



TECHNICAL SECURITY ASSESSMENT

Northstar Custody

Infrastructure Security Review · Custody

REPORT ID QX-TSA-2026-011

QUANTIX RESEARCH ASSESSMENT

MPC signing nodes (Rust), policy engine (Go), client API and approvals console (TypeScript), and their deployment configuration, for Ethereum (ECDSA) and Solana (Ed25519) signing. 13,930 nSLOC plus infrastructure configuration.

Document status

Published

Published

18 September 2026

Review period

13 July 2026 – 14 August 2026

Ecosystems

Ethereum, Solana

Languages

Rust, Go, TypeScript

Review effort

Four reviewers over twenty-five working days, including an authorised penetration test of the staging environment

Reviewed commit

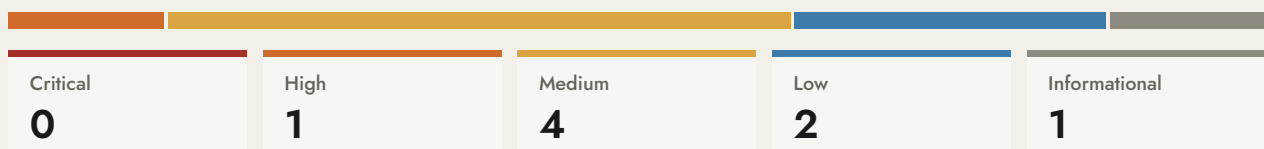
9d4b17e0c2a6f83b5e1d0a7c94f26b8e3a0c5d71

Remediation commit

c27e05a9b3f18d64e0a2c7b95d13f86a4e9b0c38

FINDINGS BY SEVERITY

8 findings · 100% remediated



This document is a Quantix research assessment of Northstar Custody, a protocol design from Quantix Protocol Research. It is not a client engagement, it does not describe a deployed system, and it is not a certification.

DOCUMENT CONTROL

Document ID	QX-TSA-2026-011
Title	Technical Security Assessment: Northstar Custody
Version	1.1
Status	Published
Assessment type	Infrastructure Security Review
Review period	13 July 2026 – 14 August 2026
Published	18 September 2026
Reviewed commit	9d4b17e0c2a6f83b5e1d0a7c94f26b8e3a0c5d71

CONTENTS

1	Executive summary	3
2	Scope	4
3	Protocol architecture	5
4	Threat model	5
5	Methodology and severity classification	6
6	Findings summary	7
7	Detailed findings	8
	NC-01 Approval quorum attributes approvals to the approver named in the request body	8
	NC-02 Policy is evaluated at request time and not re-evaluated before signing	9
	NC-03 Signing nodes sign the digest relayed by the coordinator without rebuilding it	10
	NC-04 Decommissioned signing nodes keep valid key shares until the next scheduled refresh	11
	NC-05 API request signatures do not cover a timestamp or nonce, so requests can be replayed	12
	NC-06 Webhook signing secret is passed in a URL query string in the console	13
	NC-07 Velocity limits apply per asset rather than across a wallet's total value	14
	NC-08 Unmaintained transitive dependency in the signing node	14
8	Severity and risk analysis	16
9	Remediation status	17
10	Formal verification and invariants	17
11	Recommendations	18
12	Limitations	18
A	Technical appendix	19

1

Executive summary

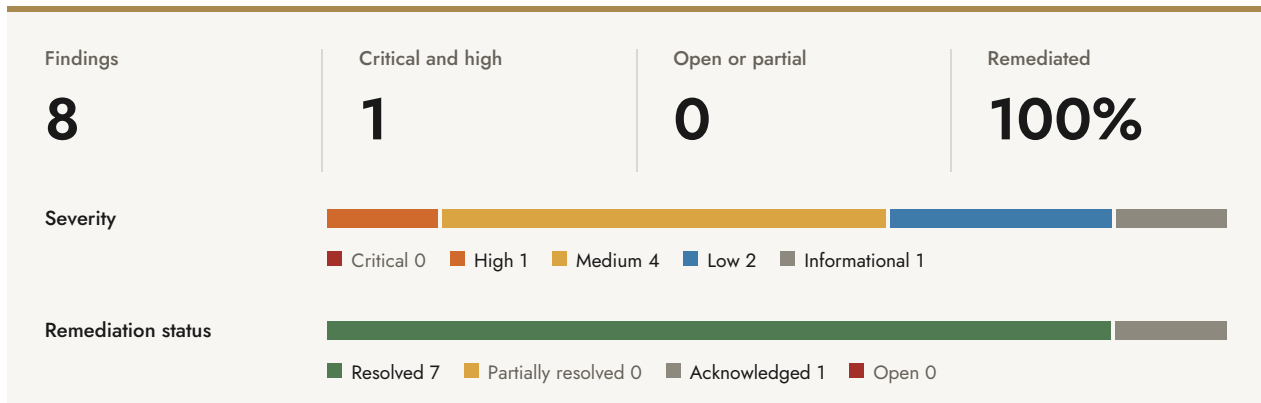
Northstar Custody is an institutional custody platform. Private keys are never assembled: key shares are held by isolated MPC signing nodes, and every transaction must pass a policy engine (whitelists, velocity limits and approval quorums) before the nodes sign. Clients submit requests through an API and approve them in a web console with hardware security keys.

