

LAYERZERO · SECURITY REVIEW

LayerZero Canton VER.

A security review of Canton VER.

CLIENT	PROJECT	AUDITORS	AUDIT WINDOW	FINAL REPORT
LayerZero	Canton VER	Andreas Zottl, Roberto Reigada	March 20 – May 7, 2026	May 7, 2026

§ 00

Contents.

92 pages: overview, confidence ranking, scope, 65 findings, disclaimer, and about Guardian.

§ 01	Overview	P. 03
§ 02	Confidence Ranking	P. 04
§ 03	Scope & Classifications	P. 05
	Files in Scope	P. 06
§ 04	Findings	P. 09
	Main Review 1 · 60 findings	P. 12
	Main Review 2 · 5 findings	P. 85
§ 05	Disclaimer	P. 91
§ 06	About Guardian	P. 92

§ 01

Overview.

Guardian reviewed Canton VER across 2 review rounds. The final report includes 3 Medium, 28 Low, 34 Informational findings.

AUDIT FIRM	REPOSITORY
Guardian	https://github.com/GuardianOrg/canton-ver-team1-1773774383947 , https://github.com/GuardianOrg/audit-external-team1-1777914234503
CLIENT	METHODOLOGY
LayerZero	Smart contract · Manual · Main Review 1 · Main Review 2
PROJECT	MAIN REVIEW 1 COMMIT
Canton VER	3e2c36d41b34
PREPARED BY	MAIN REVIEW 2 COMMIT
Andreas Zottl, Roberto Reigada	01d6c7d2553d
AUDIT WINDOW	
March 20 – May 7, 2026	
FINAL REPORT	
May 7, 2026	

1.1 Confidence Ranking

5 / 5

Very High Confidence

Guardian assigns a Confidence Ranking of 5 out of 5, combining finding severity, remediation outcome, and the reviewer’s assessment of latent risk. The full scale is on the next page.

1	2	3	4	5
VERY LOW CONFIDENCE	LOW CONFIDENCE	MODERATE CONFIDENCE	HIGH CONFIDENCE	VERY HIGH CONFIDENCE

1.2 Findings at a Glance

SEVERITY	TOTAL	PENDING	ACK	PARTIAL	RESOLVED
red circle Critical	0	0	0	0	0
orange circle High	0	0	0	0	0
yellow circle Medium	3	3	0	0	0
blue circle Low	28	28	0	0	0
grey circle Informational	34	34	0	0	0
All severities	65	65	0	0	0

§ 02

Confidence Ranking.

Guardian assigns a project-level confidence score using the scale below. The ranking combines finding severity, remediation outcome, and the reviewer's assessment of latent risk.

CONFIDENCE RANKING	DEFINITION AND RECOMMENDATION	RISK PROFILE
5 Very High Confidence ASSIGNED	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. Recommendation: code is highly secure at time of audit.	0 High/Critical findings and few Low/Medium severity findings.
4 High Confidence	Code is clean, well-structured, and adheres to best practices. Design patterns are sound and test coverage is strong. Recommendation: suitable for deployment after remediations; consider periodic review with changes.	0-1 High/Critical findings per engagement week and little to no Medium severity findings.
3 Moderate Confidence	Code is functional but shows gaps in defensive programming or test coverage. Recommendation: remediate the findings and re-review the changes before deployment.	Some High/Critical findings, with various Medium severity findings.
2 Low Confidence	Recurring issues suggest systemic gaps rather than isolated mistakes. Recommendation: remediate, then commission a follow-up review at a frozen commit.	Over 6 High and Critical findings across the engagement.
1 Very Low Confidence	The volume and severity of findings indicate the codebase is not ready for production. Recommendation: substantial rework, then a full re-review.	Over 12 High and Critical findings across the engagement.

§ 03

Scope & Classifications.

Guardian reviewed Canton VER in <https://github.com/Guardian0rg/canton-ver-team1-1773774383947>, <https://github.com/Guardian0rg/audit-external-team1-1777914234503>. The scope includes 10,509 source lines across 103 files.

3.1 Severity Matrix

	IMP · HIGH	IMP · MED	IMP · LOW
LIK · HIGH	Critical	High	Medium
LIK · MED	High	Medium	Low
LIK · LOW	Medium	Low	Low

IMPACT · HIGH

Significant loss of assets, harm to a group of users, or core protocol functionality is disrupted.

IMPACT · MEDIUM

Small amount of funds can be lost, or ancillary functionality is affected. Users may experience reduced or delayed receipt.

IMPACT · LOW

Unexpected behavior in some functionality that is notable but does not meet the criteria for higher severity.

LIKELIHOOD · HIGH

Possible with reasonable assumptions that mimic on-chain conditions; cost is low compared to amount gained.

LIKELIHOOD · MEDIUM

Possible only in uncommon cases or requires a large amount of capital relative to disruption caused.

LIKELIHOOD · LOW

Unlikely to ever occur in production.

Files in Scope 1/3.

FILE	NSLOC	TOTAL
packages/api-common/src/container/component.ts	211	335
packages/api-common/src/container/components/blocked-message-library.ts	22	35
packages/api-common/src/container/components/endpoint.ts	38	50
packages/api-common/src/container/components/uln-302.ts	61	75
packages/api-common/src/container/container.ts	127	236
packages/api-common/src/schemas/contract.ts	69	174
packages/api-common/src/schemas/transaction.ts	37	100
packages/api-common/src/schemas/vapp.ts	8	27
packages/api-common/src/schemas/vapps/blocked-message-library.ts	20	22
packages/api-common/src/schemas/vapps/endpoint.ts	282	288
packages/api-common/src/schemas/vapps/message-library.ts	64	67
packages/api-common/src/schemas/vapps/treasury.ts	65	66
packages/api-common/src/schemas/vapps/uln-302.ts	480	492
packages/common-utils/src/byte-codec.ts	184	260
packages/common-utils/src/utils/hash.ts	5	18
packages/common-utils/src/utils/hex.ts	19	30
packages/endpoint/src/common/guid.ts	19	20
packages/endpoint/src/common/packet-v1-codec/error.ts	5	5
packages/endpoint/src/common/packet-v1-codec/index.ts	136	170
packages/endpoint/src/dvn/dvn-options.ts	145	215
packages/endpoint/src/dvn/dvn.ts	424	523
packages/endpoint/src/dvn/error.ts	40	54
packages/endpoint/src/dvn/event.ts	24	27
packages/endpoint/src/dvn/fee-lib.ts	90	113
packages/endpoint/src/dvn/multisig/error.ts	50	62
packages/endpoint/src/dvn/multisig/event.ts	37	43
packages/endpoint/src/dvn/multisig/hash-builder.ts	18	26
packages/endpoint/src/dvn/multisig/hashes.ts	106	118
packages/endpoint/src/dvn/multisig/multisig.ts	9	20
packages/endpoint/src/endpoint-base/index.ts	40	47
packages/endpoint/src/endpoint-v2/error.ts	16	23
packages/endpoint/src/endpoint-v2/event.ts	138	153
packages/endpoint/src/endpoint-v2/index.ts	559	634
packages/endpoint/src/executor/error.ts	27	49
packages/endpoint/src/executor/event.ts	24	27
packages/endpoint/src/executor/executor-options.ts	99	157

Files in Scope 2/3.

FILE	NSLOC	TOTAL
packages/endpoint/src/executor/executor.ts	176	201
packages/endpoint/src/executor/fee-lib.ts	166	184
packages/endpoint/src/fee-lib-base.ts	36	43
packages/endpoint/src/message-library-manager/error.ts	51	61
packages/endpoint/src/message-library-manager/event.ts	144	172
packages/endpoint/src/message-library-manager/index.ts	538	667
packages/endpoint/src/message-library/blocked-message-library.ts	96	112
packages/endpoint/src/message-library/uln/uln-302/errors.ts	101	122
packages/endpoint/src/message-library/uln/uln-302/events.ts	246	292
packages/endpoint/src/message-library/uln/uln-302/types/executor-config.ts	22	39
packages/endpoint/src/message-library/uln/uln-302/types/uln-config.ts	118	206
packages/endpoint/src/message-library/uln/uln-302/uln-302.ts	1,104	1,384
packages/endpoint/src/message-library/uln/uln-302/uln-options.ts	127	162
packages/endpoint/src/messaging-channel/error.ts	12	15
packages/endpoint/src/messaging-channel/event.ts	99	111
packages/endpoint/src/messaging-channel/index.ts	304	364
packages/endpoint/src/price-feed/errors.ts	28	32
packages/endpoint/src/price-feed/price-feed.ts	309	388
packages/endpoint/src/transaction.ts	6	7
packages/endpoint/src/treasury/errors.ts	5	5
packages/endpoint/src/treasury/events.ts	32	38
packages/endpoint/src/treasury/treasury.ts	96	114
packages/endpoint/src/worker/acl.ts	5	14
packages/endpoint/src/worker/error.ts	8	15
packages/endpoint/src/worker/event.ts	176	212
packages/endpoint/src/worker/pausable-worker-base.ts	35	48
packages/endpoint/src/worker/worker-base.ts	220	268
packages/protocol-common/src/common/address.ts	14	22
packages/protocol-common/src/common/options-base.ts	9	23
packages/protocol-common/src/context.ts	86	123
packages/protocol-common/src/contract-component.ts	7	8
packages/protocol-common/src/contract-like.ts	29	45
packages/protocol-common/src/contract.ts	21	25
packages/protocol-common/src/event.ts	20	30
packages/protocol-common/src/ownable.ts	40	50
packages/protocol-common/src/payable.ts	38	60

Files in Scope 3/3.

FILE	NSLOC	TOTAL
packages/ver-merkle/src/contract/address.ts	20	38
packages/ver-merkle/src/contract/array.ts	74	128
packages/ver-merkle/src/contract/base.ts	20	42
packages/ver-merkle/src/contract/big-uint.ts	20	38
packages/ver-merkle/src/contract/boolean.ts	20	38
packages/ver-merkle/src/contract/codec.ts	163	216
packages/ver-merkle/src/contract/enumerable-set.ts	55	93
packages/ver-merkle/src/contract/map.ts	55	98
packages/ver-merkle/src/contract/set.ts	39	71
packages/ver-merkle/src/contract/string.ts	20	38
packages/ver-merkle/src/contract/uint8-array.ts	20	38
packages/ver-merkle/src/slots/slot-allocator.ts	14	32
packages/ver-merkle/src/slots/slot-derivation.ts	28	85
packages/ver-types/src/abi/primitives.ts	11	12
packages/ver-types/src/abi/schema-builder.ts	55	61
packages/ver-types/src/abi/types.ts	26	47
packages/ver-types/src/primitives/schemas.ts	44	56
packages/ver-types/src/primitives/types.ts	4	9
packages/ver-types/src/schemas/common.ts	64	112
packages/ver-types/src/schemas/endpoint/endpoint-v2.ts	245	346
packages/ver-types/src/schemas/endpoint/message-library-manager.ts	212	320
packages/ver-types/src/schemas/endpoint/messaging-channel.ts	99	144
packages/ver-types/src/schemas/message-library/message-library.ts	102	157
packages/ver-types/src/schemas/message-library/uIn-302.ts	303	407
packages/ver-types/src/schemas/transaction.ts	10	14
packages/ver-types/src/schemas/workers/dvn.ts	177	249
packages/ver-types/src/schemas/workers/executor.ts	81	113
packages/ver-types/src/schemas/workers/price-feed.ts	142	233
packages/ver-types/src/schemas/workers/treasury.ts	38	67
packages/ver-types/src/schemas/workers/worker-access-control.ts	36	61
packages/ver-types/src/schemas/workers/worker.ts	90	131

§ 04

Findings.

