



TECHNICAL SECURITY ASSESSMENT

Larkspur Wallet SDK

Penetration Test · Wallet Infrastructure

REPORT ID QX-TSA-2026-004

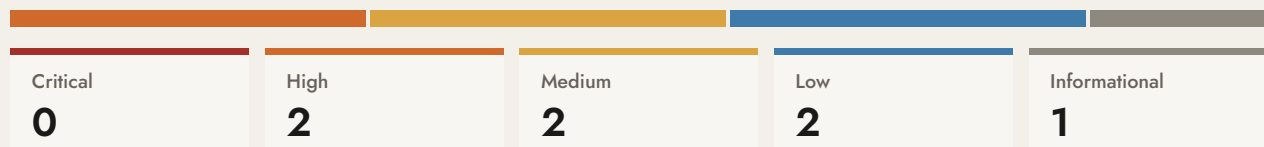
QUANTIX RESEARCH ASSESSMENT

Signing SDK, wallet connector, pairing relay API and the staging environment that hosts it.

Document status	Published
Published	9 September 2026
Review period	Ecosystems
17 August 2026 – 28 August 2026	Cosmos
Languages	Review effort
TypeScript, Go	Two testers, ten working days (source-assisted)
Reviewed commit	Remediation commit
ba74b61eb568ebb8ecd6232af9220a373971946e	de6deb2f8980e501c9ac20f57c8369eb1240bdc9

FINDINGS BY SEVERITY

7 findings · 100% remediated



This document is a Quantix research assessment of Larkspur Wallet SDK, 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-2026-004
Title	Technical Security Assessment: Larkspur Wallet SDK
Version	1.1
Status	Published
Assessment type	Penetration Test
Review period	17 August 2026 – 28 August 2026
Published	9 September 2026
Reviewed commit	ba74b61eb568ebb8ecd6232af9220a373971946e

CONTENTS

1	Executive summary	3
2	Scope	4
3	Protocol architecture	4
4	Threat model	5
5	Methodology and severity classification	6
6	Findings summary	7
7	Detailed findings	8
	LW-01 Displayed transaction summary is not derived from the signed payload	8
	LW-02 Pairing identifiers are short and not bound to the requesting origin	9
	LW-03 Relay reflects any Origin with credentials allowed	10
	LW-04 Chain registry responses are not integrity-protected	11
	LW-05 Pairing and join endpoints have no rate limiting	12
	LW-06 Error responses include stack traces	12
	LW-07 Development dependencies with published advisories	13
8	Severity and risk analysis	14
9	Remediation status	14
10	Formal verification and invariants	15
11	Recommendations	16
12	Limitations	16
A	Technical appendix	17

1

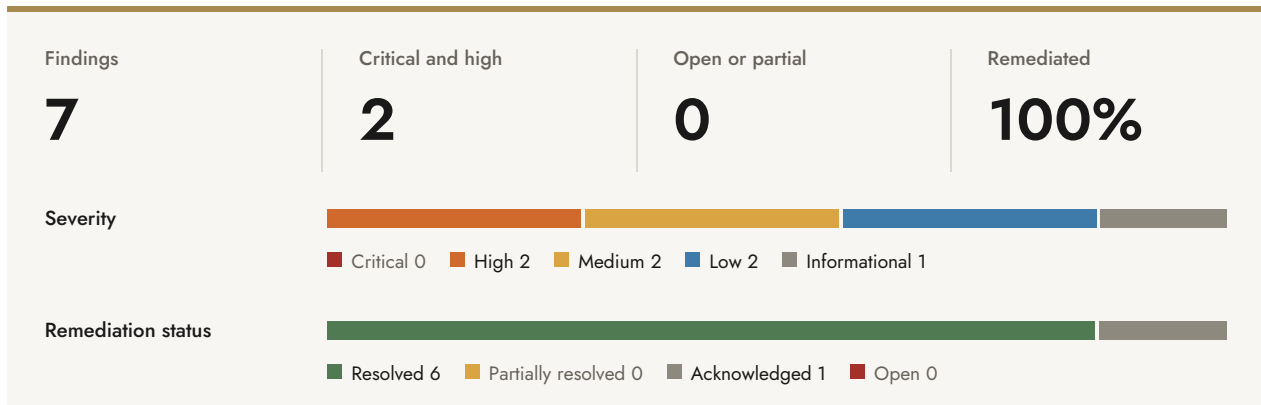
Executive summary

Larkspur is a signing SDK that lets Cosmos dApps request signatures from wallets, with a Go relay that pairs a dApp session to a wallet and forwards encrypted messages between them. A wallet SDK sits on the path between what a user sees and what they sign, so the test concentrated on whether a malicious or compromised party can make those two diverge.