Quantix reviewed the signing nodes, policy engine, client API, approvals console and deployment configuration, and ran an authorised penetration test against the staging environment. The most significant finding was a quorum bypass in the policy engine: approvals were attributed to the approver named in the request body rather than to the authenticated WebAuthn session, so one approver could record approvals on behalf of others.

Four medium-severity findings concerned a time-of-check to time-of-use gap between policy evaluation and signing, signing nodes trusting the digest supplied by the coordinator, decommissioned nodes keeping valid key shares, and replayable API requests. Low-severity findings covered a webhook secret exposed in URLs and per-asset velocity limits.

All high and medium findings were fixed and verified at the remediation commit. One low-severity finding about aggregate velocity limits is acknowledged and handled through a planned policy feature. The platform's design, with independent digest reconstruction on every node after remediation, gives strong assurance that a single compromised component cannot cause an unauthorised signature.

AT A GLANCE



SEVERITY SUMMARY

Severity	Definition	Count	Open
■ Critical	Direct loss of funds or permanent loss of control is achievable with little or no precondition.	0	0
■ High	Material loss or protocol disruption is achievable under realistic conditions.	1	0
■ Medium	Limited loss, degraded functionality, or impact that depends on specific conditions.	4	0
■ Low	Minor impact, defence-in-depth gaps, or issues with unlikely preconditions.	2	0

Severity	Definition	Count	Open
■ Informational	Code quality, documentation or best-practice observations without direct security impact.	1	0

2

Scope

MPC signing nodes (Rust), policy engine (Go), client API and approvals console (TypeScript), and their deployment configuration, for Ethereum (ECDSA) and Solana (Ed25519) signing. 13,930 nSLOC plus infrastructure configuration.

Repository	github.com/example-org/northstar-platform
Reviewed commit	9d4b17e0c2a6f83b5e1d0a7c94f26b8e3a0c5d71
Remediation commit	c27e05a9b3f18d64e0a2c7b95d13f86a4e9b0c38
Review period	13 July 2026 – 14 August 2026
Review effort	Four reviewers over twenty-five working days, including an authorised penetration test of the staging environment

Path	Description	nSLOC
services/signer	MPC signing node: session handling, digest construction, key-share storage and refresh (Rust)	4,120
services/policy	Policy evaluation: whitelists, velocity limits, approval quorums (Go)	3,560
services/api	Client API: request authentication, transaction requests, webhooks (TypeScript)	2,940
apps/console	Approvals console with WebAuthn hardware-key approvals (TypeScript)	3,310
deploy	Terraform and Kubernetes configuration for staging and production	–
Total		13,930

Out of scope

- The third-party threshold-signature library used by the signing nodes (treated as correct; its integration was in scope)
- HSM firmware and the physical security of signing-node hosts
- Individual clients' policy configurations
- Chain RPC providers

3

Protocol architecture

A client creates a transaction request through the API, authenticated with an API key and an HMAC request signature. The policy engine evaluates the request against the client's policy and, where required, opens an approval round. Approvers review the decoded transaction in the console and approve with a WebAuthn assertion from a hardware key.

When the quorum is met the policy engine issues a signed authorisation. A coordinator schedules a signing session across the MPC nodes, which run a threshold protocol (ECDSA for Ethereum, Ed25519 for Solana) and return a signature that the API broadcasts. Nodes run in separate regions and accounts with distinct operators.