Grouped by round. Visible IDs reset by severity within each round, so a finding is named the same way in the report as it was in the channel it was written in. Status is where each finding ended up; its own page says how.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
MAIN REVIEW 1					
M-01	Nonce-As-Block-Number Semantic Mismatch	Censoring	MEDIUM	PENDING	p. 12
M-02	Worker address front-running	Frontrunning	MEDIUM	PENDING	p. 14
L-01	Canonical pre-state verification for writes	Best Practices	LOW	PENDING	p. 15
L-02	Recovery of Validator Execution State	Best Practices	LOW	PENDING	p. 16
L-03	Implicit Single-Use Context Assumption	Validation	LOW	PENDING	p. 17
L-04	PriceFeed stores Arbitrum fee params globally	Configuration	LOW	PENDING	p. 18
L-05	resolveMethod() shadows same-family duplicates	Logical Error	LOW	PENDING	p. 19
L-06	finishSend() is not bound by send continuation	Configuration	LOW	PENDING	p. 20
L-07	Read enforcement applies to the top-level only	Unexpected Behavior	LOW	PENDING	p. 22
L-08	Unbounded Execution Can Halt Validators	Validation	LOW	PENDING	p. 23
L-09	DVN Duplicate Check Can Be Bypassed	Validation	LOW	PENDING	p. 24
L-10	Duplicate Contract Addresses Are Possible	Validation	LOW	PENDING	p. 25
L-11	Comment misstates nextMsgSender semantics	Best Practices	LOW	PENDING	p. 26
L-12	Weaker type match for method detection	Best Practices	LOW	PENDING	p. 27
L-13	DVN signatures are not bound to a specific DVN	Signatures	LOW	PENDING	p. 28
L-14	PacketV1 silently truncates oversized fields	Validation	LOW	PENDING	p. 29
L-15	Dvn exposes admin-only ACL and library setters	Trust Assumptions	LOW	PENDING	p. 30
L-16	Same DVN can satisfy two ULN quorum slots	Validation	LOW	PENDING	p. 31
L-17	Library registration accepts mismatched names	Validation	LOW	PENDING	p. 32
L-18	Receive support check ignores DVN registration	Validation	LOW	PENDING	p. 33
L-19	Event commitment excludes event type/name	Validation	LOW	PENDING	p. 34
L-20	PriceFeed uses different EID key spaces	Unexpected Behavior	LOW	PENDING	p. 35
L-21	Canton receive prices native options it ignores	Unexpected Behavior	LOW	PENDING	p. 37

Findings continued.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
L-22	Odd-length signer input can wedge DVN quorum	Unexpected Behavior	LOW	PENDING	p. 38
L-23	Dynamic mapping keys use non-Solidity slots	Unexpected Behavior	LOW	PENDING	p. 39
L-24	Mixed-case bytes32 hashes break matching	Validation	LOW	PENDING	p. 40
L-25	Ownable admin rotation is not exposed	Unexpected Behavior	LOW	PENDING	p. 42
L-26	Supported EIDs can still fail quote and send	Validation	LOW	PENDING	p. 43
I-01	Generic non-OApp Request_Create can reach unauthenticated ULN worker-registration methods	Access Control	INFORMATIONAL	PENDING	p. 44
I-02	Unused Error	Superfluous Code	INFORMATIONAL	PENDING	p. 46
I-03	Sender-switching does not apply to constructors	Informational	INFORMATIONAL	PENDING	p. 47
I-04	Storage compatibility for composite values	Documentation	INFORMATIONAL	PENDING	p. 48
I-05	Cross-component calls use continuation handoff semantics rather than synchronous call/return	Documentation	INFORMATIONAL	PENDING	p. 49
I-06	Slot isolation depends on sparse address spacing	Configuration	INFORMATIONAL	PENDING	p. 51
I-07	TS lzReceive is pull-based, unlike EVM	Documentation	INFORMATIONAL	PENDING	p. 52
I-08	DVN governance does not have a sequencing nonce	Signatures	INFORMATIONAL	PENDING	p. 53
I-09	Inconsistent malformed-argument handling	Error	INFORMATIONAL	PENDING	p. 54
I-10	Executor fees price only message bytes	Unexpected Behavior	INFORMATIONAL	PENDING	p. 55
I-11	DVN library revocation does not affect verify	Warning	INFORMATIONAL	PENDING	p. 56
I-12	TS_AUDIT describes outdated container semantics	Documentation	INFORMATIONAL	PENDING	p. 57
I-13	Fallback To Default Drops OApp Timeout Grace	Documentation	INFORMATIONAL	PENDING	p. 58
I-14	OApp ULN override can set zero confirmations	Validation	INFORMATIONAL	PENDING	p. 59
I-15	Nilified packets can become executable again	Validation	INFORMATIONAL	PENDING	p. 61
I-16	lzReceiveAlert() emits spoofable alerts	Warning	INFORMATIONAL	PENDING	p. 63
I-17	Config getters are unreachable via public reads	Warning	INFORMATIONAL	PENDING	p. 64
I-18	quorumChangeAdmin() drops the DVN admin check	Documentation	INFORMATIONAL	PENDING	p. 65
I-19	Default ULN zero-confirmation parity mismatch	Warning	INFORMATIONAL	PENDING	p. 67
I-20	Interleaved DVN options make regrouping costly	Warning	INFORMATIONAL	PENDING	p. 68
I-21	Unbounded payee strings	DoS	INFORMATIONAL	PENDING	p. 70

Findings continued.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
I-22	load(...) must preserve layout across TS upgrade	Informational	INFORMATIONAL	PENDING	p. 72
I-23	Receive path is not atomic due to callbacks	Informational	INFORMATIONAL	PENDING	p. 73
I-24	DAML Receive Callback and msg.value Forwarding	Documentation	INFORMATIONAL	PENDING	p. 75
I-25	No fee-based prioritization under load	Gaming	INFORMATIONAL	PENDING	p. 76
I-26	Skipped & Burnt Packets Are Reported As Executed	Unexpected Behavior	INFORMATIONAL	PENDING	p. 77
I-27	SupportedOptionTypes Is Unused	Superfluous Code	INFORMATIONAL	PENDING	p. 78
I-28	Unused LZ Token Fee Surface Remains Exposed	Informational	INFORMATIONAL	PENDING	p. 79
I-29	Stale startSend references in comments	Documentation	INFORMATIONAL	PENDING	p. 80
I-30	Consumed Nonce Marks Arbitrary Payloads Verified	Unexpected Behavior	INFORMATIONAL	PENDING	p. 81
I-31	Endpoint API exports unusable contract names	Unexpected Behavior	INFORMATIONAL	PENDING	p. 82
I-32	Mutation Testing Results	Informational	INFORMATIONAL	PENDING	p. 83

MAIN REVIEW 2

M-01	Operations Remain Callable While EID Is 0	Validation	MEDIUM	PENDING	p. 85
L-01	Inbound nonce checks can scan unbounded backlogs	Validation	LOW	PENDING	p. 86
L-02	ULN And Endpoint EIDs Can Diverge	Validation	LOW	PENDING	p. 88
I-01	DVN signatures omit deployment domain	Signatures	INFORMATIONAL	PENDING	p. 89
I-02	ULN Exposes setEid But No getEid Method	Informational	INFORMATIONAL	PENDING	p. 90

MEDIUM **M-01**

PENDING

Nonce-As-Block-Number Semantic Mismatch

CATEGORY Censoring
LOCATION Global
REVIEW Main Review 1

DESCRIPTION

The system models `Context.blockNumber` using the container's global write nonce instead of a true host-chain block number. As a result, block-number-based logic advances with write request throughput, rather than actual blockchain progression or elapsed time.

In a real blockchain:

Multiple transactions can be included in a single block without increasing `block.number`.

Block numbers only advance when a new block is produced

Block cadence introduces a natural time constraint.

In this implementation:

Each processed write request increments the modeled `blockNumber`

Failed writes also increment the counter

Rapid consecutive writes can significantly advance `blockNumber`

No real-time or block-production dependency exists

Effectively, the system behaves as:

```
blockNumber := number of processed writes
```

instead of:

```
blockNumber := number of host-chain blocks
```

Root Cause

The validator constructs the execution context as:

```
blockNumber = await container.nonce(state)
```

Meanwhile, the container increments the nonce for every write call prior to execution. This creates a direct coupling between request volume and the perceived block number.

Protocol Impact

Components such as `MessageLibraryManager` rely on `context.blockNumber` for timeout and grace-period logic:

Library validity windows depend on `timeout.expiry > context.blockNumber`

Grace periods are defined as:

```
expiry = context.blockNumber + gracePeriod
```

Timeout updates are validated against the current `blockNumber`

This leads to:

1. Premature expiration of grace periods

M-01 Nonce-As-Block-Number Semantic Mismatch

CONTINUED

2. Reduced safety margins during library migration
3. Distortion of time-based assumptions due to unrelated traffic
4. Potential disruption of in-flight message processing
5. Security & Operational Implications

While not directly enabling fund theft, this issue introduces liveness and reliability risks:

Attackers (or high traffic) can accelerate timeout windows

Library rotation processes may fail or behave unpredictably

Systems relying on block-based timing assumptions may degrade under load

The severity depends on:

Reliance on block-number-based timing in production

Cost and accessibility of generating write requests

Assumptions made by operators about timing guarantees

— RECOMMENDATION

Do not derive `Context.blockNumber` from container nonce.

Safer options:

1. use an actual host-chain block number if the host chain exposes one
2. if no true block number exists, use explicit timestamps for timeout logic instead of a synthetic nonce
3. if a synthetic counter must exist, do not use it for security-sensitive timeout semantics that assume block-cadence behavior

— RESOLUTION

Status: Pending.

MEDIUM **M-02**

PENDING

Worker address front-running

CATEGORY	Frontrunning
LOCATION	https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/36070d8a5d27ef416f0b864964c56d00aec6fa69/packages/endpoint/src/message-library/uln/uln-302/uln-302.ts#L1438-L1439
REVIEW	Main Review 1

— DESCRIPTION

`Uln302.registerDvn()`, `registerExecutor()`, and `registerPriceFeed()` derive new worker addresses from a shared monotonic nonce and a fixed worker-type hint for all senders.

Because these registration functions are externally callable without admin authorization, any caller can consume future worker addresses before a legitimate operator does.

This creates a front-running risk for integrations that precompute a worker address off-chain and then later configure it.

For example, if an operator expects the next DVN address to be `deriveGlobalAddress("dvn-N", ...)` and prepares a later configuration transaction referencing that address, and submits two transactions—one with `registerDvn()` and one with `setConfig()`—an attacker can front-run with their own `registerDvn()` call first.

For example: Safe Transactions by USDT0 Team.

The attacker then receives the exact address the operator expected, while the legitimate DVN is created at the next nonce.

The later configuration transaction can therefore end up pointing to an attacker-controlled worker.

— RECOMMENDATION

Consider making derived worker addresses incorporate caller-specific entropy, such as the registering sender's address, rather than relying only on a shared global nonce and a fixed type hint.

This would prevent one caller from stealing another caller's predicted worker address, even if registration remains externally accessible.

— RESOLUTION

Status: Pending.

LOW **L-01**

PENDING

Canonical pre-state verification for writes

CATEGORY	Best Practices
LOCATION	packages/api-common/src/controllers/validator.ts:135
REVIEW	Main Review 1

— DESCRIPTION

Validator write execution currently trusts the `stateRoot` supplied in the write request and does not independently verify that it matches the latest committed root before executing and signing the next transition.

In `packages/api-common/src/controllers/validator.ts`, `#write()` directly executes the request against the caller-provided `stateRoot`:

- `packages/api-common/src/controllers/validator.ts:135`
- `packages/api-common/src/controllers/validator.ts:147`

The intended production flow appears to rely on the sequencer to provide the correct latest root:

- `packages/api-common/src/services/request-processor.ts:42`

Under that model, correctness of the predecessor state is enforced by the sequencer/request processor rather than by each validator independently.

If the sequencer is correct and validator write endpoints are not externally reachable, this is primarily a defense-in-depth weakness rather than a direct exploit. However, validator correctness depends on an external assumption: that the supplied pre-state root is canonical and current. Misconfiguration, stale retries, or non standard callers could cause validators to execute and sign transitions derived from a non-canonical predecessor state.

— RECOMMENDATION

Consider adding a validator-side canonical pre-state check before write execution, or otherwise enforce that validators only accept requests whose predecessor root is proven to be the current committed root. Even if the sequencer remains the primary source of truth, local enforcement would reduce trust coupling and make the system more robust against stale or malformed write requests.

— RESOLUTION

Status: Pending.

LOW **L-02**

PENDING

Recovery of Validator Execution State

CATEGORY	Best Practices
LOCATION	apps/vapp/src/server.ts:23
REVIEW	Main Review 1

— DESCRIPTION

The VER execution layer stores all protocol state in the local Merkle trie, while Canton/DAML appears to persist only the resulting stateRoot. A Merkle root is only a commitment to state, not the state itself, so a validator cannot reconstruct the executable trie contents from the root alone after losing its local trie database.

In the current validator wiring, the vapp server constructs the container with an in-memory trie, while the chain-side commitment path stores only stateRoot metadata.

As a result, if the validator process restarts and loses its local trie contents, it cannot safely resume execution from the committed stateRoot unless there is an external replay/rehydration mechanism.

If deployed with ephemeral trie storage and no recovery flow, a validator restart can leave the node unable to safely continue protocol execution from the last committed state.

This can cause operational downtime, failed retries, or inconsistent recovery behavior if execution resumes without the original trie contents.

— RECOMMENDATION

Ensure validator execution state is durably recoverable by doing at least one of the following:

- persist the trie node database using a durable backend such as Postgres or DynamoDB
- implement a replay/rehydration mechanism that can deterministically reconstruct trie contents from a known checkpoint and the full request stream
- add startup checks that prevent execution when only a root is available but the underlying trie state is missing.

— RESOLUTION

Status: Pending.

LOW **L-03**

PENDING

Implicit Single-Use Context Assumption

CATEGORY	Validation
LOCATION	https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/0517a222c9408f6fe6d5fd018ca2d346730a8638/packages/api-common/src/container/container.ts#L127-L131
REVIEW	Main Review 1

DESCRIPTION

The container read-purity check assumes that the supplied `Context` is fresh, but that invariant is not enforced by the runtime API itself.

`Container.call(..., write = false)` rejects whenever the passed `Context` already contains events or payouts, but does not enforce this state.

In the current production flow this does not appear to be exploitable: the validator allocates a brand-new `Context` for every read request and for every write request before calling the container, so the sequencer/validator path does not currently reuse a dirty context.

However, this safety property is implicit rather than enforced. Any future direct container consumer, refactor, helper, or test harness that reuses a `Context` across calls could trigger false `Side effects in read-only call` failures.

RECOMMENDATION

Consider to implement one of these changes:

- Document that `Context` instances are one-shot and must not be reused across top-level container calls
- Update the read-purity check to snapshot the initial event/payout counts and compare deltas instead of absolute sizes
- Revert at the beginning of the function if the `Context` is not fresh

RESOLUTION

Status: Pending.