Seven findings were reported. The two high-severity issues were a human-readable transaction summary that was built from caller-supplied metadata instead of the bytes being signed, and short pairing identifiers that could be guessed and were not bound to the requesting origin. Together they could let a hostile page present one transaction and obtain a signature for another.

All high and medium findings were fixed and re-tested against the remediation commit. Two low findings were fixed; an informational note about development dependencies was acknowledged.

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.	2	0
■ Medium	Limited loss, degraded functionality, or impact that depends on specific conditions.	2	0
■ Low	Minor impact, defence-in-depth gaps, or issues with unlikely preconditions.	2	0
■ Informational	Code quality, documentation or best-practice observations without direct security impact.	1	0

2

Scope

Signing SDK, wallet connector, pairing relay API and the staging environment that hosts it.

Repository	github.com/example-org/larkspur-wallet
Reviewed commit	ba74b61eb568ebb8ecd6232af9220a373971946e
Remediation commit	de6deb2f8980e501c9ac20f57c8369eb1240bdc9
Review period	17 August 2026 – 28 August 2026
Review effort	Two testers, ten working days (source-assisted)

Path	Description	nSLOC
packages/sdk-core	Sign-request construction (Amino and Direct), response verification, display summaries	3,140
packages/connect	Wallet connector, session handling and chain registry client	1,860
services/relay-api	Go pairing relay: session pairing, encrypted message forwarding, HTTP API	2,420
staging.larkspur.example	Staging deployment of the relay API and registry (black-box testing)	–
Total		7,420

Out of scope

- Third-party wallet applications that implement the protocol
- Production infrastructure and cloud account configuration
- Denial-of-service testing beyond basic rate-limit checks
- Social engineering

3

Protocol architecture

A dApp imports the SDK and calls `requestSignature` with a transaction. The SDK serialises the transaction into an Amino JSON or Protobuf `SignDoc`, produces a human-readable summary, and sends both to the wallet through the pairing relay.

Pairing starts when the dApp asks the relay for a pairing ID, shows it as a QR code or deep link, and the wallet joins with the same ID. After a key exchange, messages are end-to-end encrypted; the relay sees only ciphertext and routing metadata.

Chain metadata — chain IDs, fee denominations and RPC endpoints — is fetched by the connector from a registry service and cached in the browser.

Component	Role	Trust level
Signing SDK	Builds sign requests and display summaries; verifies signatures	Trusted
Wallet connector	Session lifecycle and chain registry client	Trusted
Pairing relay	Pairs sessions and forwards encrypted messages	Semi-trusted
Chain registry	Serves chain metadata to the connector	Semi-trusted
Integrating dApp page	Calls the SDK; may be malicious or compromised	Untrusted
Wallet application	Displays the request and signs	Trusted
Staging cloud	Build pipeline and relay hosting	Semi-trusted

Trust assumptions

- The relay is not trusted with message content; end-to-end encryption must hold even if the relay is compromised.
- The wallet displays what the SDK sends; the SDK must therefore derive every displayed value from the signed bytes.
- Integrating pages may be hostile, so origin must be bound into pairing and session state.

4 Threat model

Assets

- User signatures and the transactions they authorise
- Session keys for paired dApp–wallet sessions
- Integrity of chain metadata used to build transactions

Actors

Actor	Capability	Trust
Malicious dApp page	Calls the SDK with arbitrary arguments and metadata	Untrusted
Network attacker	Observes or guesses pairing identifiers; sends cross-origin requests	Untrusted
Compromised relay operator	Reads routing metadata and drops or reorders messages	Semi-trusted
Registry tampering	Serves modified chain metadata	Untrusted

Threats

Threat	Actor and vector	Mitigation	Findings
Blind signing: summary differs from signed bytes	Malicious dApp page: Metadata passed alongside the SignDoc	Derive summary from decoded SignDoc	LW-01
Session hijack through guessed pairing ID	Network attacker: Enumerating short pairing IDs	128-bit IDs bound to origin	LW-02