Component	Role	Trust level
MPC signing nodes	Hold key shares and run threshold signing	Trusted
Policy engine	Evaluates policy and issues signing authorisations	Trusted
Signing coordinator	Schedules sessions and relays protocol messages	Semi-trusted
Client API	Request intake, authentication and webhooks	Semi-trusted
Approvals console	Human approval with hardware keys	Semi-trusted
Approvers	Client staff approving transactions	Semi-trusted
Client systems and network	Callers of the API	Untrusted

Trust assumptions

- Fewer than the signing threshold of node operators collude or are compromised.
- Approvers' hardware keys are not all compromised at once; individual approvers may act maliciously.
- The threshold-signature library implements its protocol correctly.
- Clients configure policies appropriate to their risk (see NC-07).

4

Threat model

Assets

- Key shares
- Signing authorisations
- Client policies
- Approver sessions
- API credentials and webhook secrets

Actors

Actor	Capability	Trust
Malicious approver	Holds one valid approver account and hardware key	Semi-trusted

Actor	Capability	Trust
Compromised coordinator	Controls session scheduling and message relay	Semi-trusted
Network attacker	Observes or replays client traffic	Untrusted
Former node operator	Retains material from a decommissioned node	Untrusted
Insider with log access	Reads proxy and application logs	Semi-trusted

Threats

Threat	Actor and vector	Mitigation	Findings
Meet a quorum with fewer real approvers	Malicious approver: Approver identity taken from request body	Bind approvals to the WebAuthn session	NC-01
Sign to an address removed from the whitelist	Malicious approver: Policy evaluated only at request time	Re-evaluate at authorisation time	NC-02
Obtain a signature on a different transaction	Compromised coordinator: Nodes sign the relayed digest	Nodes rebuild the digest and verify the authorisation	NC-03
Combine an old share with current shares	Former node operator: No refresh on decommission	Refresh shares on membership change	NC-04
Duplicate a transaction request	Network attacker: HMAC without timestamp or nonce	Timestamp, nonce and idempotency keys	NC-05
Forge webhooks	Insider with log access: Secret in URL query	Secret in request body, shown once	NC-06

5

Methodology and severity classification

1. Architecture and trust-boundary review with the engineering team
2. Manual code review of the signing node, policy engine, API and console
3. Authorised penetration test of the staging environment (API, console, network exposure)
4. Dependency and build-provenance review for all services
5. Model checking of the approval and signing protocol in TLA+
6. Infrastructure configuration review (Terraform, Kubernetes, cloud IAM)
7. Remediation review of every fix

Severity classification

Each finding is rated on the impact it would have if exploited and the likelihood that its preconditions occur. The rating reflects the code at the reviewed commit.

Severity	Meaning
■ Critical	Direct loss of funds or permanent loss of control is achievable with little or no precondition.
■ High	Material loss or protocol disruption is achievable under realistic conditions.

Severity	Meaning
■ Medium	Limited loss, degraded functionality, or impact that depends on specific conditions.
■ Low	Minor impact, defence-in-depth gaps, or issues with unlikely preconditions.
■ Informational	Code quality, documentation or best-practice observations without direct security impact.

Remediation status

Status	Meaning
Resolved	Fixed and verified in the remediation review.
Partially resolved	Mitigated; the residual risk is documented.
Acknowledged	Accepted by the protocol team with a stated rationale.
Open	Not yet addressed at the reviewed remediation commit.

6 Findings summary

ID	Title	Severity	Category	Status
NC-01	Approval quorum attributes approvals to the approver named in the request body	■ High	Access control	Resolved
NC-02	Policy is evaluated at request time and not re-evaluated before signing	■ Medium	Business logic	Resolved
NC-03	Signing nodes sign the digest relayed by the coordinator without rebuilding it	■ Medium	Key management	Resolved
NC-04	Decommissioned signing nodes keep valid key shares until the next scheduled refresh	■ Medium	Key management	Resolved
NC-05	API request signatures do not cover a timestamp or nonce, so requests can be replayed	■ Medium	Signature replay	Resolved
NC-06	Webhook signing secret is passed in a URL query string in the console	■ Low	Configuration	Resolved
NC-07	Velocity limits apply per asset rather than across a wallet's total value	■ Low	Business logic	Acknowledged
NC-08	Unmaintained transitive dependency in the signing node	■ Informational	Supply chain	Resolved

7

Detailed findings

NC-01 ■ High Resolved

Approval quorum attributes approvals to the approver named in the request body

services/policy/approvals/handler.go:L74-L121 (RecordApproval)

Category	CWE	Likelihood	Impact
Access control	CWE-639	Medium	High

DESCRIPTION

The console posts approvals to the policy engine as JSON containing `request_id`, `approver_id` and the WebAuthn assertion. `RecordApproval` verifies the assertion against the session's registered credential, then stores the approval under `body.ApproverID`. The verified credential is never compared with `approver_id`.

