script is fully out of the scope for this review. Only the content of the script is validated

Excluded Incidents and Non-Covered Risks

For avoidance of doubt, this report does not cover and shall not be interpreted as accepting responsibility for incidents, losses, defects, outages, exploits, misconfigurations, or failures arising from:

- protocol contract vulnerabilities or accounting defects
- bridge design flaws or cross-chain message execution behavior outside the reviewed script
- LayerZero endpoint, message library, DVN, executor, or default-configuration behavior
- Movement, Ethereum, or other chain-level behavior, reorgs, finality assumptions, node behavior, or API behavior
- RPC, REST API, infrastructure, CI/CD, deployment, monitoring, or operational failures
- governance actions, admin actions, compromised keys, signer mistakes, multisig behavior, or emergency interventions
- changes to the script, configuration files, dependencies, reviewed commit, on-chain state, or protocol state after the stated review timestamp
- economic, liquidity, market, oracle, slashing, validator, or business-logic risks
- any condition not expressly checked by the reviewed script and described as in scope in this report.

This report is limited to the reviewed script and its expressly described verification behavior. Any use of this report to infer broader protocol safety, production correctness, or absence of risk is outside the intended purpose of the review.

Privileged Functions

- None

Core Invariants:

INV 1: **DVN quorum.** Required-DVN set is {Canary, Horizen, LayerZero Labs, Nethermind} on every EVM pathway, or {P2P, Horizen, LayerZero Labs, Nethermind} on the Movement route (P2P substitutes for Canary, since Canary does not currently operate a Movement DVN). optionalDVNCount, optionalDVNThreshold are zero.

INV 2: **Confirmations.** 64 on every pathway whose source chain is Ethereum or any EVM spoke; 30,000 when the source chain is Movement (calibrated to give a wall-clock finality window equivalent to Ethereum's 64-slot threshold given Movement's sub-second block production).

INV 3: **Send-library pinning.** The OApp's send library for every pathway is pinned per-OApp (isDefaultSendLibrary == false) and equals the canonical SendUln302 for the source chain.

INV 4: **Receive-library pinning.** Same shape on the receive side: isDefault == false and lib == canonical_recvLib.

INV 5: **ULN config pinning.** Per-OApp ULN config is explicitly written for both send and receive directions, so default-config swaps by the message-library owners cannot weaken the pathway.

INV 6: **Executor pinning (V5 closure).** Per-OApp executor is set to a concrete, non-zero address equal to the canonical executor for the source chain.

INV 7: **maxMessageSize.** Set to 10000 on every send-side pathway.

INV 8: **Enforced LZ_RECEIVE gas.** Set to 200000 on both SEND and SEND_AND_CALL for every send-side pathway.

INV 9: **owner == delegate.** The OApp's owner() and EndpointV2.delegates(oapp) resolve to the same address per chain, a single account holds both privileged roles (the only callers that can modify any per-pathway value).

INV 10: **Hub-and-spoke topology.** Every spoke OApp's peers[hub.eid] equals the hub OFTAdapter address (bytes32-padded); the hub OApp's peers[spoke.eid] equals each spoke's OFT address (bytes32-padded); every other peer entry on every OApp is bytes32(0) (no L2 ↔️ L2 wiring).

Issue_01	Send-library pinning is checked with the wrong primitive
Severity	**High**
Description	The script attempts to infer send-library pinning by comparing the address returned from EndpointV2.getSendLibrary(oapp, dstEid) to the chain's canonical SendUln302. The cell logic is inverted: when the returned library matches the canonical (the expected-safe case) the cell is rendered silently with no status; when it does not match (often the unsafe case), the cell renders as pinned ✓. The check therefore neither confirms pinning nor flags non-canonical configurations. An OApp using the protocol default, the exact case V2 closure is meant to detect, is invisible to this check.
Recommendations	Replace the heuristic with an explicit call to EndpointV2.isDefaultSendLibrary(oapp, dstEid) returns (bool). This view function exists on EndpointV2 and is the symmetric counterpart of the receive side's isDefault flag. Combined with the canonical-match check from issue (2), pinned ✓ should render only when isDefault == false AND the resolved library address equals the canonical SendUln302 for that chain.
Comments / Resolution	Fixed, followed recommendation. EndpointV2.isDefaultSendLibrary is now called per pathway and combined with the canonical-match check.

Issue_02	Resolved library is never compared to the canonical (both directions)
Severity	**High**
Description	On the receive side the script calls getReceiveLibrary and uses the isDefault flag, but never compares the returned lib to the canonical ReceiveUln302. A pathway that is pinned (isDefault == false) but pinned to an attacker-deployed or otherwise non-canonical library renders as pinned ✓ with no warning. The same gap exists on the send side. An owner / delegate compromise that swaps any pathway's library to a malicious contract bypasses DVN verification entirely while passing this check.
Recommendations	For every pathway in both directions, compare the resolved library address (returned by getSendLibrary / getReceiveLibrary) to the canonical SendUln302 / ReceiveUln302 address for that chain. Render the cell as ⚠ non-canonical 0x... when they differ, regardless of whether the library is in default state or pinned. Treat pinned ✓ as a state that requires both isDefault == false AND lib == canonical.
Comments / Resolution	Fixed, followed recommendation. Explicit libraryMatchesCanonical comparison in both directions, with a dedicated check that fires when the resolved library differs from the canonical.