Threat	Actor and vector	Mitigation	Findings
Cross-origin session API access	Network attacker: Reflected CORS with credentials	Origin allowlist	LW-03
Transactions built against a malicious chain config	Registry tampering: Unsigned registry responses	Signed registry manifest	LW-04
Pairing enumeration at scale	Network attacker: Unthrottled pairing endpoints	Rate limiting	LW-05

5

Methodology and severity classification

1. Source-assisted review of the SDK sign path: serialisation, summary generation and signature verification
2. Black-box and grey-box testing of the relay API on staging, following the OWASP Web Security Testing Guide
3. Session and pairing analysis: identifier entropy, origin binding, replay and expiry
4. Supply-chain review of SDK dependencies and build output
5. Re-test of every finding against the remediation commit on a fresh staging deployment

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.
■ 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
LW-01	Displayed transaction summary is not derived from the signed payload	High	Business logic	Resolved
LW-02	Pairing identifiers are short and not bound to the requesting origin	High	Cryptography	Resolved
LW-03	Relay reflects any Origin with credentials allowed	Medium	Configuration	Resolved
LW-04	Chain registry responses are not integrity-protected	Medium	Supply chain	Resolved
LW-05	Pairing and join endpoints have no rate limiting	Low	Denial of service	Resolved
LW-06	Error responses include stack traces	Low	Configuration	Resolved
LW-07	Development dependencies with published advisories	Informational	Supply chain	Acknowledged

7

Detailed findings

LW-01 ■ High Resolved

Displayed transaction summary is not derived from the signed payload

packages/sdk-core/src/sign/request.ts:L64-L118

Category	CWE	Likelihood	Impact
Business logic	CWE-451	Medium	High

DESCRIPTION

`buildSignRequest` accepts a transaction and an optional `display` object. When `display` is present, its recipient, amount and memo are used for the human-readable summary sent to the wallet; the `SignDoc` that will actually be signed is serialised separately from the transaction argument.

Nothing checks that the two agree. A page that controls both arguments can show the user "Send 1 ATOM to cosmos1...abc" while the `SignDoc` contains a `MsgSend` of the full balance to a different address, or an `authz MsgGrant`.

Wallets that trust the SDK summary, which is the intended integration pattern, display the false summary next to the approval button.

PRECONDITIONS

- The user connects a wallet to a malicious or compromised page
- The wallet displays the SDK-provided summary

EXPLOIT SCENARIO

1. A compromised dApp page calls `buildSignRequest` with a `MsgGrant` giving the attacker `authz` over the user's funds.
2. It passes `display = { recipient: <the expected address>, amount: "1 ATOM" }`.
3. The wallet shows the small transfer; the user approves.
4. The signed `MsgGrant` lets the attacker execute transfers on the user's behalf.

IMPACT

Users can be induced to sign transactions materially different from what they were shown, up to loss of all funds controlled by the key.

RECOMMENDATION

Remove the `display` override. Generate the summary by decoding the exact bytes of the `SignDoc` that will be signed, render unknown message types explicitly as unrecognised, and add differential tests that decode each request independently.

FIX

packages/sdk-core/src/sign/request.ts	DIFF
1 - export function buildSignRequest(tx: TxInput, display?: DisplayHints): SignRequest {	
2 - const signDoc = serializeSignDoc(tx)	
3 - const summary = display ? summaryFromHints(display) : summaryFromTx(tx)	
4 + export function buildSignRequest(tx: TxInput): SignRequest {	
5 + const signDoc = serializeSignDoc(tx)	
6 + // The summary is always decoded from the bytes that will be signed.	
7 + const summary = summaryFromSignDoc(decodeSignDoc(signDoc.bytes))	
8 return { signDoc, summary }	

9 }

RESOLUTION

Display hints removed from the public API; the summary is decoded from SignDoc bytes. Unknown message types are rendered as "Unrecognised message" with the raw type URL. Differential tests (DT-01) added.