An approver can therefore submit their own valid assertion several times with different `approver_id` values. Each one is stored as a distinct approval, and the quorum check counts distinct `approver_ids`, so a 3-of-5 policy can be satisfied by one person.

The researcher programme for Northstar lists this outcome as high severity. It does not by itself allow signing outside policy, because whitelists and limits still apply, but quorum is the main control for large or unusual transfers.

PRECONDITIONS

- A valid approver account with a registered hardware key.
- Knowledge of other approvers' IDs, which the console shows in the approval list.

EXPLOIT SCENARIO

1. A malicious approver creates or waits for a large transfer to a whitelisted address that requires three approvals.
2. They approve once in the console, then replay the request three times with the IDs of two colleagues, reusing a fresh assertion from their own key each time.
3. The policy engine records three approvals and issues a signing authorisation.

IMPACT

A single approver can authorise transactions that require several, defeating separation of duties.

RECOMMENDATION

Derive the approver identity from the authenticated session and the credential verified in the assertion; ignore any identity in the request body. Include the request ID and transaction digest in the WebAuthn challenge.

FIX

services/policy/approvals/handler.go (remediation commit)	DIFF
1 - cred, err := h.webauthn.Verify(sess, body.Assertion)	
2 + cred, err := h.webauthn.Verify(sess, body.Assertion, challengeFor(body.RequestID, req.Digest))	
3 if err != nil {	
4 return ErrBadAssertion	
5 }	
6 - return h.store.AddApproval(body.RequestID, body.ApproverID)	
7 + approver, err := h.users.ByCredential(cred.ID)	
8 + if err != nil approver.ID != sess.UserID {	
9 + return ErrApproverMismatch	
10 + }	

```
11 + return h.store.AddApproval(body.RequestID, approver.ID)
```

RESOLUTION

Fixed and verified. The approver is resolved from the verified credential and must match the session; the challenge binds the request and digest. Property P3 holds on the remediated model.

REFERENCES

- CWE-639: Authorization Bypass Through User-Controlled Key
<https://cwe.mitre.org/data/definitions/639.html>
- W3C Web Authentication Level 2
<https://www.w3.org/TR/webauthn-2/>
- OWASP API Security Project
<https://owasp.org/www-project-api-security/>

NC-02 ■ Medium Resolved

Policy is evaluated at request time and not re-evaluated before signing

services/policy/engine/evaluate.go:L38-L96, services/policy/authorise.go:L22-L58

Category	CWE	Likelihood	Impact
Business logic	CWE-367	Low	High

DESCRIPTION

The policy engine evaluates whitelists and velocity limits when a request is created and stores the result. When the quorum is later reached, `Authorise` issues the signing authorisation based on that stored result. Approval rounds can stay open for days.

If a client removes an address from its whitelist, or lowers a limit, during that window (typically because the address is believed to be compromised), the pending request is still signed. Velocity limits are similarly checked against usage at creation time, so several requests created together can each pass and together exceed the limit.

PRECONDITIONS

- A pending request created before a policy change, or several concurrent requests near a limit.

EXPLOIT SCENARIO

1. An attacker who controls a whitelisted counterparty address creates a request to it.
2. The client learns of the compromise and removes the address from its whitelist.
3. Approvers who have not noticed the change complete the quorum, and the transaction is signed to the removed address.

IMPACT

Transfers to addresses the client has explicitly removed, or beyond velocity limits.

RECOMMENDATION

Re-evaluate the full policy, against current usage, inside `Authorise` and reserve velocity usage atomically when the authorisation is issued.

FIX

services/policy/authorise.go (remediation commit)		DIFF
1	func (e *Engine) Authorise(ctx context.Context, req *Request) (*Authorisation, error) {	
2	- if req.Evaluation.Allowed && e.quorumMet(req) {	
3	- return e.issue(req)	
4	- }	

```

5 + tx, err := e.store.BeginSerializable(ctx)
6 + if err != nil {
7 +     return nil, err
8 + }
9 + defer tx.Rollback()
10 + eval := e.Evaluate(ctx, tx, req) // current policy and current usage
11 + if !eval.Allowed || !e.quorumMet(req) {
12 +     return nil, ErrPolicy
13 + }
14 + if err := tx.ReserveUsage(req); err != nil {
15 +     return nil, err
16 + }
17 + return e.issueAndCommit(tx, req)
18 }

```

RESOLUTION

Fixed. Policy and usage are evaluated again inside a serialisable transaction when the authorisation is issued.