LOW **L-04**

PENDING

PriceFeed stores Arbitrum fee params globally

CATEGORY	Configuration
LOCATION	packages/endpoint/src/price-feed/price-feed.ts:196
REVIEW	Main Review 1

— DESCRIPTION

PriceFeed applies a dedicated fee model for Arbitrum-style chains because that model requires extra parameters beyond the default gas-price inputs, including `gasPerL2Tx` and `gasPerL1CallDataByte`. Although the setter API is EID-scoped, the implementation stores the Arbitrum extension data in a single shared record rather than under the supplied destination EID.

If multiple Arbitrum-model destinations use different extension parameters, the most recent `setPriceForArbitrum()` call overwrites the shared extension used by all of them. Updating chain A can therefore silently change fee estimation for chain B.

— RECOMMENDATION

If there is a possibility of multiple Arbitrum-style chains (orbit chains), being supported by LayerZero, consider storing `ArbitrumPriceExt` per destination EID and updating the getter and fee-estimation paths to read the extension for the specific destination being quoted.

— RESOLUTION

Status: Pending.

LOW **L-05**

PENDING

resolveMethod() shadows same-family duplicates

CATEGORY	Logical Error
LOCATION	packages/api-common/src/container/component.ts:286
REVIEW	Main Review 1

— DESCRIPTION

`ContainerComponent.#resolveMethod()` resolves callable methods by bare function name and returns the first match found among the loaded contract and its sub-components. It does not distinguish overloaded signatures and does not reject ambiguous same-family method names.

The resolution order is:

- top-level contract
- then each entry from `contract.getComponents()`

and the first match wins. This means if a top-level contract and one of its sub-components expose the same method name in the same read/write category, one implementation silently shadows the other based solely on component ordering.

We confirmed one current same-family duplicate in the EndpointV2 family:

- `EndpointV2.getId()`
- `MessagingChannel.getId()`

Under current ordering, `EndpointV2.getId()` wins first. This appears benign today because the top-level method delegates to `MessagingChannel`, but the behavior is still an architectural footgun.

Future duplicate names could change reachability silently without any ambiguity error.

— RECOMMENDATION

Consider rejecting ambiguous same-family method names during API construction or method resolution. If overload-like behavior is ever needed, resolve by a full typed signature rather than function name alone.

— RESOLUTION

Status: Pending.

LOW **L-06**

PENDING

finishSend() is not bound by send continuation

CATEGORY Configuration

LOCATION <https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/36070d8a5d27ef416f0b864964c56d00aec6fa69/packages/endpoint/src/endpoint-v2/endpoint-v2.ts#L259-L260>

REVIEW Main Review 1

DESCRIPTION

`EndpointV2.send()` returns a continuation to the configured message library, and that library later returns a continuation back into `EndpointV2.finishSend()`.

The final hop is accepted in
`packages/endpoint/src/endpoint-v2/endpoint-v2.ts:252`
if:

- `msg.sender` is a registered library
- `msg.sender === state.library`

That means `finishSend()` trusts the runtime continuation caller plus the carried `state.library`, but it does not prove that this `finishSend()` is the unique successor of a specific earlier `EndpointV2.send()` invocation.

There is no explicit continuation/session nonce or unforgeable handoff token binding the two hops together.

In the intended `Uln302` path this is mitigated because `uln-302.ts:306` requires the caller of `Uln302.send()` to be the Endpoint, so an external caller cannot normally fabricate the middle hop.

But structurally, the safety of the chain depends on every reachable library implementation enforcing that same invariant. A malicious or incorrectly exposed library could manufacture a continuation into `finishSend()`.

This is different from a normal EVM call stack, where the intermediate call chain is immediate and execution-context coupling is stronger. In this runtime, continuation ordering is reconstructed by the container across separate redispatch steps, so the trust boundary is more explicit.

RECOMMENDATION

L-06 finishSend() is not bound by send continuation

CONTINUED

Consider adding a continuation/session binding, for example a token derived from the originating `send()` context and checked in `finishSend()`, so the final hop is tied to a specific initiated flow rather than only to `state.library` and caller identity.

Even if you do not change the design, this should at least be documented as a security assumption: every externally reachable library `send()` must enforce endpoint-only access.

— RESOLUTION

Status: Pending.

LOW **L-07**

PENDING

Read enforcement applies to the top-level only

CATEGORY	Unexpected Behavior
LOCATION	Global
REVIEW	Main Review 1

— DESCRIPTION

The runtime enforces read-only behavior only at the end of a top-level `Container.call(..., write=false)` by checking whether final state, events, or payouts changed.

This means the read/write split in the vApp schema is an external dispatch boundary, not a per-hop or per-method purity guarantee. Inside a top-level write flow, nested continuation hops inherit the same `write=true` mode, and direct intra-

component proxy calls do not go back through the schema dispatcher.

As a result, a method that is externally categorized as read-only can still mutate state when it is invoked internally during a larger write path without triggering the container's read-only protection.

— RECOMMENDATION

If stronger guarantees are desired, consider adding an explicit read-only mode to propagated execution context and enforcing it on continuation hops and/or direct proxy-mediated internal calls.

— RESOLUTION

Status: Pending.

LOW **L-08**

PENDING

Unbounded Execution Can Halt Validators

CATEGORY	Validation
LOCATION	https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/36070d8a5d27ef416f0b864964c56d00aec6fa69/packages/api-common/src/container/container.ts#L162-L186
REVIEW	Main Review 1

DESCRIPTION

As the system misses gas like computation constraints it is vulnerable to wasting computation power because of malicious requests and could run into endless loops.

Request creation on the DAML/Canton side does not appear to price requests based on the cost of the TS execution they will trigger.

In `RequestFactory.daml:73` request

creation validates only that the supplied fee satisfies the configured minimum (`config.minFee`) before transferring the fee amount to treasury and creating the request.

As a result, request-layer cost is largely decoupled from TS runtime cost. If an externally reachable request path can be made unusually expensive in CPU, memory, or serialization work, an attacker may be able to impose disproportionate load on validators at near-flat request cost.

For example the continuation loop is created with `while (true)` and it will go on for ever as long as a continuation is returned by the called contract. This enables a potential endless loop DoS vector for future versions of the codebase.

RECOMMENDATION

Beaware of this consideration and consider adding gas like logic to prevent such cases.

RESOLUTION

Status: Pending.

LOW **L-09**

PENDING

DVN Duplicate Check Can Be Bypassed

CATEGORY Validation

LOCATION <https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/36070d8a5d27ef416f0b864964c56d00aec6fa69/packages/endpoint/src/message-library/uIn/uIn-302/types/uIn-config.ts#L129>

REVIEW Main Review 1

DESCRIPTION

The `assertSortedNoDuplicates` function compares raw strings. Therefore a config containing the same DVN twice with different hex casing passes validation, is stored, and decodes back into two identical addresses. The verification loop then iterates both entries, so one DVN verification can satisfy multiple positions in `requiredDvns` or `optionalDvns`.

In `setSendConfig` and `setReceiveConfig` the `decodeUInConfig` function is called before to get rid of this issue but this does not happen in the set default functions.

RECOMMENDATION

Canonicalize all address inputs before validation and then perform duplicate detection.

RESOLUTION

Status: Pending.

LOW **L-10**

PENDING

Duplicate Contract Addresses Are Possible

CATEGORY Validation
LOCATION GLOBAL
REVIEW Main Review 1

— DESCRIPTION

Each component registers its contracts into a local contracts map during `initializeContracts()`, but there is no global uniqueness check across components. The container later resolves an address by probing every component and taking the first match via `findIndex(Boolean)`. If two components ever register the same address, the later one becomes unreachable and execution is silently routed to the first matching component instead of failing fast. This is a correctness and safety issue for future deployments, configuration mistakes, or address collisions.

— RECOMMENDATION

Consider to enforce contract-address uniqueness.

— RESOLUTION

Status: Pending.

LOW **L-11**

PENDING

Comment misstates nextMsgSender semantics

CATEGORY Best Practices

LOCATION <https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/0ccb36e3c7eb6f15b1adb349275af40323fe17bd/packages/protocol-common/src/contract-like.ts#L33-L34>

REVIEW Main Review 1

— DESCRIPTION

The live Context constructor initializes nextMsgSender to the same value as msgSender, not to 0x0 or a meaningless placeholder. In packages/protocol-common/src/context.ts, the constructor sets:

```
``plain text
this.#msgSender = msgSender;
this.#nextMsgSender = msgSender;
``
```

However, packages/protocol-common/src/contract-like.ts still documents that the initial value of nextMsgSender is "meaningless and set to 0x0 in practice." That is no longer true in the current implementation.

This is not a direct correctness issue because runtime behavior follows the constructor, not the comment.

— RECOMMENDATION

Consider updating the comment

— RESOLUTION

Status: Pending.

LOW **L-12**

PENDING

Weaker type match for method detection

CATEGORY	Best Practices
LOCATION	https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/0ccb36e3c7eb6f15b1adb349275af40323fe17bd/packages/protocol-common/src/contract-like.ts#L15-L16
REVIEW	Main Review 1

— DESCRIPTION

The proxy wrapper in `contract-like.ts` decides whether to apply Context / msg.sender rewriting based on:

```
``plain text
if (!(value instanceof Function) || property === 'constructor') {
  return value;
}
```

In normal same-realm Node/TypeScript execution this behaves as expected for class methods, async methods, and arrow-function fields. However, `instanceof Function` is not the most robust callable check in JavaScript and can misclassify cross-realm or otherwise exotic callable values. In such cases, a callable property could bypass the wrapper logic and avoid the intended `switchMsgSender(...)` behavior.

— RECOMMENDATION

Consider replacing the check with `typeof value !== 'function'` to reduce the edge-case risk.

— RESOLUTION

Status: Pending.

LOW **L-13**

PENDING

DVN signatures are not bound to a specific DVN

CATEGORY	Signatures
LOCATION	https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/0ccb36e3c7eb6f15b1adb349275af40323fe17bd/packages/endpoint/src/dvn/dvn.ts#L206-L207
REVIEW	Main Review 1

DESCRIPTION

`hashVerify(...)` signs the packet header, payload hash, confirmations, target ULN address, vid, and expiration, but not the DVN contract address, in `hashes.ts:102`. `Dvn.verify()` then validates those signatures and forwards the attestation into `Uln302.verify()` in `dvn.ts:186`. If multiple DVNs are configured with the same signer set, quorum, vid, and linked ULN target, the same signature bundle can be reused across them. This does not let signatures cross ULN targets, but it does mean verification signatures are domain-separated by message library rather than by individual DVN identity.

RECOMMENDATION

If the protocol expects separate DVN addresses to represent independently authorized verifier sets, binding the DVN address into the signed payload would make that assumption explicit.

RESOLUTION

Status: Pending.

LOW **L-14**

PENDING

PacketV1 silently truncates oversized fields

CATEGORY Validation

LOCATION packages/endpoint/src/common/packet-v1-codec/index.ts:63

REVIEW Main Review 1

DESCRIPTION

`PacketV1Codec.encodePacketHeader()` silently truncates oversized nonce, srcEid, and dstEid values instead of rejecting them. The encoder uses `BigInt.asUintN(64, ...)` for nonce and `BigInt.asUintN(32, ...)` for both EIDs.

This wraps values modulo the target bit width.

At the same time, the ABI/Zod layer does not enforce uint32/uint64 width, only non-negativity, so out-of-range values can survive request decoding and reach the codec.

The intended `EndpointV2.send/quote` path is partially protected because `generateGuid(...)` validates uint64/uint32 bounds first. However, the message-library APIs expose direct send/quote entry points.

RECOMMENDATION

Consider adding explicit range checks in `PacketV1Codec.encodePacketHeader()` and reject values outside uint64 / uint32 bounds instead of truncating. As defense in depth, also enforce ABI-width limits during schema decoding for uint32 and uint64 rather than treating all uintN as generic non-negative bigints.

RESOLUTION

Status: Pending.

LOW **L-15**

PENDING

Dvn exposes admin-only ACL and library setters

CATEGORY Trust Assumptions

LOCATION uln-302.ts

REVIEW Main Review 1

— DESCRIPTION

`uln-302.ts` builds the `Dvn` write API by spreading `workerBase[MethodType.WRITE]` into the `Dvn` write set. That makes the inherited `setMessageLibrary`, `setAllowlisted` and `setDenylist` methods callable next to the quorum-signed `setDvnMessageLibrary`, `setAllowlist` and `setDenylist` methods. This matters because the inherited methods do not enforce the DVN signature checks. `Dvn.create()` disables `roleAdmin` by setting it to zero, but `WorkerBase` still lets any configured `admin` call those inherited setters through `#assertRoleAdminOrAdmin()`. Consequently, an admin can update the DVN allowlist, denylist, or message-library set without `vid`, expiration, quorum signatures or replay consumption. This does not let an outsider take over the DVN, but it does weaken the documented governance model and makes those protections optional in practice.

— RECOMMENDATION

Remove `setMessageLibrary`, `setAllowlisted` and `setDenylist` from the exported `Dvn` API. If those operations are meant to stay callable on `Dvn`, override them so they use the same checks as `setDvnMessageLibrary`, `setAllowlist` and `setDenylist`.

— RESOLUTION

Status: Pending.

LOW **L-16**

PENDING

Same DVN can satisfy two ULN quorum slots

CATEGORY Validation
LOCATION uln-config.ts
REVIEW Main Review 1

— DESCRIPTION

`assertNoDuplicateDvns()` only checks sorting and duplicates inside `requiredDvns` and inside `optionalDvns`. It never rejects overlap between the two arrays. `resolveUlnConfig()` later merges defaults with OApp overrides and only calls `assertValidStandaloneConfig()`, so an overlap introduced by a partial override also survives.

