Building on Quicksand

Dependency Security Foundations

Daniel 'phrawzty' Maher
Senior Security Advocate, Datadog

Building on Quicksand: Dependency Security Foundations

What systematic assessment of major registries actually found

About me

- Senior Security Advocate at Datadog
- Previously: large-scale production systems at Ubisoft and Mozilla
- Slide minimalist

A note on the research

- This research is not yet publicly released
- The questions and methodology are still being refined
- More registries to come
- Goal is a community-driven model: open submissions, collective assessment, fewer individual biases

"crates.io is probably more secure than npm"

- Rust: a language built around security and correctness
- npm: repeated high-profile supply chain incidents
- crates.io: smaller, more curated, community-governed
- ...right?

Software Supply Chain Security Scorecard (SSCSS)

- 52 questions across 8 security domains
- Questions are concrete and enforceable
- "Partially implemented" is not a security boundary
- Assessed 6 registries and counting...
  - npm, PyPI, RubyGems, crates.io, Packagist, Hex.pm

(yes it is pronounced suss cuss)

Eight domains (sections)

Section	Max
S1: Identity & Authentication (Human)	4
S2: Automated Publishing Security (Machine)	7
S3: Authorisation & Access Control	7
S4: Build & Provenance	8
S5: Integrity & Signing	6
S6: Vulnerability Management	7
S7: Abuse Prevention & Response	6
S8: Transparency & Auditability	7

Headline scores

Registry	Score	%
npm	44/52	85%
PyPI	32/52	62%
RubyGems	31/52	60%
crates.io	24/52	46%
Hex.pm	24/52	46%
Packagist	18/52	35%

Section-by-section: the full picture

Domain	npm	PyPI	RubyGems	crates.io	Hex.pm	Packagist	Max
S1 Auth (Human)	3	3	0	1	0	0	4
S2 Auth (Machine)	6	4	4	4	2	2	7
S3 Access Control	6	4	4	4	5	4	7
S4 Build & Provenance	6	4	3	2	2	2	8
S5 Integrity & Signing	5	5	6	2	3	2	6
S6 Vulnerability Mgmt	7	3	4	3	4	4	7
S7 Abuse Prevention	6	5	5	5	5	2	6
S8 Transparency	5	4	5	3	3	2	7

S2, S3, S6, S7: the steadier ground

- S2 (Machine auth): npm 6/7, improving via OIDC trusted publishing across the field
- S3 (Access control): broadly consistent 4–6/7; namespace protection widespread
- S6 (Vulnerability mgmt): npm perfect (7/7); most others 3–4/7; active scanning the differentiator
- S7 (Abuse prevention): 5 of 6 registries score 5–6/6 — visible failures drove investment
    - n.b. Packagist (2/6)

npm: the best-in-class gaps

- S1: 3/4 — MFA and verified identity, but no enforced SSO for organisations
- S4: 6/8 — no reproducible builds, no isolated build environments
- S8: 5/7 — no public audit logs of account events, no security metrics published

crates.io: the security-culture paradox

- Rust has a strong security engineering culture
- crates.io scores 24/52 — tied with Hex.pm, near the bottom of the field
- S4: 2/8 — only source linking and lockfile support
- S5: 2/6 — checksums only; no cryptographic signing, no attestations
- S8: 3/7 — no transparency log, no build logs, no security metrics

S1: Human authentication — still uneven

Registry	MFA enforced	SSO/Federation	Publisher verified	Org link
npm	Yes	No	Yes	Yes
PyPI	Yes	No	Yes	Yes
RubyGems	No	No	No	No
crates.io	Yes (via GitHub)	No	No	No
Packagist	No	No	No	No
Hex.pm	No	No	No	No

S4: Build & Provenance — the persistent gap

- Reproducible builds: 0/6 registries
- Sandboxed build environments: 0/6 registries
- SLSA provenance metadata: npm and PyPI only
- Source repository linking: 6/6 registries
- SBOM availability: npm only

S5: Integrity — a brighter spot, with caveats

- Checksums near-universal: published and verified on install
- Cryptographic signing by publishers: RubyGems and PyPI
- Keyless signing via Sigstore: RubyGems, npm, and PyPI
- Attestation verification: RubyGems, npm, PyPI
- crates.io, Packagist: checksums only; Hex.pm adds registry-level verification

S8: Transparency — the audit trail problem

- Transparency log of publications: npm, PyPI, RubyGems
- Build log access: npm only
- Public audit logs of account events: Hex.pm only
- Malicious package removal feed: npm, RubyGems, crates.io
- Security researcher APIs: most registries

The pattern: visible vs invisible controls

Visible controls (enforced at interaction time)
- MFA: you either log in or you don't
- Checksums: verified on install, failure is immediate

Invisible controls (benefit is diffuse, failure is silent)
- Provenance: `pip install` succeeds whether or not it was attested
- Transparency logs: you don't know you needed them until an incident
- Audit trails: absence is only noticed during forensics

Why this matters for consumers

- You cannot control the registry's security policies
- You can control your own verification practices
- The invisible gaps require compensating controls on your side
- Build your pipeline to verify what the registry does not

Where to focus: a priority map

High priority (registry gaps + high consumer impact)
- Provenance verification where available (npm, PyPI via SLSA)
- Dependency pinning and lockfile enforcement across all ecosystems
- Monitor for unexpected version changes in your dependency graph

Medium priority (emerging capability, worth tracking)
- Sigstore-based signature verification (RubyGems, npm, PyPI)
- Private registry proxies with approval gates before packages reach builds

Baseline (already widely implemented by registries — lean on them)
- Checksum verification — the registries handle this
- MFA on your publisher accounts — enforce it yourself if the registry won't

Engaging the ecosystem

- Advocate to registries that haven't mandated MFA to do so
- Fund or contribute engineering time toward the gaps
- Reach out to critical maintainers in your chain — help them enable MFA
- Use OIDC trusted publishing where supported (npm, PyPI) — eliminate long-lived CI tokens

What the data tells us about trajectory

- Authentication: improving — npm, PyPI have moved; others lagging but moving
- Trusted publishing (OIDC-based tokens): gaining traction on npm, PyPI
- Provenance infrastructure: being built, slowly, in the right ecosystems
- Transparency & Auditability: most neglected domain; investment consistently deferred

Summary

- Systematic measurement reveals gaps that intuition misses
- npm leads at 85%, but still has provenance and transparency gaps
- crates.io — a reminder that security culture doesn't automatically translate into registry infrastructure
- S4 (Build & Provenance) and S8 (Transparency) show the widest cross-registry gaps — most registries fall furthest short here
- The visible-versus-invisible pattern explains where the gaps cluster
- Consumer compensating controls matter most where S4 and S8 are absent

Thank you

Questions?

- https://securitylabs.datadoghq.com/
  - Super great blog, really!
  - Sign up for our human-written monthly newsletter :)