REFERENCES

- CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition
<https://cwe.mitre.org/data/definitions/367.html>

NC-03 ■ Medium Resolved

Signing nodes sign the digest relayed by the coordinator without rebuilding it

services/signer/src/session.rs:L142-L205 (start_session)

Category	CWE	Likelihood	Impact
Key management	CWE-345	Low	High

DESCRIPTION

A signing session starts with a message from the coordinator containing the authorisation ID and the 32-byte digest to sign. Each node checks that the authorisation ID exists and is unused, then signs `msg.digest`. Nodes do not fetch the authorised transaction, rebuild the digest themselves, or verify the policy engine's signature over the authorisation.

The coordinator is meant to be a scheduler and relay, but in this design it can choose what is signed. A compromised coordinator can attach any digest to a valid, unused authorisation, for example one for a transfer of the full balance to an attacker address.

The MPC threshold protects against compromise of individual nodes; this finding is about the component in front of them, which was implicitly trusted.

PRECONDITIONS

- Compromise of the coordinator service.
- An authorisation that has been issued but not yet used.

EXPLOIT SCENARIO

1. An attacker gains code execution on the coordinator.
2. When a legitimate authorisation for a small transfer arrives, it substitutes the digest of a transaction draining the wallet.
3. The nodes see a valid, unused authorisation ID and produce the signature.

IMPACT

Arbitrary signatures from any wallet with a pending authorisation, bypassing policy.

RECOMMENDATION

Have each node verify the policy engine's signature over the authorisation, fetch the authorised transaction, rebuild the digest independently with chain-specific encoding, and refuse to sign on mismatch.

FIX

services/signer/src/session.rs (remediation commit)	DIFF
1 pub fn start_session(&self, msg: StartSession) -> Result<Session, SignerError> {	
2 - let auth = self.auths.get_unused(&msg.authorisation_id?);	
3 - self.protocol.begin(auth.key_id, msg.digest)	
4 + let auth = self.auths.get_unused(&msg.authorisation_id?);	
5 + self.policy_key.verify(&auth.canonical_bytes(), &auth.policy_signature?);	
6 + let digest = match auth.chain {	
7 + Chain::Ethereum => eip1559_signing_hash(&auth.transaction)?,	
8 + Chain::Solana => solana_message_bytes(&auth.transaction)?,	
9 + };	
10 + if digest != msg.digest {	
11 + return Err(SignerError::DigestMismatch);	
12 + }	
13 + self.protocol.begin(auth.key_id, digest)	
14 }	

RESOLUTION

Fixed. Each node verifies the authorisation signature and rebuilds the payload to sign; the coordinator can no longer influence what is signed.

REFERENCES

- CWE-345: Insufficient Verification of Data Authenticity
<https://cwe.mitre.org/data/definitions/345.html>
- EIP-1559: Fee market change for ETH 1.0 chain
<https://eips.ethereum.org/EIPS/eip-1559>

NC-04 ■ Medium Resolved

Decommissioned signing nodes keep valid key shares until the next scheduled refresh

services/signer/src/membership.rs:L60-L118, deploy/runbooks/node-replace.md

Category	CWE	Likelihood	Impact
Key management	CWE-324	Low	High

DESCRIPTION

Key shares are refreshed (re-randomised without changing the public key) on a 90-day schedule. Replacing a node adds a new node through resharing but does not trigger a refresh, so the share held by the removed node remains compatible with the current shares until the next scheduled run.

The runbook wipes the removed node's volume, but backups and snapshots of that volume are kept for 30 days. Anyone with access to those snapshots holds a valid share that, combined with threshold minus one current shares, can sign.

PRECONDITIONS

- A node replaced within the current refresh period.
- Access to snapshots of the removed node, plus compromise of further nodes up to the threshold.

EXPLOIT SCENARIO

1. A former operator retains a snapshot of a replaced node. Combined with two compromised current nodes in a 3-of-4 configuration, the old share completes the threshold.

IMPACT

Lowers the effective number of compromises needed to sign during the refresh window.

RECOMMENDATION

Run a proactive share refresh as part of every membership change and exclude signing-node volumes from general backups.