Issue_03	ULN and executor configs are read from the canonical library, not the actually-resolved one
Severity	**High**
Description	SendUln302.getAppUlnConfig, ReceiveUln302.getAppUlnConfig, and SendUln302.executorConfigs are called against the hardcoded canonical addresses (chain.sendLib, chain.recvLib) rather than against the library the OApp actually resolves to via getSendLibrary / getReceiveLibrary. If a pathway has been pointed at a non-canonical library, the verifier still reads the canonical contract's per-OApp config, which will typically be zero or unused, and renders the resulting DVN set / confirmations / executor as if those values were enforced on the live pathway. They are not.
Recommendations	Resolve the library via the EndpointV2 view function first, then read getAppUlnConfig and executorConfigs from that resolved address. Combined with issue (2), this makes the verification surface internally consistent: the cells the report shows are the cells actually enforced on the message at send / receive time.
Comments / Resolution	Fixed, followed recommendation. ULN and executor reads are now routed through the resolved library address on both EVM and Movement.

Issue_04	Hub-and-spoke topology is assumed by iteration order, not verified
Severity	High
Description	The "L2 ↔️ L2 pathways have been removed; peers cleared" claim from the README is enforced only by an iteration filter inside the script: if (deployment.chain === token.hubChain || peer.slug === token.hubChain). Cells the iteration does not generate are simply absent from the report, not verified to be cleared on-chain. The script never calls OApp.peers(eid). If any spoke OApp still holds a non-zero peers[someL2Eid] from before the hub-and-spoke flatten, for instance because a setPeer(eid, bytes32(0)) call was missed during the migration, the report renders ✓ everywhere because the row is simply not generated. Stale L2-L2 peers re-introduce the exact bypass the hub-and-spoke topology was meant to eliminate: an inbound from an L2 with a stale peer entry on the destination spoke is accepted and credited *without ever transiting the hub-side OFTAdapter accounting*. The hub-and-spoke property is what makes the lockbox-on-Ethereum design safe; without on-chain verification, the property is asserted by hopeful enumeration rather than by reading state.
Recommendations	For each OApp in the token, read OApp.peers(eid) for every other deployment in the token. Compare against the topology-derived expected value: bytes32(target OFT) for active hub links (origin or target is the hub), bytes32(0) for every other pair. Render the result as a single M×M matrix per token so the topology is visible at a glance, a clean hub-and-spoke shows as a populated "cross" along the hub row and column with all other cells cleared. List any non-matching cells in detail beneath the matrix (origin chain, target eid, expected bytes32, on-chain bytes32). A more thorough variant, and the one we'd recommend if scope allows, replays each chain's PeerSet(eid, peer) events from the OApp's deployment block forward and reconciles against the live peers map. This catches a peer that was set to a non-Kelp eid the script does not know about (i.e., a forgotten chain that's not in the deployments enumeration). The M×M matrix bound to the token's

	known deployments is sufficient as a first pass; event-based reconciliation is the audit-grade closure.
Comments / Resolution	Fixed (M×M matrix variant), event-replay variant acknowledged. Per-token peer matrix reads OApp.peers(eid) (or oapp_core::has_peer / get_peer on Movement) for every other deployment in the topology. Hub-spoke pairs require the padded peer OFT/Adapter address; L2-L2 pairs require bytes32(0).

Issue_05	Movement side is barely checked
Severity	High
Description	Movement (the rsETH non-EVM spoke on Aptos-derived L1) is treated as out-of-scope for everything except a single REST read of the OAppStore resource that returns admin and delegate. All other Movement-side state, peers map, ULN config (DVN set, confirmations, library / executor pinning), enforced options, is implicitly trusted to match the declarative configuration in move.layerzero.config.ts. The script's own README acknowledges this gap. The implications are non-trivial: - **Movement's peers map** is the on-chain authorisation gate for who Movement will accept messages from. If Movement still trusts a stale L2 peer from before the hub-and-spoke flatten, an inbound from that L2 reaches the rsETH OFT Move module without ever transiting Ethereum. - **Movement-side ULN config** determines the DVN set and confirmations enforced for Movement → Ethereum and Movement ← Ethereum on the Movement side. If declarative config drifts from on-chain state, security is whatever's deployed, not whatever's declared. - **Movement-side enforced options** carry the same DoS / griefing surface as the EVM side.
Recommendations	Movement exposes view functions on the same module path the script already uses for OAppStore. - Read the Movement peer map via oapp_core::has_peer(eid) and oapp_core::get_peer(eid) for every other deployment in the token, called per eid via the Aptos /v1/view REST endpoint. This is the same pattern as the existing REST integration. It closes the Movement side of the hub-and-spoke topology check (issue 8). - Read Movement-side enforced options via oapp_core::get_enforced_options(u32, u16) on the same module. Mirrors the EVM-side enforced-gas check from issue (4). - Read Movement-side ULN config (DVN set, confirmations,