REFERENCES

- Cosmos SDK: transaction signing modes
<https://docs.cosmos.network/main/learn/advanced/transactions>
- CWE-451: User Interface Misrepresentation of Critical Information
<https://cwe.mitre.org/data/definitions/451.html>

LW-02 ■ High Resolved

Pairing identifiers are short and not bound to the requesting origin

services/relay-api/internal/pairing/id.go:L12

Category	CWE	Likelihood	Impact
Cryptography	CWE-330	Medium	High

DESCRIPTION

Pairing IDs are 32-bit values from `math/rand` encoded in base32. With many pairings open at once, an attacker who enumerates IDs has a realistic chance of joining another user's pending pairing before the wallet does.

The relay also does not record the origin that requested the pairing, so a wallet joining a pairing cannot tell whether it was created by the page the user is looking at. `math/rand` is not a cryptographic generator.

PRECONDITIONS

- Pending pairings exist on the relay
- The attacker can send requests to the relay (it is public)

EXPLOIT SCENARIO

1. The attacker repeatedly calls the join endpoint with random 32-bit IDs.
2. A guess hits a pending pairing created by a victim's dApp session.
3. The attacker completes the key exchange before the victim's wallet and receives sign requests intended for the victim, or injects responses.

IMPACT

Session hijacking: the attacker can read sign requests and, combined with LW-01, present false summaries to a victim wallet.

RECOMMENDATION

Generate 128-bit pairing IDs with `crypto/rand`, bind the requesting origin into the pairing record and the key-exchange transcript, and expire unjoined pairings within minutes.

FIX

services/relay-api/internal/pairing/id.go	DIFF
1	-import "math/rand"
2	+import "crypto/rand"
3	
4	-func NewID() string {
5	- return base32.StdEncoding.EncodeToString(uint32Bytes(rand.Uint32()))
6	+func NewID() (string, error) {

```

7 +   b := make([]byte, 16)
8 +   if _, err := rand.Read(b); err != nil {
9 +       return "", err
10 +   }
11 +   return base32.StdEncoding.WithPadding(base32.NoPadding).EncodeToString(b), nil
12 + }

```

RESOLUTION

128-bit IDs from crypto/rand; origin stored in the pairing record and included in the key-exchange transcript; unjoined pairings expire after five minutes.

REFERENCES

- Go crypto/rand package
<https://pkg.go.dev/crypto/rand>
- CWE-330: Use of Insufficiently Random Values
<https://cwe.mitre.org/data/definitions/330.html>
- OWASP Session Management Cheat Sheet
https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

LW-03 ■ Medium Resolved

Relay reflects any Origin with credentials allowed

services/relay-api/cmd/server/cors.go:L8

Category	CWE	Likelihood	Impact
Configuration	CWE-942	Medium	Medium

DESCRIPTION

The CORS middleware copies the request Origin header into Access-Control-Allow-Origin and sets Access-Control-Allow-Credentials: true. Any website can therefore make credentialed requests to the session-management endpoints and read the responses.

PRECONDITIONS

- A user with an active relay session visits an attacker-controlled site

EXPLOIT SCENARIO

- The attacker site issues fetch() to the session list endpoint with credentials: "include".
- The browser sends the session cookie and exposes the response to the attacker.

IMPACT

Disclosure of session metadata and the ability to close or modify sessions from any origin.

RECOMMENDATION

Allow only configured dApp origins, and do not combine credentials with dynamic origins.

FIX

	services/relay-api/cmd/server/cors.go	DIFF
1	- w.Header().Set("Access-Control-Allow-Origin", r.Header.Get("Origin"))	
2	- w.Header().Set("Access-Control-Allow-Credentials", "true")	
3	+ if origin := r.Header.Get("Origin"); cfg.AllowedOrigins.Contains(origin) {	
4	+ w.Header().Set("Access-Control-Allow-Origin", origin)	
5	+ w.Header().Set("Vary", "Origin")	
6	+ }	

RESOLUTION

Allowlist from configuration; credentials removed from cross-origin responses because the API moved to bearer tokens.

REFERENCES

- MDN: Cross-Origin Resource Sharing
<https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>
- CWE-942: Permissive Cross-domain Policy with Untrusted Domains
<https://cwe.mitre.org/data/definitions/942.html>

LW-04 ■ Medium Resolved

Chain registry responses are not integrity-protected

packages/connect/src/registry.ts:L22-L47

Category	CWE	Likelihood	Impact
Supply chain	CWE-494	Low	High

DESCRIPTION

The connector fetches chain metadata — chain ID, fee denominations, RPC and REST endpoints — from the registry over HTTPS and caches it in `localStorage` without any signature or pinning. A compromised registry, or any script able to write to the page's storage, can redirect RPC traffic or change the chain ID and fee denomination used to build transactions.