This weakens the effective verification quorum. `Uln302.#checkVerifiable()` first requires every DVN in `requiredDvns`, then iterates `optionalDvns` and decrements `optionalDvnThreshold` for each verified DVN. Therefore the same DVN can be counted once as required and again as optional. For example, `requiredDvns = [A]`, `optionalDvns = [A]` and `optionalDvnThreshold = 1` become verifiable after a single attestation from `A`, even though the config appears to require two independent checks.

— RECOMMENDATION

Reject any address that appears in both DVN arrays. The smallest safe change is to extend `assertNoDuplicateDvns()` with a cross-array overlap check, then run that validation again after `resolveUlnConfig()` builds the effective config.

— RESOLUTION

Status: Pending.

LOW **L-17**

PENDING

Library registration accepts mismatched names

CATEGORY Validation
LOCATION index.ts
REVIEW Main Review 1

DESCRIPTION

`registerLibrary()` stores any non-empty `name` string for a library address. It does not verify that the address is a deployed message library or that the supplied name matches the contract's canonical name. That stored name is later reused as `transaction.functionSignature.contract` by `getLibraryName()`. `ContainerComponent.#resolveContract()` then checks that the requested contract name exactly matches the canonical name stored in the contracts map for that address. Therefore, if an admin registers a real library address with a typo or alternate name, later manager-generated calls to that library revert with `ContractNotFoundError` even though the address itself is correct. This can leave the library unusable for manager-mediated dispatch such as `send`, `quote`, `config` continuations and verification-state callbacks. The misregistration is sticky because there is no rename or unregister function and the same address cannot be registered again.

RECOMMENDATION

Do not trust a user-supplied library name. `registerLibrary()` should verify that the target address resolves to a deployed contract and that the supplied name matches the contract's canonical `getName()` result. A safer change is to stop accepting `name` as input and derive the canonical contract name directly from the container's contract registry.

RESOLUTION

Status: Pending.

LOW **L-18**

PENDING

Receive support check ignores DVN registration

CATEGORY Validation
LOCATION uln-302.ts
REVIEW Main Review 1

— DESCRIPTION

`Uln302.#isSupportedReceiveEid()` returns `true` as soon as the default receive ULN config exists and names at least one DVN. It does not check whether those DVN addresses are actually registered with the library.

That mismatch is reachable because `MessageLibraryManager` relies on the library's `assertSupportedEid()` continuation when a default receive library is selected. A route can therefore be accepted as supported even though its configured receive quorum is impossible to satisfy.

`Uln302.verify()` only records attestations from callers present in `#dvns` and `commitVerification()` later waits for the configured DVNs through `#checkVerifiable()`. If the receive config depends on an unregistered required DVN, or on enough unregistered optional DVNs to make the threshold unreachable, verification for that EID can never complete.

— RECOMMENDATION

Keep the receive support predicate aligned with the actual receive requirements. The smallest safe change is to make `#isSupportedReceiveEid()` require every DVN in the default receive config to be registered before returning `true`.

Consider also rejecting default and OApp receive configs that reference unregistered DVNs so broken routes fail at configuration time instead of after packets start arriving.

— RESOLUTION

Status: Pending.

LOW **L-19**

PENDING

Event commitment excludes event type/name

CATEGORY	Validation
LOCATION	https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/0ccb36e3c7eb6f15b1adb349275af40323fe17bd/packages/api-common/src/container/component.ts#L228-L229
REVIEW	Main Review 1

— DESCRIPTION

In the current TS runtime, the component-level `eventHash` does not commit the event class/name. The base Event type exposes `eventName`, but the hashing path only includes `event.emitter` and `event.encode()` as payload. The base definition in `event.ts` confirms `eventName` is available for serialization, and `validator.ts:397` uses it for outward event reporting, but it is not part of the committed event hash.

As a result, two distinct event classes emitted by the same contract would be commitment-equivalent if they ABI-encode to the same payload bytes. This is not an immediate integrity break if event payload schemas are disjoint in practice, but it means the commitment is not formally binding to the semantic event type, only to (emitter, encoded payload).

— RECOMMENDATION

If event-type distinction is intended to be part of the commitment, include a stable event discriminator in `encode()` or in the outer event entry hashing format. Otherwise, document that event identity is defined by emitter plus payload shape, not by event name.

— RESOLUTION

Status: Pending.

LOW L-20

PENDING

PriceFeed uses different EID key spaces

CATEGORY Unexpected Behavior

LOCATION price-feed.ts

REVIEW Main Review 1

DESCRIPTION

`setPrice()`, `setPriceForArbitrum()` and `setEidToModelType()` store entries under the exact `eid` or `dstEid` they receive. `getPrice()` and `getModelType()` also read the exact key back.

`estimateFeeByEid()` does not use that same key space. It first rewrites the requested destination to `dstEid % 30_000` before it loads the model type and price data.

This means configuration and estimation use different EID key spaces. A caller can configure prices and model types under the raw destination EID that the rest of the endpoint stack uses, while fee estimation later reads the normalized key instead. The bug is not in the fee math itself. The bug is that the admin surface and the quote surface disagree on which storage key represents the same destination.

For example, an operator may treat `30001` as the destination EID everywhere else in the system. They call `setPrice([{ eid: 30001, ... }])` and `setEidToModelType([{ dstEid: 30001, modelType: OPTIMISM }])`. Later, `Executor` or `Dvn` asks `PriceFeed` to quote `dstEid = 30001`. `estimateFeeByEid()` converts that value to `1` before reading storage. If key `1` has no price, the quote reverts with

`PriceFeed_NoPriceSetError`. If key `1` has a price but no model type, the quote succeeds under the default model instead of the intended Optimism model. In both cases the operator configured the same EID that the rest of the endpoint stack uses, but estimation looked somewhere else.

The problem is not that every v2 EID quote is wrong. The problem is that correctness depends on the operator already knowing which representation `estimateFeeByEid()` will use internally and manually configuring storage under that alternate key. `Uln302`, `Executor` and `Dvn` all forward the raw `dstEid` through their normal fee flow, so the mismatch is reachable during ordinary `quote()` and `send()` execution.

RECOMMENDATION

L-20 PriceFeed uses different EID key spaces

CONTINUED

Pick one canonical representation for destination EIDs and use it consistently across both configuration and estimation. The smallest fix is to normalize `eid` and `dstEid` inside `setPrice()`, `setPriceForArbitrum()`, `setEidToModelType()` and the matching getters with the same helper that `estimateFeeByEid()` uses. If v1 and v2 aliases are not meant to be interchangeable, remove the modulo rewrite from `estimateFeeByEid()` and require callers to configure and query the exact stored EID.

— RESOLUTION

Status: Pending.

LOW **L-21**

PENDING

Canton receive prices native options it ignores

CATEGORY Unexpected Behavior

LOCATION fee-lib.ts

REVIEW Main Review 1

— DESCRIPTION

`ExecutorFeeLib.#parseExecutorOptions()` treats non-zero `lzReceive` value and `nativeDrop` amount as real executor-side native value. It adds both into `totalValue`, and `getFee()` later charges for that value through `convertAndApplyPremiumToValue()`.

That assumption does not hold for the Canton receive implementation. `DelegateLzReceive` only forwards a single `feeAmount` into `OApp_LzReceive`. The receive request builder consumes that fee for the Amulet-backed request and decodes only the LayerZero message. It does not decode executor options into native-value delivery. Canton does have a separate `NativeDrop` choice, but it is exposed as a separate helper and is not wired into `DelegateLzReceive`.

Consequently, a Canton-bound send can be quoted and accepted with non-zero `lzReceive` value or `nativeDrop` amount even though the destination receive execution ignores those options. The result is a mismatched option surface. The sender can be charged for executor value that the Canton receive stack does not deliver.

— RECOMMENDATION

Reject non-zero `lzReceive` value and `nativeDrop` options for Canton destinations until the receive implementation supports them end to end.

If those options are intended to work on Canton, decode them during executor handling and map them into concrete Canton operations. That includes a real equivalent for `lzReceive` value delivery and an automatic `NativeDrop` invocation.

— RESOLUTION

Status: Pending.

LOW L-22

PENDING

Odd-length signer input can wedge DVN quorum

CATEGORY Unexpected Behavior

LOCATION dvn.ts

REVIEW Main Review 1

DESCRIPTION

`setSigner()` approves one signer key and stores another when the input hex string has an odd number of nibbles. This happens because the request layer accepts odd-length values for `bytes64`, `hashSetDvnSigner()` and `#assertValidSignerPublicKey()` both normalize that input through `hexToBytes()`, and `#signers` then stores the raw caller string through `hexStringCodec`. Consequently, the quorum signs the left-padded 64-byte public key, but the committee stores a different key. `verifySignatures()` later recovers the canonical 64-byte signer key, so the malformed member cannot satisfy quorum even though it was added successfully. For a 127-nibble input, the stored array element is worse than just mismatched: `hexStringCodec` cannot decode it back. `getSigners()` can then fail, and removing the malformed member can also fail when deletion needs to read that array slot. This can leave the DVN with a committee size that looks larger than the number of signers that can actually authenticate. If quorum is configured against that inflated size, later governance updates and verification can become impossible to satisfy.

RECOMMENDATION

Enforce exact width for fixed-size byte arguments before `setSigner()` runs. A `bytes64` signer should require exactly 128 hex nibbles after `0x`, with no odd-length inputs accepted.

`setSigner()` should also canonicalize the signer key once and reuse that same canonical value for hashing, membership checks, storage, and recovered-signer comparison. Using a fixed-width signer codec instead of `hexStringCodec` is the safest way to keep those representations aligned.

RESOLUTION

Status: Pending.

LOW **L-23**

PENDING

Dynamic mapping keys use non-Solidity slots

CATEGORY Unexpected Behavior

LOCATION map.ts

REVIEW Main Review 1

— DESCRIPTION

`MapState.#deriveSlot()` derives mapping slots from `z.encode(codec, key)`. That is incorrect for dynamic keys. `stringCodec`, `hexStringCodec`, and `uint8ArrayCodec` emit full ABI value encodings with offset, length, and padded data, while Solidity mapping lookup for `bytes` and `string` uses the raw key bytes as the hashed preimage. Consequently, `MapState` writes dynamic-key entries under different trie slots than a Solidity-compatible verifier would derive.

This is already live in the repo. `EnumerableSetState` reuses its element codec as the backing `MapState` key codec, and `Dvn` builds its signer committee with `hexStringCodec`. The signer set is therefore committed under non-Solidity slots even though `@layerzerolabs/ver-merkle` advertises Ethereum-compatible roots and proofs. Local reads still succeed because the repo repeats the same derivation on both write and read. External proofs or slot lookups that follow Solidity rules will inspect a different location and can fail to verify signer membership correctly.

— RECOMMENDATION

Separate mapping-key encoding from value encoding. `MapState` and `SetState` should accept dedicated key encoders that implement Solidity's mapping-key rule directly, with 32-byte padded encoding for value types and raw bytes for `bytes` and `string`.

Until that split exists, reject `stringCodec`, `hexStringCodec`, and `uint8ArrayCodec` as mapping-key codecs. The current DVN signer set should use a fixed-width signer-key codec instead of `hexStringCodec`.

— RESOLUTION

Status: Pending.

LOW **L-24**

PENDING

Mixed-case bytes32 hashes break matching

CATEGORY Validation
LOCATION uIn-302.ts
REVIEW Main Review 1

— DESCRIPTION

`hexStringSchema` only adds a missing `0x` prefix. `createZodSchemaFromAbi()` then reuses that codec for every fixed-width `bytesN` field, including the `payloadHash` arguments on `MessagingChannel.nilify()`, `MessagingChannel.burn()` and `Uln302.finishGetVerificationState()`. Mixed-case `bytes32` hashes therefore survive request decoding unchanged.

That interacts badly with the rest of the hash handling code. Inbound payload hashes are stored through `hashCodec`, which decodes ABI `bytes32` values with `viem`. I confirmed locally against the installed `viem` that this decode lowercases the hex body. `nilify()` and `burn()` compare the stored hash to the caller-supplied `payloadHash` with raw string inequality and `finishGetVerificationState()` does the same with `existingPayloadHash === payloadHash`. The same 32-byte value therefore stops matching itself if the request uses checksum or other mixed-case hex.

A legitimate OApp can hit `MessagingChannel_PayloadHashNotFoundError` on `nilify()` or `burn()` even when it supplies the correct payload hash bytes. The verification-state helper can also report `Verifying` or `Verifiable` for a packet that is already committed, because the committed hash in storage does not string-match the mixed-case request argument.

These functions are packet-recovery and packet-status tools. A caller should not need the runtime's exact lowercase string form to nilify or burn a known payload hash. The read-side misreporting also makes off-chain automation less reliable, because one committed packet can appear unfinished purely due to hex casing.

— RECOMMENDATION

Canonicalize fixed-width hash values before comparing them. The smallest fix is to make `hexStringSchema`, or at least the `bytesN` branch of `createZodSchemaFromAbi()`, normalize hex to one representation such as lowercase `0x`-prefixed strings.

Avoid raw string equality for `Hash` values in `MessagingChannel` and `Uln302`. Compare canonicalized strings or compare the underlying bytes instead.

L-24 Mixed-case bytes32 hashes break matchingCONTINUED

— RESOLUTION

Status: Pending.