	library / executor pinning) from the Movement ULN302 module (0xc33752e0220faf79e45385dd73fb28d681dcd9f1569a1480725507c1f3c3aba9). Until these checks are in place, the Movement side's hardening is asserted by config files, not verified on-chain.
Comments / Resolution	Fixed end-to-end. All three sub-recommendations implemented: - Movement peer matrix via oapp_core::has_peer / oapp_core::get_peer for every other deployment EID. - Movement-side enforced LZ_RECEIVE gas via oapp_core::get_enforced_options(eid, msgType) for both msgTypes. - Movement-side ULN config (DVN set, confirmations, optional fields) via msglib::get_app_send_config / get_app_receive_config and executor via msglib::get_app_executor_config, plus library resolution via endpoint::get_effective_send_library / get_effective_receive_library. Movement admin/delegate migrated from the OAppStore resource read to view functions. The msglib::* view-function paths are routed through the resolved library address rather than the canonical (same correctness fix as the EVM side of finding 3).

Issue_06	enforcedOptions are not read
Severity	**Medium**
Description	The hardening pins LZ_RECEIVE gas to 200,000 on both msgType 1 (SEND) and msgType 2 (SEND_AND_CALL), per the README. The script never calls OApp.enforcedOptions(eid, msgType) and never renders a corresponding cell. Under-funded LZ_RECEIVE gas is a real DoS / griefing surface, the executor delivers the message, the destination call OOGs, the message stalls; for SEND_AND_CALL the inner compose dispatch may fail. The hardening claim is asserted by the README but not verified by the script.
Recommendations	For every send-side pathway, read OApp.enforcedOptions(eid, 1) and OApp.enforcedOptions(eid, 2). Parse the canonical TYPE_3 single-LZ_RECEIVE blob (0x0003 01 0011 01 + <gas:uint128>, 22 bytes total) and assert the trailing 16 bytes decode to 200000. Flag any pathway where either msgType deviates, where the option blob is empty, or where the format does not match the expected canonical shape (an unrecognised shape may indicate stacked sub-options the verifier is not parsing).
Comments / Resolution	Fixed, followed recommendation. Both msgType = 1 and msgType = 2 are read; the decoder is strict (validates exact length 44 hex chars AND prefix 0x000301001101) and the gas value is asserted equal to 200000. Applied symmetrically on the EVM and Movement sides.

Issue_07	maxMessageSize, optionalDVNCount, and optionalDVNThreshold are read but not checked
Severity	**Medium**
Description	All three values are returned by getAppUlnConfig / executorConfigs but the verification logic discards them. The hardening claims maxMessageSize == 10000, no optional DVNs (optionalDVNCount == 0), and optionalDVNThreshold == 0. None are asserted. The optional-DVN check is the load-bearing one: a non-zero optional set with a non-zero threshold **changes the security model** , messages can verify with required + optional ≥ threshold confirmations, so the 4-of-4 required guarantee is silently weakened. This reintroduces exactly the V3 / V4 surface the hardening was meant to close, and a single-DVN compromise becomes load-bearing again.
Recommendations	On every pathway, in both directions, assert optionalDVNCount === 0 and optionalDVNThreshold === 0. Downgrade the DVN-quorum cell to ⚠ when either is non-zero, even if the required-DVN set is correct. On every send-side pathway, assert maxMessageSize === 10000 and surface it as its own column with a ✓ / ⚠ glyph.
Comments / Resolution	Fixed, followed recommendation. All three values asserted with explicit pass/fail entries on every pathway.

Issue_08	owner == delegate is printed but not asserted
Severity	**Low**
Description	The script reads OFT.owner() and EndpointV2.delegates(oapp) per chain and renders both in the contracts table. The README claims they should be the same address per chain, these are the only two callers that can invoke setSendLibrary / setReceiveLibrary / setConfig. The script does not compare them. A typo or copy-paste mistake during a multi-chain handover that splits the privileged role across two accounts would not be flagged programmatically; the auditor must eyeball each chain's table by hand. The current report has no ✓ / ⚠ on the invariant.
Recommendations	Compare the two addresses programmatically and emit an explicit ✓ / ⚠ callout in each chain section directly under the contracts table. When they differ, surface both addresses in the warning so the divergence is visible at a glance. The same check should be applied to the Movement OAppStore.admin and OAppStore.delegate fields.
Comments / Resolution	Fixed, followed recommendation. Programmatic comparison stored as ownerEqualsDelegate and aggregated into a per-token summary count. Applied to Movement via the oapp_core::get_admin / get_delegate view functions.