PRECONDITIONS

- Registry compromise, or script execution on the integrating origin

EXPLOIT SCENARIO

1. A modified registry entry points the RPC endpoint at an attacker node.
2. The attacker node returns false account numbers and balances, shaping what the user signs.

IMPACT

Transactions can be built against attacker-chosen metadata, and balance displays can be falsified.

RECOMMENDATION

Sign the registry manifest with a key held offline and verify it in the connector before use; re-verify cached entries on load.

FIX

packages/connect/src/registry.ts	DIFF
1 - const chains = await (await fetch(REGISTRY_URL)).json()	
2 + const { manifest, signature } = await (await fetch(REGISTRY_URL)).json()	
3 + if (!(await verifyManifest(manifest, signature, REGISTRY_PUBLIC_KEY))) {	
4 + throw new RegistryIntegrityError()	
5 + }	
6 + const chains = manifest.chains	

RESOLUTION

Manifest signed with an Ed25519 key; the public key ships in the SDK. Cached entries are re-verified on load.

REFERENCES

- Cosmos chain registry
<https://github.com/cosmos/chain-registry>

- CWE-494: Download of Code Without Integrity Check
<https://cwe.mitre.org/data/definitions/494.html>

LW-05 ■ Low Resolved

Pairing and join endpoints have no rate limiting

[services/relay-api/internal/http/routes.go:L31](#)

Category	CWE	Likelihood	Impact
Denial of service	CWE-770	Medium	Low

DESCRIPTION

The pairing creation and join endpoints accept unlimited requests per client. This made the LW-02 enumeration practical and allows resource exhaustion through mass pairing creation.

PRECONDITIONS

- Network access to the relay

EXPLOIT SCENARIO

1. An attacker scripts pairing creation in a loop, filling the pairing table until legitimate pairings fail.

IMPACT

Degraded relay availability and support for enumeration attacks.

RECOMMENDATION

Apply per-IP and per-origin token-bucket limits to pairing creation and join, and cap pending pairings per origin.

RESOLUTION

Token-bucket limits added at the HTTP layer; pending pairings capped per origin.

REFERENCES

- CWE-770: Allocation of Resources Without Limits or Throttling
<https://cwe.mitre.org/data/definitions/770.html>

LW-06 ■ Low Resolved

Error responses include stack traces

[services/relay-api/internal/http/errors.go:L19](#)

Category	CWE	Likelihood	Impact
Configuration	CWE-209	Medium	Low

DESCRIPTION

Unhandled errors return the Go stack trace in the response body, disclosing internal package paths, dependency versions and file layout.

PRECONDITIONS

- Any request that triggers an unhandled error

EXPLOIT SCENARIO

1. A malformed join request returns a stack trace that reveals the router and dependency versions.

IMPACT

Information disclosure that assists further attacks.

RECOMMENDATION

Return generic error bodies with a correlation ID and log details server-side only.

RESOLUTION

Generic error bodies with correlation IDs; details logged server-side.

REFERENCES

- CWE-209: Generation of Error Message Containing Sensitive Information
<https://cwe.mitre.org/data/definitions/209.html>

LW-07 ■ Informational Acknowledged

Development dependencies with published advisories

package-lock.json

Category	CWE	Likelihood	Impact
Supply chain	—	Low	Low

DESCRIPTION

Several development-only packages had published advisories at the time of testing. None ship in the SDK bundle, but they run in CI with access to the publishing token.

PRECONDITIONS

- A malicious or vulnerable development dependency executes in CI

EXPLOIT SCENARIO

1. A compromised build tool exfiltrates the npm publishing token during CI.

IMPACT

No runtime impact; supply-chain exposure in the build pipeline.

RECOMMENDATION

Update affected packages, enable automated dependency updates, and scope CI publishing tokens to the publish job only.

RESOLUTION

Acknowledged. Automated dependency updates enabled; publishing moved to a separate job with a scoped token.