LOW **L-25**

PENDING

Ownable admin rotation is not exposed

CATEGORY Unexpected Behavior

LOCATION component.ts

REVIEW Main Review 1

DESCRIPTION

`ContainerComponent.#resolveMethod()` can only call methods on the root contract and the components returned by `contract.getComponents()`. Contracts inherit `getComponents() = []` by default, so an embedded `Ownable` helper stays invisible unless a contract explicitly exposes it.

This matters because several scoped contracts use `Ownable` to guard live admin actions, but the public vapp schemas do not export `Ownable.getAdmin()` or `Ownable.setAdmin()`. `Treasury` exposes `setLzTokenEnabled()`, `setNativeFeeBp()` and `setLzTokenFee()`. `PriceFeed` exposes `setPriceUpdater()` and other admin setters. `Uln302` exposes default-config setters and worker registration. All of those operations depend on `Ownable.assertAdmin()`, but none of them exposes a way to rotate that owner through the same dispatch surface. Consequently, a lost or compromised admin key cannot be replaced with a normal VER transaction. This is port-specific. It comes from the TypeScript container and schema surface, not from upstream LayerZero contract behavior.

RECOMMENDATION

Expose ownership management anywhere `Ownable` protects mutable state. The smallest safe fix is to surface the `Ownable` component from each affected contract with `getComponents()` and add `getAdmin` plus `setAdmin` to the exported vapp schemas. If ownership transfer should stay contract-specific, add explicit wrapper methods instead. Emit an ownership-change event and reject the zero address so a bad update cannot freeze the contract permanently.

RESOLUTION

Status: Pending.

LOW **L-26**

PENDING

Supported EIDs can still fail quote and send

CATEGORY Validation
LOCATION uln-302.ts
REVIEW Main Review 1

— DESCRIPTION

`Uln302.#isSupportedSendEid()` only checks for a default ULN config with at least one DVN. It does not check whether the EID has a usable executor configuration or whether the resolved executor is actually registered in this library. `MessageLibraryManager.setSendLibrary()` commits the library selection and then relies on the library's `assertSupportedEid()` continuation as its acceptance gate. Therefore an admin or OApp can successfully select a send library for an EID that is still not send-ready.

The runtime checks are stricter. `quote()` and `send()` both resolve the executor through `#getExecutorConfigOrThrow()` and `#getExecutor()`. If no executor is resolved, they throw `Uln302_InvalidExecutorError`. If the resolved executor is not in `#executors`, they throw `Uln302_ExecutorNotRegisteredError`. As a result, the configuration can be accepted first and fail only when the first real quote or send is attempted.

The zero-address case is already rejected by `setDefaultExecutorConfigs()`. The remaining issue is the readiness mismatch: "supported" only proves the DVN side is configured, while quote and send still require a valid registered executor.

— RECOMMENDATION

Keep the support check aligned with the conditions that `quote()` and `send()` actually require. The smallest safe fix is to make `assertSupportedEid()` reject EIDs whose resolved executor config is empty or points to an unregistered executor.

— RESOLUTION

Status: Pending.

INFORMATIONAL I-01

PENDING

Generic non-OApp Request_Create can reach unauthenticated ULN worker-registration methods

CATEGORY	Access Control
LOCATION	Global
REVIEW	Main Review 1

DESCRIPTION

The generic non-OApp request path allows arbitrary VER write transactions to be executed with minimal DAML-side validation, and `Uln302.registerPriceFeed()`, `registerDvn()`, and `registerExecutor()` do not enforce any local authorization before mutating trie-backed protocol state.

On the DAML side, non-OApp `IRequestFactory.Request_Create` is controlled only by sender, and `RequestFactory` validates only:

- that `callContext` has single-path shape
- that `callerAddr == sender`

It does not authenticate `callerName`, `targetName`, `targetAddr`, function name, or arguments for non-OApp requests. The TS layer then converts the `callContext` into executable VER transactions verbatim and dispatches them through the container.

As a result, any party that can use the generic non-OApp request path can target arbitrary VER write methods. In `Uln302`, the public methods `registerPriceFeed()`, `registerDvn()`, and `registerExecutor()` are exposed in the write ABI and do not perform `assertAdmin()` or any equivalent authorization check. An attacker can therefore use a generic request to create and register attacker-controlled workers directly inside the shared VER trie.

These methods do not merely store metadata.

They:

- increment ULN worker nonce
- create new `PriceFeed`, `Dvn`, and `Executor` contract instances in the trie
- add those addresses to ULN's registered worker sets
- write them into the component contract registry

This is an unauthorized protocol-state mutation primitive against the VER execution state.

RECOMMENDATION

I-01 Generic non-OApp Request_Create can reach unauthenticated ULN worker-registration methods

CONTINUED

Apply defense at both layers:

- Add local authorization checks to Uln302.registerPriceFeed(), registerDvn(), and registerExecutor() such as this.#ownable.assertAdmin(context).
- Restrict generic non-OApp Request_Create if arbitrary VER write execution is not intended for unprivileged parties.
- Consider allowlisting callable target contracts/functions for non-OApp requests, rather than relying on method-level discipline alone.
- More broadly, treat all VappApi-exposed write methods as externally reachable unless explicitly blocked by contract logic.

— RESOLUTION

Status: Pending.

INFORMATIONAL I-02

PENDING

Unused Error

CATEGORY	Superfluous Code
LOCATION	https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/0517a222c9408f6fe6d5fd018ca2d346730a8638/packages/endpoint/src/message-library-manager/error.ts#L57-L61
REVIEW	Main Review 1

DESCRIPTION

The `MessageLibraryManager_UnsupportedEidError` error is declared but never used.

RECOMMENDATION

Consider to remove unused code.

RESOLUTION

Status: Pending.

INFORMATIONAL I-03

PENDING

Sender-switching does not apply to constructors

CATEGORY	Informational
LOCATION	packages/protocol-common/src/contract-like.ts
REVIEW	Main Review 1

— DESCRIPTION

ContractLike uses a JavaScript Proxy to rewrite Context.msgSender / nextMsgSender for ordinary method calls between virtual contracts. However, that mechanism only applies when methods are accessed through the proxied instance.

Constructor execution itself is not part of that method-interception flow, and the proxy explicitly excludes the constructor property from wrapping.

As a result, if a constructor were to perform context-sensitive calls into another contract/helper expecting normal inter-contract caller semantics, the usual msg.sender switching behavior would not be guaranteed. This creates a semantic footgun: constructor-originated cross-contract logic would not behave like normal routed/proxied contract calls.

In the current scoped codebase, constructors appear to be used only for object wiring and state-wrapper initialization, and we did not identify a live bug caused by this pattern. The concern is architectural and preventative rather than an active exploit.

— RECOMMENDATION

Beware of this behaviour and dont make calls inside constructor for future as well.

— RESOLUTION

Status: Pending.

INFORMATIONAL I-04

PENDING

Storage compatibility for composite values

CATEGORY	Documentation
LOCATION	https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/36070d8a5d27ef416f0b864964c56d00aec6fa69/docs/audit/TS_AUDIT.md#L122-L123
REVIEW	Main Review 1

— DESCRIPTION

For current documentation, it multiple places its mentioned that storage follows solidity's storage layout. The current ver-merkle implementation is Ethereum-like in slot derivation and trie commitment, but composite values are encoded as single ABI-encoded blobs rather than stored using true Solidity field-by-field storage layout. The README still describes the package in terms that imply broader Ethereum/Solidity storage compatibility than the implementation actually provides.

— RECOMMENDATION

Beware and document this design difference from EVM.

— RESOLUTION

Status: Pending.

INFORMATIONAL I-05

PENDING

Cross-component calls use continuation handoff semantics rather than synchronous call/return

CATEGORY	Documentation
LOCATION	Global
REVIEW	Main Review 1

— DESCRIPTION

Cross-component interactions in the VER runtime do not behave like EVM-style synchronous contract calls. When a contract wants to invoke a contract in another component, it returns a continuation Transaction to the Container. The current method then finishes, the component state is committed, and the Container starts a new dispatch loop iteration for the next target.

This means cross-component execution follows a handoff model:

- contract A returns a continuation targeting contract B
- Container switches msg.sender to A's address
- Container resolves the component for B's address
- contract B executes as a fresh call

Contract A does not resume the remainder of its original method body after B returns.

Any later "return" to A must be expressed as another continuation into a new A method invocation rather than a resumed stack frame.

This is expected by design, but it is a meaningful semantic difference from EVM cross-contract calls and should be documented clearly because incorrect assumptions here can lead to flawed reasoning about atomicity, call stacks, and control flow.

— RECOMMENDATION

Document explicitly that:

- intra-component direct calls are synchronous and use ContractLike proxy sender rewriting
- cross-component calls are continuation-based handoffs mediated by Container
- cross-component flows cannot resume the original caller's remaining method body after the callee finishes

I - Cross-component calls use continuation handoff semantics rather than
05 synchronous call/return **CONTINUED**

— **RESOLUTION**

Status: Pending.

INFORMATIONAL I-06

PENDING

Slot isolation depends on sparse address spacing

CATEGORY	Configuration
LOCATION	packages/ver-merkle/src/contract/base.ts:22
REVIEW	Main Review 1

DESCRIPTION

For primitive storage wrappers, ver-merkle derives the base slot as `holderAddress + slotAllocator.next()`, then pads that numeric result into a 32-byte trie key.

This means primitive-slot namespaces are separated by the numerical distance between holder addresses, not by a structurally independent (address, slot) storage domain.

As a result, two contracts with sufficiently close holder addresses can overlap in principle if one contract allocates enough slots to numerically reach into the other contract's range.

This concern is much weaker for dynamically derived contract addresses, because those addresses are generated via `deriveGlobalAddress(...)` using `keccak256(domain || seed)` and are therefore effectively random and sparse in a 256-bit address space.

The concern is more relevant for `manually` chosen static addresses, especially in tests or bootstrap code that use small values such as 0x1000, 0x2000, 0x3000. Those addresses are close enough numerically that slot ranges can overlap in principle if enough primitive slots are allocated.

This is specific to the current storage model because slot derivation for primitive values is based on numeric address offset. It does not apply to EVM in the same way.

In EVM, contract storage is scoped by contract account first, and slot 0 in contract A is not numerically adjacent to slot 0 in contract B in a shared global keyspace.

RECOMMENDATION

- Avoid tightly spaced manually chosen static addresses in bootstrap configurations.
- Prefer deterministic hash-derived addresses for deployed contract holders.
- Consider a slot derivation scheme that structurally separates holder identity from slot index, rather than using numeric address offset for primitive slots.

RESOLUTION

Status: Pending.

INFORMATIONAL I-07

PENDING

TS lzReceive is pull-based, unlike EVM

CATEGORY	Documentation
LOCATION	https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/36070d8a5d27ef416f0b864964c56d00aec6fa69/packages/endpoint/src/endpoint-v2/endpoint-v2.ts#L372-L373
REVIEW	Main Review 1

— DESCRIPTION

In the TS implementation, `endpoint-v2.ts:365` treats `context.msgSender` as the receiver OApp and clears the committed payload for that caller.

This differs from the EVM model, where the executor calls `EndpointV2.lzReceive()`, and the endpoint then pushes delivery into the receiver contract.

So OApps, or some off-chain agent acting with OApp identity, must actively initiate the final delivery request.

This is a semantic divergence from EVM and should be documented as an integration assumption, because liveness and auth now depend on the TS↔request-layer pull model rather than on executor-driven push delivery alone.

— RECOMMENDATION

Document this explicitly for OApps deployed on canton.

— RESOLUTION

Status: Pending.

INFORMATIONAL I-08

PENDING

DVN governance does not have a sequencing nonce

CATEGORY	Signatures
LOCATION	https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/0ccb36e3c7eb6f15b1adb349275af40323fe17bd/packages/endpoint/src/dvn/dvn.ts#L508-L509
REVIEW	Main Review 1

DESCRIPTION

DVN multisig governance actions rely on expiration, vid, and exact-hash replay protection through `#usedHashes`. There is no monotonic governance nonce or epoch in the signed payloads for actions like `setQuorum`, `setSigner`, `setAllowlist`, `setDenylist`, `setDvnMessageLibrary`, and `quorumChangeAdmin`.

As a result, newer governance actions do not automatically invalidate older signed instructions; only the exact same signed message is consumed. This makes governance ordering rely on expiration and current signer/quorum drift rather than on an explicit single-step approval sequence.

RECOMMENDATION

Beware of this consideration.

RESOLUTION

Status: Pending.

INFORMATIONAL I-09

PENDING

Inconsistent malformed-argument handling

CATEGORY	Error
LOCATION	packages/api-common/src/container/component.ts:205
REVIEW	Main Review 1

— DESCRIPTION

Malformed external arguments can escape the normal `UserError` path because method argument decoding happens before the try/catch in the component dispatch boundary.

In component.ts:209, `request.schema.decode(args)` is executed before the block that wraps thrown Errors into `UserError`. As a result, malformed transactions such as missing required fields, wrong types, strict-object key violations, or malformed nested tuples can throw directly from schema decoding and bypass the container's normal in-band rejection flow.

This creates inconsistent boundary behavior.

— RECOMMENDATION

Consider moving request decoding inside the existing try/catch, or wrap decode failures into `UserError` at the same boundary as contract-thrown errors.

— RESOLUTION

Status: Pending.

