CHECKLIST

Node.js Secure Dependency Management Checklist

A practical checklist for evaluating package trust, locking dependency versions, triaging audits, and verifying Node.js dependency changes before production release.

Node.js Secure Dependency Management Checklist

Use this checklist before introducing a new package or promoting a dependency update to production.

1) Package approval

- [ ] Confirm the package is needed and there is no safer built-in or existing dependency alternative.
- [ ] Identify whether the package is direct or transitive.
- [ ] Review the package maintainer, ownership, and release history.
- [ ] Check whether the package scope, name, and repository appear consistent and legitimate.
- [ ] Verify the package has recent, stable maintenance activity appropriate to your use case.
- [ ] Flag packages with suspiciously fast growth, sudden maintainership changes, or unclear provenance.

2) Trust and behavior review

- [ ] Inspect the package README, install instructions, and published metadata.
- [ ] Check for install-time scripts such as preinstall, install, or postinstall.
- [ ] Determine whether the package requires native compilation or external build tools.
- [ ] Review whether the package needs elevated permissions, network access, or shell execution during install.
- [ ] Compare the package name with popular libraries to detect lookalike or typosquatting risk.
- [ ] Confirm the package behavior matches the security sensitivity of the application.

3) Dependency graph review

- [ ] Review the direct dependency change and the transitive packages it introduces.
- [ ] Identify whether the update changes authentication, authorization, parsing, serialization, logging, or crypto behavior.
- [ ] Check whether the update adds new transitive packages that expand trust assumptions.
- [ ] Confirm that any high-risk dependency is explicitly reviewed rather than accepted implicitly.
- [ ] Document any package that influences secrets, tokens, uploads, or artifact generation.

4) Version control and reproducibility

- [ ] Pin the intended direct dependency version.
- [ ] Update the lockfile intentionally and review it as part of code review.
- [ ] Ensure the manifest and lockfile are consistent.
- [ ] Use the lockfile in CI, staging, and production builds.
- [ ] Confirm the install is reproducible from a clean environment.
- [ ] Avoid unreviewed lockfile drift caused by local installs or ad hoc updates.

5) Build and CI validation

- [ ] Run dependency installs in a clean CI environment.
- [ ] Verify the build uses declared sources only.
- [ ] Fail the pipeline if the lockfile and manifest are out of sync.
- [ ] Confirm tests pass against the exact dependency tree that will be released.
- [ ] Check that no unexpected install scripts ran during CI.
- [ ] Record the build evidence needed to support a release decision.

6) Audit and vulnerability triage

- [ ] Run a dependency audit or equivalent vulnerability check.
- [ ] Classify findings by severity and by application context.
- [ ] Distinguish between exploitable issues and low-risk alerts.
- [ ] Track each finding as accepted, deferred, mitigated, or fixed.
- [ ] Set a due date and owner for unresolved high-risk findings.
- [ ] Re-check the dependency tree after remediation to confirm the fix landed.

7) Release gating

- [ ] Require explicit approval for packages affecting authentication, authorization, cryptography, input validation, or artifact generation.
- [ ] Confirm the release candidate matches the tested dependency tree.
- [ ] Verify no unapproved dependency changes are present in the final artifact.
- [ ] Ensure the release notes include meaningful dependency updates and known risks.
- [ ] Promote the build only after evidence is recorded and reviewed.

8) Operational follow-up

- [ ] Schedule a regular dependency review cadence.

- [ ] Keep a policy for when exact pinning is required versus when controlled updates are acceptable.
- [ ] Review dependency alerts for patterns, not just one-off issues.
- [ ] Remove unused packages and reduce dependency sprawl where possible.
- [ ] Reassess trust in packages with abandoned maintenance or unusual behavior.
- [ ] Treat packages in the security trust boundary as production-critical assets.

Quick release decision test

Before approval, answer these four questions:

- [ ] Can we explain why this package is allowed?
- [ ] Can we prove exactly what version will run in production?
- [ ] Can we reproduce the install in a clean CI environment?
- [ ] Can we show the evidence that supports the release decision?

If any answer is ?no,? do not promote the release yet.

Notes

- Treat lockfiles as release artifacts, not housekeeping files.
- Secure dependency management reduces supply-chain risk, but it does not replace code review or runtime controls.
- For packages that affect trust boundaries, require a higher level of scrutiny and a documented approval trail.