FIX

services/signer/src/membership.rs (remediation commit)		DIFF
1	pub fn replace_node(&mut self, old: NodeId, new: NodeId) -> Result<(), SignerError> {	
2	self.resshare_to(new)?;	
3	self.remove(old)?;	
4	+ self.refresh_all_shares()?; // old share is no longer compatible	
5	+ self.audit.record(MembershipEvent::Refreshed { removed: old });	
6	Ok(())	
7	}	

RESOLUTION

Fixed. Membership changes trigger an immediate refresh, and signing-node volumes are excluded from snapshot policies.

REFERENCES

- NIST SP 800-57 Part 1 Rev. 5: Recommendation for Key Management
<https://csrc.nist.gov/pubs/sp/800/57/pt1/r5/final>
- CWE-324: Use of a Key Past its Expiration Date
<https://cwe.mitre.org/data/definitions/324.html>

NC-05 ■ Medium Resolved

API request signatures do not cover a timestamp or nonce, so requests can be replayed

services/api/src/auth/hmac.ts:L18-L57

Category	CWE	Likelihood	Impact
Signature replay	CWE-294	Medium	Medium

DESCRIPTION

Clients sign requests with `HMAC-SHA256(secret, method + path + body)`. The signature has no timestamp or nonce, and transaction-creation requests have no idempotency key. A captured request can be resent indefinitely and each copy creates a new transaction request.

Replayed requests still need policy approval, so this does not lead to unauthorised signing on its own. It does create duplicate requests that approvers may approve in bulk, and it lets an attacker with log or proxy access exhaust velocity limits to block legitimate transfers.

PRECONDITIONS

- Access to a captured signed request, for example from client-side logs or a TLS-terminating proxy.

EXPLOIT SCENARIO

1. An attacker replays a captured payroll request twenty times. Approvers who batch-approve routine payroll approve the duplicates.

IMPACT

Duplicate transfers if approvers do not notice, and denial of service through velocity-limit exhaustion.

RECOMMENDATION

Include a timestamp and a random nonce in the signed string, reject requests outside a short window or with a seen nonce, and require idempotency keys on creation endpoints.

FIX

services/api/src/auth/hmac.ts (remediation commit)	DIFF
<pre>1 - const payload = req.method + req.path + req.rawBody 2 + const ts = Number(req.header('x-ns-timestamp')) 3 + const nonce = req.header('x-ns-nonce') ?? '' 4 + if (!Number.isFinite(ts) Math.abs(Date.now() / 1000 - ts) > 300) throw new AuthError('stale') 5 + if (!(await nonces.claim(keyId, nonce, 600))) throw new AuthError('replayed') 6 + const payload = [req.method, req.path, ts, nonce, sha256Hex(req.rawBody)].join('\n')</pre>	

RESOLUTION

Fixed. Signatures cover timestamp and nonce with a five-minute window; creation endpoints require an idempotency key.

REFERENCES

- CWE-294: Authentication Bypass by Capture-replay
<https://cwe.mitre.org/data/definitions/294.html>
- RFC 2104: HMAC
<https://www.rfc-editor.org/rfc/rfc2104>

NC-06 ■ Low Resolved

Webhook signing secret is passed in a URL query string in the console

apps/console/src/settings/webhooks.tsx:L88, services/api/src/routes/webhooks.ts:L41

Category	CWE	Likelihood	Impact
Configuration	CWE-598	Low	Medium

DESCRIPTION

The webhook settings page loads the signing secret with `GET /v1/webhooks/{id}?reveal=true&secret=...` when testing a webhook, so the secret appears in browser history and in the access logs of the API gateway. Anyone who can read those logs can forge webhooks that clients use to update their own ledgers.

PRECONDITIONS

- Read access to gateway logs or the approver's browser history.

EXPLOIT SCENARIO

1. An insider with log access forges a "transaction confirmed" webhook to a client's back office.

IMPACT

Forged webhook notifications; no effect on signing.

RECOMMENDATION

Send secrets only in request bodies, show them once at creation, and support rotation.

RESOLUTION

Fixed. Secrets are shown once, test calls use a POST body, and logs were purged.

REFERENCES

- CWE-598: Use of GET Request Method With Sensitive Query Strings
<https://cwe.mitre.org/data/definitions/598.html>

NC-07 ■ Low Acknowledged

Velocity limits apply per asset rather than across a wallet's total value

services/policy/engine/velocity.go:L30-L84

Category	CWE	Likelihood	Impact
Business logic	—	Low	Medium

DESCRIPTION

Velocity limits are configured per asset (for example 500,000 USDC per day). A wallet holding many assets has no aggregate limit, so an attacker who can pass approvals can move far more value in one day by spreading transfers across assets than any single limit suggests. Clients may not realise this when configuring policies.