REFERENCES

- npm docs: npm audit
<https://docs.npmjs.com/cli/commands/npm-audit>

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	1 LW-04	2 LW-01 LW-02	0
Medium	0	1 LW-03	0
Low	1 LW-07	2 LW-05 LW-06	0
	Low	Medium	High

Likelihood

Distribution by vulnerability class

Configuration	2
Supply chain	2
Business logic	1
Cryptography	1
Denial of service	1

Severity distribution



9

Remediation status

6 of 7 findings are resolved, 1 acknowledged, 0 partially resolved and 0 open, as reviewed at remediation commit de6deb2f8980.

Severity	Resolved	Partially resolved	Acknowledged	Open	Total
Critical	0	0	0	0	0
High	2	0	0	0	2

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

Remediation progress

100%

Remediation history



Residual items

ID	Status	Notes
LW-07	Acknowledged	Acknowledged. Automated dependency updates enabled; publishing moved to a separate job with a scoped token.

10

Formal verification and invariants

Not applicable to this engagement. Security properties of the sign path were tested with differential tests that decode each SignDoc independently and compare against the displayed summary.

ID	Property	Result	Notes
DT-01	Every field shown in the summary equals the corresponding field of the decoded SignDoc	■ Not in scope	Checked by differential tests, not proved. Failed on the original commit (LW-01); passes after the fix.

11

Recommendations

Make "what you see is what you sign" a tested property

Keep the differential tests from DT-01 in CI and extend them to every supported message type. Treat any summary not derived from the signed bytes as a security bug.

Bind origin end to end

Carry the requesting origin through pairing, key exchange and every sign request so wallets can display it and refuse mismatches.

Harden the release pipeline

Use provenance attestations for published packages and require two-person review for releases of the signing SDK.

12

Limitations

- Testing was performed against the staging environment; production infrastructure was not tested.
- Third-party wallets that implement the protocol were not assessed.
- Denial-of-service testing was limited to basic rate-limit checks.

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
Intercepting proxy	Manual API and session testing
Differential decoder	Independent SignDoc decoding for summary comparison
Dependency audit	Advisory review of npm and Go modules

A.2 Reviewed files (SHA-256)

File	SHA-256
packages/sdk-core/src/sign/request.ts	175afbdfded2d6cec378c725c90d280798187647a040aaace8704af5c7f0f0ec6
packages/connect/src/registry.ts	f4ba6479172ef81f802a44a0e4c5ef9a45878c1487c3dec0e9fd7e1c4146cd19
services/relay-api/internal/pairing/id.go	d840dc93ca602b306dd6c38fcc2f586f62e01aff4d74fe5433bfc527a08d6520
services/relay-api/cmd/server/cors.go	fa0856dbd40c66605f27f7eb66dac41a398ef5caa6378c51bbab2734e9fb3b91

A.3 Notes

- Testing followed the OWASP Web Security Testing Guide for the relay API (<https://owasp.org/www-project-web-security-testing-guide/>).

A.4 Methodology and technical references

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

1. Cosmos SDK: transaction signing modes
<https://docs.cosmos.network/main/learn/advanced/transactions>
2. CWE-451: User Interface Misrepresentation of Critical Information
<https://cwe.mitre.org/data/definitions/451.html>
3. Go crypto/rand package
<https://pkg.go.dev/crypto/rand>
4. CWE-330: Use of Insufficiently Random Values
<https://cwe.mitre.org/data/definitions/330.html>
5. OWASP Session Management Cheat Sheet
https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html
6. MDN: Cross-Origin Resource Sharing
<https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>
7. CWE-942: Permissive Cross-domain Policy with Untrusted Domains
<https://cwe.mitre.org/data/definitions/942.html>
8. Cosmos chain registry
<https://github.com/cosmos/chain-registry>
9. CWE-494: Download of Code Without Integrity Check
<https://cwe.mitre.org/data/definitions/494.html>

10. CWE-770: Allocation of Resources Without Limits or Throttling
<https://cwe.mitre.org/data/definitions/770.html>
11. CWE-209: Generation of Error Message Containing Sensitive Information
<https://cwe.mitre.org/data/definitions/209.html>
12. npm docs: npm audit
<https://docs.npmjs.com/cli/commands/npm-audit>

A.5 Document information

Document	Quantix Technical Security Assessment QX-TSA-2026-004, version 1.1
Subject	Larkspur Wallet SDK, Quantix Protocol Research
Status	Published 9 September 2026
Disclosure	This document is a Quantix research assessment of Larkspur Wallet SDK, 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.