INFORMATIONAL I-10

PENDING

Executor fees price only message bytes

CATEGORY Unexpected Behavior
LOCATION uln-302.ts
REVIEW Main Review 1

DESCRIPTION

`Uln302` prices executor delivery with `BigInt(message.length)` and passes that value into `Executor.assignJob()` as `calldataSize`. `ExecutorFeeLib` and `PriceFeed` then treat that input as the calldata size used for per-byte pricing.

That is smaller than the delivered payload. The destination delivery call is `EndpointV2.lzReceive(origin, guid, message)`, and the payload codec also prepends a fixed 32-byte `guid` to every message. However, the executor fee path prices only the raw message bytes and ignores those fixed bytes. The same pattern appears in `#quoteExecutor()`, so both quoting and fee collection use the reduced size.

This creates a systematic undercharge on chains where calldata bytes are priced separately from execution gas. The effect is strongest on Optimism-style pricing, where `PriceFeed` charges a distinct L1 data term from `callDataSize`. `lzReceiveBaseGas` can compensate for execution overhead, but it cannot precisely model missing calldata bytes in that separate L1 fee component. As a result, executor fees can be consistently too low, especially for short messages where the omitted fixed bytes are a large share of the total.

RECOMMENDATION

Do not use `message.length` as the executor `calldataSize`. Derive the billed size from the actual destination delivery footprint and reuse that value in both `quote` and `send`. At minimum, include the fixed bytes that are always delivered alongside the message, such as the `guid` and any other fixed arguments in `EndpointV2.lzReceive(...)`.

RESOLUTION

Status: Pending.

INFORMATIONAL I-11

PENDING

DVN library revocation does not affect verify

CATEGORY	Warning
LOCATION	dvn.ts
REVIEW	Main Review 1

DESCRIPTION

`setDvnMessageLibrary` updates the worker's mutable `messageLibraries` ACL, but `verify` does not read that ACL. Instead, `verify` accepts the call as long as `msgSender` is an admin, the quorum signatures are valid and the provided `messageLibrary` equals the DVN's fixed `#uInAddress`.

This means removing a library with `setDvnMessageLibrary(..., false)` only affects entry points that call `assertMessageLibrary`, such as `assignJob`. It does not disable `verify` for the ULN address that was bound when the DVN was created. Therefore, the library toggle does not revoke all DVN interaction with that library. It only revokes worker-style caller access.

`setDvnMessageLibrary` does not control all DVN interactions with a library. It only updates the worker ACL that gates calls such as `assignJob`. `verify` is governed by a different rule: it can still submit attestations to the DVN's fixed `#uInAddress` and it still requires an admin caller plus valid quorum signatures. The problem is that the API presents both controls as "message library" management even though they affect different behavior. After revoking a library, an operator can believe the DVN is fully disconnected from it when `verify` still remains bound to the originally configured ULN.

RECOMMENDATION

Make the interface reflect the actual control boundary. If `setDvnMessageLibrary` is only meant to govern worker caller access, rename it or document that `verify` is intentionally tied to the DVN's fixed ULN address. If the intent is to revoke all interaction with a library, then `verify` should also require the target library to remain present in `messageLibraries` or the DVN should store that target in one place instead of splitting the rule across `#uInAddress` and the worker ACL set.

RESOLUTION

Status: Pending.

INFORMATIONAL I-12

PENDING

TS_AUDIT describes outdated container semantics

CATEGORY	Documentation
LOCATION	https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/0ccb36e3c7eb6f15b1adb349275af40323fe17bd/docs/audit/TS_AUDIT.md#L442-L443
REVIEW	Main Review 1

DESCRIPTION

The audit document states that the container dispatches a transaction to all components in parallel via `Promise.allSettled`, expecting exactly one success and the others to reject.

The live implementation in `packages/api-common/src/container/container.ts` no longer behaves that way. Instead, it performs parallel ownership checks with `component.hasContract(...)` using `Promise.all`, selects

the first matching component, and executes only that component's `call(...)`.

This drift matters because the documented invariants are different from the runtime invariants. The current design still depends on unique contract ownership across components, but it does not rely on "exactly one execution succeeds; others fail" semantics.

RECOMMENDATION

Consider correcting the comments.

RESOLUTION

Status: Pending.

INFORMATIONAL I-13

PENDING

Fallback To Default Drops OApp Timeout Grace

CATEGORY Documentation

LOCATION packages/endpoint/src/message-library-manager/index.ts:455

REVIEW Main Review 1

DESCRIPTION

When an OApp switches its receive library from a non-default override back to the default library, the previous OApp-specific timeout is not preserved or consulted afterward.

`setReceiveLibrary(...)` explicitly forbids applying a `gracePeriod` when either the old or new library is the default sentinel (0x0). If the OApp falls back to default with `gracePeriod === 0`, the per-OApp timeout entry is deleted in `packages/endpoint/src/message-library-manager/index.ts:485`. After that, `getReceiveLibrary(...)` resolves the path as `default: true`, causing `isValidReceiveLibrary(...)` to consult only the default timeout branch, not the old OApp-specific timeout.

This means migration behavior is asymmetric: non-default to non-default supports a grace window, but non-default to default is immediate from the OApp-specific library's perspective. That may be intended, but it is easy to misread the timeout model as "old library remains valid for some grace period regardless of whether the new target is default or explicit," which is not how the current implementation behaves.

RECOMMENDATION

Document this explicitly in the receive-library migration semantics, especially for operators expecting a graceful fallback from custom receive libraries back to defaults.

RESOLUTION

Status: Pending.

INFORMATIONAL I-14

PENDING

OApp ULN override can set zero confirmations

CATEGORY Validation
LOCATION uln-config.ts
REVIEW Main Review 1

DESCRIPTION

`resolveUlnConfig()` merges the default ULN config with an OApp-specific override, then validates the merged result with `assertValidStandaloneConfig()`. That validation checks DVN presence, threshold bounds and total DVN count. It does not call `assertValidConfirmations()`. Therefore an OApp can set `useDefaultForConfirmations = false` and `confirmations = 0` and still produce a merged config that passes validation as long as the DVN arrays are otherwise valid.

This value is not inert. `Uln302.#setOappUlnSendConfig()` and `Uln302.#setOappUlnReceiveConfig()` store the override first, then treat `#getOappUlnSendConfig()` or `#getOappUlnReceiveConfig()` as the only post-merge validation step. Those helpers call `resolveUlnConfig()`, so the zero-confirmation override becomes the live config used by the library.

`Uln302.#checkVerifiable()` later consumes the merged `confirmations` value when it decides whether a packet is ready to commit. It passes that value into `#verified()` and `#verified()` only requires `verification.submitted` and `verification.confirmations >= requiredConfirmations`. Therefore a zero-confirmation override removes the finality delay entirely. A packet still needs the configured DVN attestations, but each attestation becomes sufficient as soon as it is recorded. Default configs reject the same value through `assertValidUlnConfig()`, so the confirmation invariant is enforced inconsistently between default settings and OApp-specific overrides.

RECOMMENDATION

Validate the merged config with the same confirmation rule that default configs already use. The smallest safe fix is to call `assertValidConfirmations(resolvedConfig)` inside `resolveUlnConfig()` before returning. Using `assertValidUlnConfig(resolvedConfig)` is also reasonable if the duplicate-DVN checks remain compatible with merged overrides. Either change keeps `confirmations > 0` enforced after the merge, which preserves the intended finality floor for OApp-specific configs.

I-14 OApp ULN override can set zero confirmations

CONTINUED

— **RESOLUTION**

Status: Pending.

INFORMATIONAL I-15

PENDING

Nilified packets can become executable again

CATEGORY Validation
LOCATION endpoint-v2.ts
REVIEW Main Review 1

DESCRIPTION

`MessagingChannel.nilify()` is an OApp-authorized control that stores `NIL_PAYLOAD_HASH` for a nonce:
```typescript

```
// packages/endpoint/src/messaging-channel/index.ts:127-141
```

```
await this.#base.assertAuthorized(context, oapp);
```

```
const currentPayloadHash = await this.#getInboundPayloadHash(oapp, srcEid, sender, nonce);
```

```
if (currentPayloadHash !== payloadHash) {
```

```
throw new MessagingChannel_PayloadHashNotFoundError(currentPayloadHash, payloadHash);
```

```
}
```

```
if (
```

```
nonce <= (await this.lazyInboundNonce(context, oapp, srcEid, sender)) &&
```

```
currentPayloadHash === EMPTY_PAYLOAD_HASH
```

```
) {
```

```
throw new MessagingChannel_InvalidNonceError(nonce);
```

```
}
```

```
await this.setInboundPayloadHash(context, oapp, srcEid, sender, nonce, NIL_PAYLOAD_HASH);
```