PRECONDITIONS

- A wallet holding several assets and a path to approval.

EXPLOIT SCENARIO

1. Transfers just under each asset's limit move the value of eight assets in one day.

IMPACT

Limits give less protection than clients expect.

RECOMMENDATION

Offer an aggregate limit in a reference currency and warn about policies without one.

RESOLUTION

Acknowledged. Aggregate limits and policy linting are planned; in the meantime the console shows the combined daily exposure.

REFERENCES

- OWASP Application Security Verification Standard
<https://owasp.org/www-project-application-security-verification-standard/>

NC-08 ■ Informational Resolved

Unmaintained transitive dependency in the signing node

services/signer/Cargo.lock

Category	CWE	Likelihood	Impact
Supply chain	CWE-1104	Low	Low

DESCRIPTION

`cargo audit` reported an unmaintained crate pulled in by a logging dependency. No vulnerability was known in it, but unmaintained code in the signing node increases long-term risk.

PRECONDITIONS

- None.

EXPLOIT SCENARIO

1. Not exploitable at the time of review.

IMPACT

Supply-chain hygiene.

RECOMMENDATION

Replace the dependency and run `cargo audit` and `cargo deny` in CI.

RESOLUTION

Fixed; dependency replaced and checks added to CI.

REFERENCES

- RustSec Advisory Database
<https://rustsec.org/>

8

Severity and risk analysis

The matrix places each finding by the likelihood of its preconditions and the impact of exploitation. Identifiers in each cell refer to the detailed findings in section 7.

Impact

	Low	Medium	High
High	3 NC-02 NC-03 NC-04	1 NC-01	0
Medium	2 NC-06 NC-07	1 NC-05	0
Low	1 NC-08	0	0
	Low	Medium	High

Likelihood

Distribution by vulnerability class

Business logic	2
Key management	2
Access control	1
Configuration	1
Signature replay	1
Supply chain	1

Severity distribution



9

Remediation status

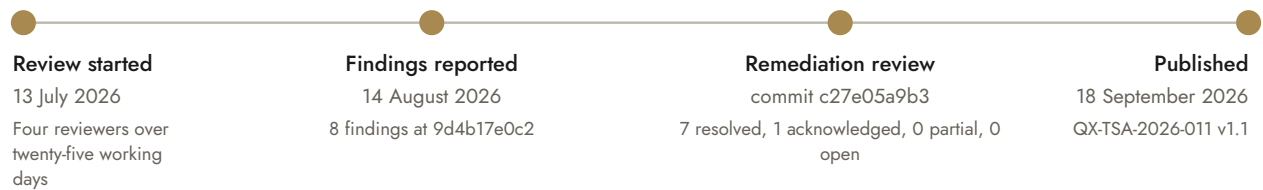
7 of 8 findings are resolved, 1 acknowledged, 0 partially resolved and 0 open, as reviewed at remediation commit c27e05a9b3f1.

Severity	Resolved	Partially resolved	Acknowledged	Open	Total
■ Critical	0	0	0	0	0
■ High	1	0	0	0	1
■ Medium	4	0	0	0	4
■ Low	1	0	1	0	2
■ Informational	1	0	0	0	1

Remediation progress

100%

Remediation history



Residual items

ID	Status	Notes
NC-07	Acknowledged	Acknowledged. Aggregate limits and policy linting are planned; in the meantime the console shows the combined daily exposure.

10

Formal verification and invariants

The approval and signing protocol was modelled in TLA+ and checked with TLC for three approvers, two policies, four signing nodes (threshold three) and up to two concurrent requests. The model of the reviewed version produced a counterexample to the distinct-approver property (NC-01) and to policy freshness (NC-02); the remediated model satisfies all safety properties within these bounds.

Tool	TLA+ with the TLC model checker
------	---------------------------------

ID	Property	Result	Notes
P1	A signature is produced only for a transaction whose authorisation was issued after a successful policy evaluation and quorum.	■ Proved	Exhaustive within the model bounds; counterexample on the reviewed version (NC-02).
P2	Each signing session produces at most one signature, over the digest named in its authorisation.	■ Proved	Within bounds, with nodes rebuilding the digest (NC-03 fix).
P3	A quorum of k counts k distinct authenticated approvers.	■ Proved	Remediated model; counterexample on the reviewed version (NC-01).
P4	Every approved request is eventually signed or rejected under fair scheduling.	■ Inconclusive	Liveness check did not complete within the state-space budget for two concurrent requests.

11

Recommendations

Keep the coordinator untrusted

Treat the coordinator purely as a relay. Any new message type it sends to nodes should be authenticated by the policy engine and validated independently by each node.

Bind approvals to what is shown

Approvers should sign a challenge derived from the decoded transaction they see. The NC-01 fix does this; extend it to batch approvals.

Cross-check chain data

Simulate transactions against two independent RPC providers before showing them to approvers, so a single provider cannot mislead them.

Lint client policies

Warn clients about wildcard whitelists, missing aggregate limits and single-approver quorums (NC-07).

Rehearse node replacement

Include the post-replacement share refresh (NC-04) in the scheduled recovery exercise.

12

Limitations

- The penetration test covered the staging environment only; production was reviewed through configuration and code, not tested directly.
- The threshold-signature library was treated as correct; only its integration and parameters were reviewed.
- Model checking is exhaustive only within the stated bounds.

No assessment can show that a system is free of vulnerabilities. Conclusions apply to the code at the commits listed and the conditions stated in this document.

Appendix

Technical appendix

A.1 Tools

Tool	Purpose
TLA+ / TLC	Model checking of the approval and signing protocol
cargo audit, cargo deny	Rust dependency review
govulncheck	Go vulnerability scanning
Semgrep	Custom rules for TypeScript and Go request handling
Burp Suite	API and console penetration testing in staging
Checkov	Terraform and Kubernetes configuration review

A.2 Reviewed files (SHA-256)

File	SHA-256
services/signer/src/session.rs	7216a8571a8f7ecd838f2c73b4149fcb0d42d93e348c41dc64c3f15e452cbdbd
services/signer/src/membership.rs	d4578b0b349004abb1596ec769bd2a4bc69d4f52cd54c8502d2dafa3a4854dfb
services/policy/approvals/handler.go	51584eb395cf17e1722a7ebfdfb98d4cdd1d844b8095985e6f23b03b4192a07b
services/policy/authorise.go	59c016a429a641e93ed9da465834ff215756a065aee10b77fbeb0836818f138b
services/api/src/auth/hmac.ts	204ebdc3ce9b2572028ed478f3d5c382eee02f0231f94c1e0e195bfa9fb8168b

A.3 Notes

- Penetration testing was performed against staging hosts under the *.northstar-custody.example test domain with written authorisation.
- File hashes are SHA-256 digests of the reviewed sources at the audited commit.

A.4 Methodology and technical references

Standards, specifications and weakness classifications cited in the findings. Each entry links to the official source.

1. CWE-639: Authorization Bypass Through User-Controlled Key
<https://cwe.mitre.org/data/definitions/639.html>
2. W3C Web Authentication Level 2
<https://www.w3.org/TR/webauthn-2/>
3. OWASP API Security Project
<https://owasp.org/www-project-api-security/>
4. CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition
<https://cwe.mitre.org/data/definitions/367.html>
5. CWE-345: Insufficient Verification of Data Authenticity
<https://cwe.mitre.org/data/definitions/345.html>
6. EIP-1559: Fee market change for ETH 1.0 chain
<https://eips.ethereum.org/EIPS/eip-1559>

7. NIST SP 800-57 Part 1 Rev. 5: Recommendation for Key Management
<https://csrc.nist.gov/pubs/sp/800/57/pt1/r5/final>
8. CWE-324: Use of a Key Past its Expiration Date
<https://cwe.mitre.org/data/definitions/324.html>
9. CWE-294: Authentication Bypass by Capture-replay
<https://cwe.mitre.org/data/definitions/294.html>
10. RFC 2104: HMAC
<https://www.rfc-editor.org/rfc/rfc2104>
11. CWE-598: Use of GET Request Method With Sensitive Query Strings
<https://cwe.mitre.org/data/definitions/598.html>
12. OWASP Application Security Verification Standard
<https://owasp.org/www-project-application-security-verification-standard/>
13. RustSec Advisory Database
<https://rustsec.org/>

A.5 Document information

Document	Quantix Technical Security Assessment QX-TSA-2026-011, version 1.1
Subject	Northstar Custody, Quantix Protocol Research
Status	Published 18 September 2026
Disclosure	This document is a Quantix research assessment of Northstar Custody, a protocol design from Quantix Protocol Research. It is not a client engagement, it does not describe a deployed system, and it is not a certification.