```
``
```

`EndpointV2.getExecutionState()` already treats that sentinel as a canceled packet and returns `NotExecutable`.

`EndpointV2.verify()` does not preserve that invariant. It calls `#verifiable()` and `#verifiable()` treats any non-empty stored hash as verifiable again. Since `NIL_PAYLOAD_HASH` is not `EMPTY_PAYLOAD_HASH`, a valid receive library can call `verify()` for the same nonce after the OApp nilified it. `verify()` then overwrites the nil sentinel with a live payload hash through `setInboundPayloadHash(...)`.

This lets the receive library undo an OApp's nilification decision and make the packet executable again. The OApp-only nilification control is therefore not final. A receiver that nilified a message to avoid processing it can later be forced back into the same message once the library recommits it.

---

**I-15** Nilified packets can become executable again**CONTINUED**

---

**— RECOMMENDATION**

Treat `NIL_PAYLOAD_HASH` as terminal inside the endpoint. `#verifiable()` should reject nilified nonces and `verify()` should refuse to overwrite an existing nil sentinel with a new payload hash.

**— RESOLUTION**

Status: Pending.

INFORMATIONAL I-16

PENDING

# lzReceiveAlert() emits spoofable alerts

CATEGORY Warning  
LOCATION endpoint-v2.ts  
REVIEW Main Review 1

## DESCRIPTION

`EndpointV2.lzReceiveAlert()` emits `EndpointV2_LzReceiveAlertEvent` with `context.msgSender` recorded as the executor. The function does not check that the caller is a registered executor, that the caller belongs to the configured message library, or that any delivery attempt actually failed. The intended helper in `packages/endpoint/src/executor/executor.ts:114-141` is admin-gated, but that does not protect the endpoint endpoint itself. Any external caller can invoke `EndpointV2.lzReceiveAlert()` directly and write a canonical alert event with arbitrary `receiver`, `origin`, `guid`, `message`, `extraData` and `reason` values. This makes the alert stream spoofable. Off-chain systems that rely on these events for delivery monitoring, retry logic, or incident handling can be grieved with fake failures. The write also hashes attacker-controlled dynamic byte arrays into the canonical event stream without any endpoint-side authorization gate.

## RECOMMENDATION

Restrict `lzReceiveAlert()` to the expected executor set, or at minimum to callers that are authorized through the configured receive stack. If the event is only valid after a failed delivery attempt, bind it to concrete packet state so arbitrary callers cannot mint standalone alerts.

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-17

PENDING

# Config getters are unreachable via public reads

|          |               |
|----------|---------------|
| CATEGORY | Warning       |
| LOCATION | endpoint.ts   |
| REVIEW   | Main Review 1 |

## DESCRIPTION

The endpoint API publishes `MessageLibraryManager.getSendConfig()` and `getReceiveConfig()` as public read methods. That makes them appear callable through the sequencer and validator read interfaces. Those getters are not actually public reads. The shared config getter still requires the caller to be the OApp or its delegate before it returns the continuation transaction to the target library. The existing tests only exercise these methods with an explicitly injected OApp caller.

The external read pipeline never carries a caller identity. Read requests only contain a transaction and state root and the validator executes them with both `txOrigin` and `msgSender` set to the zero address. Consequently, these two methods can never pass their own authorization check for any real OApp or delegate. The schema advertises live config inspection, but the only production read endpoint always calls them as `0x0` and they revert with `EndpointV2_UnauthorizedError` instead.

This matters because send and receive config are part of the live trust configuration for libraries, executors and DVNs. Exposing auth-gated getters through a callerless read API creates a silent observability gap during integration checks and incident response.

## RECOMMENDATION

Make the interface consistent in one direction or the other. If these getters are meant to stay authorization-gated, extend the public read request format and validator read context so the authenticated caller is propagated into `Context`. If the public read API is intentionally anonymous, remove `getSendConfig` and `getReceiveConfig` from the exported `READ` registry until they can be called successfully through that interface.

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-18

PENDING

# quorumChangeAdmin() drops the DVN admin check

CATEGORY Documentation

LOCATION dvn.ts

REVIEW Main Review 1

## DESCRIPTION

`docs/audit/TS_AUDIT.md` says `quorumChangeAdmin()` requires both admin authorization and quorum signatures. The implementation at `packages/endpoint/src/dvn/dvn.ts:339-357` does not call `assertAdmin(context)`, so that statement reads like a missing access-control check at first glance. After comparing the port against canonical LayerZero documentation, this is not a Canton-specific privilege bypass. It matches the upstream DVN design and follows the canonical LayerZero EVM-side recovery model.

LayerZero's DVN technical reference docs describe `quorumChangeAdmin()` as an emergency control that lets the signer quorum reassign the admin role without the current admin's permission. Those same docs also state that anyone can submit the signed transaction. This is the same rule used on the LayerZero EVM side. Therefore the signer quorum is supposed to authorize the change, while the submitter is only a relayer. For example, enough DVN signers can sign an instruction that says "add Bob as admin". Eve does not need to be an admin to submit that signed bundle and Bob still becomes admin.

A compromised signer quorum does not need cooperation from any current admin to seize the admin role. It can add an attacker-controlled admin through `quorumChangeAdmin()`, then use that new admin to exercise every admin-only capability on the worker. That includes fee and routing controls such as `setDstConfig()`, `setPriceFeed()` and payee management, plus the rest of the admin-plus-quorum governance surface. However, that is the documented upstream recovery model, not a TypeScript-port regression.

It also means any observer who gets a valid signed admin-rotation bundle can relay or front-run that rotation without current-admin participation. The weaker boundary is real, but the inconsistency is between the local audit documentation and the implemented upstream-compatible design.

## RECOMMENDATION

Update `docs/audit/TS_AUDIT.md` and any related local notes to say `quorumChangeAdmin()` is a quorum-only emergency admin rotation that does not require current-admin participation and can be relayed by any caller. If the Canton deployment truly wants a stricter local

**I-18** quorumChangeAdmin() drops the DVN admin check

CONTINUED

trust model than upstream, then add `assertAdmin(context)` and document that the port intentionally diverges from LayerZero.

**— RESOLUTION**

Status: Pending.

## INFORMATIONAL I-19

PENDING

# Default ULN zero-confirmation parity mismatch

|          |               |
|----------|---------------|
| CATEGORY | Warning       |
| LOCATION | uln-config.ts |
| REVIEW   | Main Review 1 |

## DESCRIPTION

`assertValidConfirmations()` rejects every config with `confirmations <= 0` and `setDefaultSendConfigs()` plus `setDefaultReceiveConfigs()` route default configs through that validator. As a result, the local port cannot accept a default ULN config with `confirmations = 0`. That behavior differs from the canonical LayerZero EVM implementation. Upstream `UlnBase.setDefaultUlnConfigs()` treats zero as a literal valid default value and reserves only `type(uint64).max` as the NIL sentinel.

## RECOMMENDATION

If the goal is strict LayerZero EVM parity, validate default ULN configs with a dedicated rule that reserves only the NIL sentinel and allows literal zero confirmations.

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-20

PENDING

# Interleaved DVN options make regrouping costly

CATEGORY Warning  
LOCATION dvn-options.ts  
REVIEW Main Review 1

## DESCRIPTION

`DvnOptions.groupDvnOptionsByIndex()` is meant to regroup DVN options by `dvn_index`. It stays cheap when all options for the same DVN are already adjacent. It gets much more expensive when the input is interleaved, for example `0,1,0,1,0,1`.

This happens because every time an index appears again, `#insertDvnOptions()` rebuilds the bytes collected for that DVN with `new ByteCodec().bytes(existing).bytes(newOptions).toBytes()`. That copies the old bytes again before adding the new chunk. If the same few indices keep alternating, the helper keeps re-copying larger and larger buffers instead of grouping everything in one linear pass.

`EndpointV2.quote()` and `EndpointV2.send()` both reach this helper through `Uln302`. In this repository that cost is paid in synchronous Node CPU. `quote` is also a public read method, so an external caller can force the service to spend time on regrouping before any useful work is done. I would still treat this as informational because the practical impact depends on request sizing and service limits rather than on a protocol safety failure.

LayerZero EVM shows the same underlying behavior. The official `DVNOptions.sol` at <https://github.com/LayerZero-Labs/LayerZero-v2/blob/main/packages/layerzero-v2/evm/messagelib/contracts/uln/libs/DVNOptions.sol> uses the same grouping pattern and concatenates prior bytes with `abi.encodePacked(...)` in `_insertDVNOptions`. The official `SendUlnBase.sol` at <https://github.com/LayerZero-Labs/LayerZero-v2/blob/main/packages/layerzero-v2/evm/messagelib/contracts/uln/SendUlnBase.sol> calls that helper from both `_quoteDVNs()` and `_payDVNs()`. Therefore this is not a TypeScript-only divergence. The impact is different, however. On EVM the caller pays gas and execution is gas-bounded. Here the same pattern burns service CPU on a public quote surface.

## RECOMMENDATION

Keep the regrouping step linear. Collect chunks per DVN index during the scan, track the final size for each group and build each grouped byte array once at the end. If the current behavior is intentionally aligned with upstream, add request-size or segment-count limits so public quoting cannot spend excessive CPU on regrouping alone.

---

**I-20** Interleaved DVN options make regrouping costly

---

CONTINUED

— **RESOLUTION**

Status: Pending.

## INFORMATIONAL I-21

PENDING

# Unbounded payee strings

|          |                                                                  |
|----------|------------------------------------------------------------------|
| CATEGORY | DoS                                                              |
| LOCATION | packages/endpoint/src/message-library/uln/uln-302/uln-302.ts:405 |
| REVIEW   | Main Review 1                                                    |

## DESCRIPTION

The `registerDvn` and `registerExecutor` functions accept a `payee: string` parameter without any contract-level length constraint.

At the ABI/schema layer, these values are also treated as unconstrained:

- `packages/ver-types/src/schemas/message-library/uln-302.ts:239` defines the field as `type: 'string'`
- `packages/ver-types/src/abi/schema-builder.ts:35` maps `string` to a plain `z.string()` with no maximum length enforcement

This value is subsequently ABI-encoded and written into trie-backed `StringState` storage:

- `packages/endpoint/src/worker/worker-base.ts:101`
- `packages/ver-merkle/src/contract/string.ts:30`
- `packages/ver-merkle/src/contract/codec.ts:48`

The public HTTP servers in this repository rely on default `express.json()` parsing, with no explicit request size limits configured:

- `apps/sequencer/src/server.ts:62`
- `apps/vapp/src/server.ts:32`

As a result, current protection appears to depend on transport or parser defaults rather than protocol-level constraints.

If larger payloads can reach execution—whether through deployment-specific ingress paths or direct Canton submission—oversized strings could introduce unnecessary memory consumption and encoding overhead in the shared TypeScript runtime.

In theory, if sufficiently large request payloads can reach execution, unbounded payee strings could cause excessive memory allocation in TS validators and potentially lead to process instability or denial of service.

## RECOMMENDATION

Consider adding explicit maximum lengths for payee and other unbounded string fields at the contract or schema boundary. Do not rely on HTTP body limits or external transport defaults as the only safeguard.

---

**I-21** Unbounded payee strings

CONTINUED

---

— **RESOLUTION**

Status: Pending.

## INFORMATIONAL I-22

PENDING

# load(...) must preserve layout across TS upgrade

|          |               |
|----------|---------------|
| CATEGORY | Informational |
| LOCATION | Global        |
| REVIEW   | Main Review 1 |

## — DESCRIPTION

Persistent contract state is reconstructed via `load(...)` methods rather than being redeployed from scratch. These loaders recreate state handles using a fresh `SlotAllocator`, and correctness depends on rebuilding them with the same field order, holder address, and codecs as the original layout.

Slot derivation is based on `address + slotAllocator.next()` (`packages/ver-merkle/src/contract/base.ts:24`). As a result, any post-deployment code change that alters the field order or layout assumptions within `load(...)` can cause upgraded code to read existing trie data from incorrect slots.

This is analogous to storage layout compatibility in upgradeable Solidity contracts, but centered on loader reconstruction. New code can safely extend state only by appending fields, provided that `load(...)` continues to rebuild all pre-existing state handles exactly as before.

## — RECOMMENDATION

Beware of this constraint, and always make sure in general that order of object's initialization in create matches with load.

## — RESOLUTION

Status: Pending.

## INFORMATIONAL I-23

PENDING

# Receive path is not atomic due to callbacks

|          |               |
|----------|---------------|
| CATEGORY | Informational |
| LOCATION | Global        |
| REVIEW   | Main Review 1 |

## DESCRIPTION

In Solidity, `EndpointV2.lzReceive(...)` clears the payload and invokes the OApp receiver within the same transaction. If the OApp callback reverts, the entire transaction reverts, and the packet remains uncleared.

In the TS/DAML design, the receive flow is split into two stages:

1. The TS endpoint clears the payload
2. The OApp-specific callback is finalized separately via `PendingCallback` / `FinalizeCallback`

As a result, `packages/endpoint/src/endpoint-v2/endpoint-v2.ts:481` may return

`ExecutionState.Executed` as soon as the payload is cleared and the lazy inbound nonce advances—even if the downstream DAML callback has not yet succeeded or is still retrying.

This changes the semantics of `ExecutionState.Executed`. Unlike Solidity, it no longer guarantees that:

>

"the packet was delivered and the OApp callback succeeded atomically."

Instead, it only indicates that the endpoint-side protocol state has consumed the packet. The application-layer effect may still be pending in `PendingCallback` finalization.

While failed callback finalizations remain retryable (since

`daml/Layerzero/Request/daml/PendingCallback.daml:59` archives the callback only after successful dispatch), this split execution model introduces:

- Potential liveness issues
- Misleading status signals for monitors or integrations assuming EVM-like semantics

## RECOMMENDATION

Consider documenting `ExecutionState.Executed` as meaning *"endpoint-consumed"* rather than *"fully finalized at the OApp layer."*

Any application or monitoring logic that requires end-to-end delivery guarantees should also track the DAML callback finalization state.

---

**I-23** Receive path is not atomic due to callbacks

CONTINUED

---

— RESOLUTION

Status: Pending.

## INFORMATIONAL I-24

PENDING

# DAML Receive Callback and msg.value Forwarding

CATEGORY Documentation

LOCATION daml/Layerzero/Oft/daml/OApp/LzReceive/Call/Impl.daml:28

REVIEW Main Review 1

## DESCRIPTION

In `lz-evm`, the executor invokes the OApp receiver as:

```
`plain text`
```

```
ILayerZeroReceiver(_receiver).lzReceive{ value: msg.value }(...
```

This allows native value ( `msg.value` ) to be forwarded directly into the receive callback.

In the Canton/DAML design, the subsequent `LzReceive_callback` `accept path`

( `daml/Layerzero/Oft/daml/OApp/LzReceive/Callback/Accept.daml:27` ) only processes message, configuration, and rate-limit inputs. It does **not** receive any direct native-value or Amulet-value equivalent at callback time.

As a result, the funding model for receive callbacks differs materially from Solidity:

- There is no equivalent of forwarding `msg.value` at execution time
- All economic support for the operation must be preconfigured during request creation

**Implication:**

Application logic cannot assume EVM-like semantics where callbacks are invoked with attached native value. Any required funding must already be accounted for within the request and fee infrastructure.

## RECOMMENDATION

Document this as a callback funding model difference from `lz-evm`, ensuring that OApp developers do not rely on `msg.value`-style behavior during DAML callback finalization.

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-25

PENDING

# No fee-based prioritization under load

|          |                                                           |
|----------|-----------------------------------------------------------|
| CATEGORY | Gaming                                                    |
| LOCATION | daml/Layerzero/RequestFactory/daml/RequestFactory.daml:73 |
| REVIEW   | Main Review 1                                             |

## — DESCRIPTION

The DAML/Canton request layer validates only a minimum fee at request creation rather than pricing requests according to expected TS execution cost. As a result, the system also lacks a native fee-based priority mechanism: under congestion, users cannot bid more to prefer urgent requests over background or adversarial traffic. This differs from gas-market systems where price signals help both spam resistance and transaction prioritization.

## — RECOMMENDATION

If latency under load matters, introduce explicit prioritization policy, differentiated request classes, or fee tiers tied to scheduling rather than relying on a flat minimum request fee alone.

## — RESOLUTION

Status: Pending.

## INFORMATIONAL I-26

PENDING

# Skipped & Burnt Packets Are Reported As Executed

|          |                                                                                                                                                                                                                                                                                                                                                             |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Unexpected Behavior                                                                                                                                                                                                                                                                                                                                         |
| LOCATION | <a href="https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/36070d8a5d27ef416f0b864964c56d00aec6fa69/packages/endpoint/src/endpoint-v2/endpoint-v2.ts#L481-L488">https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/36070d8a5d27ef416f0b864964c56d00aec6fa69/packages/endpoint/src/endpoint-v2/endpoint-v2.ts#L481-L488</a> |
| REVIEW   | Main Review 1                                                                                                                                                                                                                                                                                                                                               |

## DESCRIPTION

The `getExecutionState` function classifies any packet with `EMPTY_PAYLOAD_HASH` and nonce `\<= lazyInboundNonce` as `Executed`. However, that storage shape is not unique to packets that were actually delivered to the OApp. `skip` advances the `lazyInboundNonce` without delivery, and `burn` deletes the stored payload hash, which also leaves the packet with an empty hash inside the lazy inbound window. As a result, administratively skipped or burnt packets are reported as `Executed`, conflating operational cleanup actions with successful delivery. This mainly affects observability.

## RECOMMENDATION

Consider to document this behavior or to track skipped and burnt packets explicitly.

## RESOLUTION

Status: Pending.

INFORMATIONAL I-27

PENDING

# SupportedOptionTypes Is Unused

|          |                                                                                                                                                                                                                                                                                                                                                   |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Superfluous Code                                                                                                                                                                                                                                                                                                                                  |
| LOCATION | <a href="https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/36070d8a5d27ef416f0b864964c56d00aec6fa69/packages/endpoint/src/worker/worker-base.ts#L211-L223">https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/36070d8a5d27ef416f0b864964c56d00aec6fa69/packages/endpoint/src/worker/worker-base.ts#L211-L223</a> |
| REVIEW   | Main Review 1                                                                                                                                                                                                                                                                                                                                     |

## DESCRIPTION

The `SupportedOptionTypes` functionality seems to be unused in the codebase.

## RECOMMENDATION

Consider removing the `SupportedOptionTypes` functionality if it is not used to follow best practices.

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-28

PENDING

# Unused LZ Token Fee Surface Remains Exposed

|          |                                                                                                                                                                                                                                                                   |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Informational                                                                                                                                                                                                                                                     |
| LOCATION | <a href="https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/main/packages/api-common/src/schemas/vapps/treasury.ts">https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/main/packages/api-common/src/schemas/vapps/treasury.ts</a> |
| REVIEW   | Main Review 1                                                                                                                                                                                                                                                     |

## DESCRIPTION

The VER explicitly skips LZ-token payment handling in `finishSend`, but the treasury layer still exposes LZ Token related functions like for example: `getLzTokenFee`.

## RECOMMENDATION

Consider to remove or disable the LZ-token fee surface completely or to document this behavior.

## RESOLUTION

Status: Pending.

INFORMATIONAL I-29

PENDING

# Stale startSend references in comments

|          |                                                      |
|----------|------------------------------------------------------|
| CATEGORY | Documentation                                        |
| LOCATION | packages/api-common/src/schemas/transaction.ts:25,53 |
| REVIEW   | Main Review 1                                        |

— DESCRIPTION

The comments in `transaction.ts` use `startSend` and `Endpoint.startSend` as example method names, but there is no live `startSend` path in the repository.

While this does not affect runtime behavior, inaccurate inline documentation is a documentation hygiene issue and diverges from best practices.

— RECOMMENDATION

Consider to update these comments to reference currently supported methods to follow best practices and reduce confusion.

— RESOLUTION

Status: Pending.

INFORMATIONAL I-30

PENDING

# Consumed Nonce Marks Arbitrary Payloads Verified

|          |                                                                                                                                                                                                                                                                                                                                                                           |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Unexpected Behavior                                                                                                                                                                                                                                                                                                                                                       |
| LOCATION | <a href="https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/36070d8a5d27ef416f0b864964c56d00aec6fa69/packages/endpoint/src/message-library/uln/uln-302/uln-302.ts#L829">https://github.com/GuardianOrg/canton-ver-team1-1773774383947/blob/36070d8a5d27ef416f0b864964c56d00aec6fa69/packages/endpoint/src/message-library/uln/uln-302/uln-302.ts#L829</a> |
| REVIEW   | Main Review 1                                                                                                                                                                                                                                                                                                                                                             |

## DESCRIPTION

The `finishGetVerificationState` function returns `Verified` whenever `endpointVerifiable` is false. After a packet has already been cleared and the nonce is consumed, `endpointVerifiable` becomes false for that nonce. As a result, `getVerificationState` can be queried with a different payload hash for the same nonce and still report `Verified`, even though that payload hash was never actually verified. This does not appear to let an invalid payload execute, but it can mislead offchain tooling.

## RECOMMENDATION

Be aware and consider to document this behavior.

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-31

PENDING

# Endpoint API exports unusable contract names

CATEGORY Unexpected Behavior

LOCATION endpoint.ts

REVIEW Main Review 1

## DESCRIPTION

`endpointVappApi` exports `MessagingChannel` and `MessageLibraryManager` as top-level contract identities next to `EndpointV2`. That is not how the endpoint family is actually addressable at runtime. `EndpointComponent` registers the endpoint address only under `EndpointV2`, and `ContainerComponent.#resolveContract()` requires `functionSignature.contract` to match that stored name before method dispatch reaches `getComponents()`. Consequently, a client that follows the exported API and sends `contract: "MessagingChannel"` or `contract: "MessageLibraryManager"` to the endpoint address gets `ContractNotFoundError`, even though the corresponding method exists on the loaded endpoint family. Internal calls avoid this only because they already use the hidden convention of targeting `EndpointV2` for those methods.

This makes the wire-level contract field misleading for real endpoint operations. Methods such as `skip`, `nilify`, `burn`, `registerLibrary`, and the default-library setters can appear unavailable if a caller relies on the exported contract names instead of the undocumented `EndpointV2` convention.

## RECOMMENDATION

Make the endpoint family use one contract-identity model consistently. The simplest fix is to expose all endpoint-family methods only under `EndpointV2` and remove `MessagingChannel` plus `MessageLibraryManager` from `endpointVappApi`. If those subcomponents are meant to stay independently addressable, then register callable identities for them and update contract resolution so those names can be loaded safely at the endpoint address.

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-32

PENDING

# Mutation Testing Results

|          |               |
|----------|---------------|
| CATEGORY | Informational |
| LOCATION | Global        |
| REVIEW   | Main Review 1 |

## DESCRIPTION

We ran Stryker to evaluate the quality of the current tests. Below is a summary of the key gaps where coverage could be improved.

**\*\*Full results:\*\***

[<https://github.com/GuardianOrg/canton-ver-team1-1773774383947/tree/guardian/mutation-testing-results>]  
(<https://github.com/GuardianOrg/canton-ver-team1-1773774383947/tree/guardian/mutation-testing-results>)

### 1. ULN config resolution and fee plumbing

Files:

- `packages/endpoint/src/message-library/uln/uln-302/uln-302.ts:623`
- `packages/endpoint/src/message-library/uln/uln-302/types/uln-config.ts:73`

Surviving-mutant themes:

- required vs optional DVN handling
- optional-threshold satisfaction
- supported-EID gating
- fee-path edge cases around treasury and worker-fee composition
- DVN option routing/index handling

### 2. Executor fee arithmetic

File:

- `packages/endpoint/src/executor/fee-lib.ts:58`

Surviving-mutant themes:

- `dstEid < 30000` boundary
- ordered-execution surcharge
- compose gas aggregation
- native-cap enforcement
- value-premium arithmetic

### 3. Price-model boundaries

File:

- `packages/endpoint/src/price-feed/price-feed.ts:304`

Surviving-mutant themes:

- "no price set" detection

## I-32 Mutuation Testing Results

CONTINUED

- Optimism L1 lookup routing
- fee assertions around partial-zero price inputs
- EID bucket / lookup transitions

### ### 4. DVN multisig validation

Files:

- `packages/endpoint/src/dvn/dvn.ts:196`
- `packages/endpoint/src/dvn/dvn.ts:455`

Surviving-mutant themes:

- wrong `messageLibrary` handling
- duplicate / non-strict signer ordering
- recovery-id validation
- malformed recovered public-key handling

### ### 5. Receive-state transitions

Files:

- `packages/endpoint/src/endpoint-v2/endpoint-v2.ts:470`
- `packages/endpoint/src/messaging-channel/index.ts:129`

Surviving-mutant themes:

- `EMPTY` vs `NIL` payload-hash behavior
- lazy inbound nonce boundaries
- distinction between `Executed`, `Executable`, and only `Verified`

### ### 6. ACL / admin / runtime glue

Files:

- `packages/endpoint/src/worker/worker-base.ts:118`
- `packages/protocol-common/src/context.ts:114`
- `packages/api-common/src/container/component.ts:150`

These are less protocol-core than the items above, but still security-relevant.

Surviving-mutant themes:

- allowlist size bookkeeping
- exact `msg.value` depletion boundaries
- event hashing across continuations
- dynamic method dispatch and void-response encoding

### — RECOMMENDATION

Consider improving coverage at these points

### — RESOLUTION

Status: Pending.

MEDIUM **M-01**

PENDING

# Operations Remain Callable While EID Is 0

CATEGORY Validation

LOCATION <https://github.com/GuardianOrg/audit-external-team1-1777914234503/blob/01d6c7d2553d2b2233f42db3a6d54cee8216a478/contracts/protocol/ver-endpoint/src/common/guid.ts#L31-L33>

REVIEW Main Review 2

## DESCRIPTION

`EndpointV2.create` now leaves the endpoint `EID` unset at genesis and expects a later admin call to `setEid`.

However, `quote` and `send` still read `srcEid` directly from the messaging channel without asserting that it has been initialized, and `validateGuidInputs` only rejects EIDs outside the uint32 range, therefore it accepts `0n`.

As a result these functions are callable while the EID is not set.

## RECOMMENDATION

Consider to reject if the local EID equals 0.

## RESOLUTION

Status: Pending.

LOW **L-01**

PENDING

# Inbound nonce checks can scan unbounded backlogs

CATEGORY Validation

LOCATION `contracts/protocol/ver-endpoint/src/messaging-channel/index.ts`

REVIEW Main Review 2

## — DESCRIPTION

`MessagingChannel.setInboundPayloadHash()` stores a payload hash for any nonce that passes the endpoint's verifiability check. It does not update an explicit pending nonce list and it does not enforce a maximum distance from the current lazy inbound nonce.

`MessagingChannel.inboundNonce()` then reconstructs the effective inbound nonce by starting from `lazyInboundNonces` and reading one map slot at a time until it finds a missing payload hash.

`clearPayload()` performs the same kind of linear walk when a receiver clears a later nonce. Therefore a valid receive library can create a large consecutive backlog of verified nonces and later calls that need the effective inbound nonce must scan the whole backlog before returning.

This can make `inboundNonce`, `skip`, `clearPayload` and endpoint execution-state checks grow linearly with the number of verified but uncleared messages for a `(receiver, srcEid, sender)` tuple. If VER execution is metered, those calls can exceed the execution budget. If it is not metered, a large backlog still creates an avoidable storage-read DoS surface for receivers and delegates.

The Stellar endpoint implementation in this repository handles the same problem with a bounded pending inbound nonce list and a `PENDING_INBOUND_NONCE_MAX_LEN` limit. This TypeScript implementation has no equivalent bound.

## — RECOMMENDATION

Track pending inbound nonces explicitly and advance the effective inbound nonce during verification instead of reconstructing it with an unbounded scan. Reject future nonces that are more than a configured maximum gap beyond the current inbound nonce.

At minimum, add a hard cap to the scan in `inboundNonce()` and `clearPayload()`, then return a clear error when the cap is reached. Add regression tests that verify large out-of-order or consecutive verified backlogs cannot make receive-side calls perform unbounded storage reads.

---

**L-01** Inbound nonce checks can scan unbounded backlogsCONTINUED

---

**— RESOLUTION**

Status: Pending.

LOW **L-02**

PENDING

# ULN And Endpoint EIDs Can Diverge

CATEGORY Validation  
LOCATION GLOBAL  
REVIEW Main Review 2

## — DESCRIPTION

The endpoint-side `EID` and the ULN-side `EID` are now configured by two independent one-shot admin setters. There is no invariant enforcing equality between these stored values.

## — RECOMMENDATION

Consider to remove the duplicated source of truth or enforce equality.

## — RESOLUTION

Status: Pending.

## INFORMATIONAL I-01

PENDING

# DVN signatures omit deployment domain

|          |                                                                  |
|----------|------------------------------------------------------------------|
| CATEGORY | Signatures                                                       |
| LOCATION | contracts/protocol/ver-endpoint/src/dvn/multisig/hash-builder.ts |
| REVIEW   | Main Review 2                                                    |

## DESCRIPTION

`DvnHashBuilder.hash` hashes the packed DVN action bytes directly. The action helpers include the action selector, DVN address, VID, expiration, and action-specific fields, but they do not include a deployment domain such as chain ID, VER network ID, vApp identity or hash schema version.

`Dvn.verifySignatures` also recovers signers from the raw digest, without an EIP-191 or typed-data domain prefix. Therefore a signature is only scoped by the values present in the packed payload. If two deployments reuse the same signer set and can produce the same DVN address, VID, target address and action payload, a signature from one deployment can verify in the other until it expires.

## RECOMMENDATION

Add an explicit domain separator to every DVN multisig preimage. The domain should include at least a protocol name, hash schema version, deployment or network identifier and the verifying DVN address. If raw secp256k1 digests are intentional, document that signer infrastructure must sign this exact domain-separated digest rather than generic wallet messages.

## RESOLUTION

Status: Pending.

INFORMATIONAL I-02

PENDING

# ULN Exposes setEid But No getEid Method

|          |                                                                                                                                                                                                                                                                                                                             |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Informational                                                                                                                                                                                                                                                                                                               |
| LOCATION | <a href="https://github.com/GuardianOrg/audit-external-team1-1777914234503/blob/main/contracts/protocol/ver-endpoint/src/message-library/uln/uln-302/uln-302.ts">https://github.com/GuardianOrg/audit-external-team1-1777914234503/blob/main/contracts/protocol/ver-endpoint/src/message-library/uln/uln-302/uln-302.ts</a> |
| REVIEW   | Main Review 2                                                                                                                                                                                                                                                                                                               |

## DESCRIPTION

ULN now has a `setEid` function but misses a `getEid` function as implemented in other parts of the system.

## RECOMMENDATION

Consider to implement a `getEid` function.

## RESOLUTION

Status: Pending.

## § 05

# Disclaimer.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian's position is that each company and individual are responsible for their own due diligence and continuous security. Guardian's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis.

Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract's safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

§ 06

# About Guardian.

Guardian is a smart contract security firm. Every engagement is worked by two independent teams competing on the same scope, each with its own tooling and methods, so no single reviewer's blind spot survives.

**\$45B+**DIGITAL ASSETS UNDER  
REVIEW**200+**

ENGAGEMENTS COMPLETED

**500+**CRITICAL & HIGH FINDINGS  
REPORTED

Reach us at [audits@guardianaudits.com](mailto:audits@guardianaudits.com).