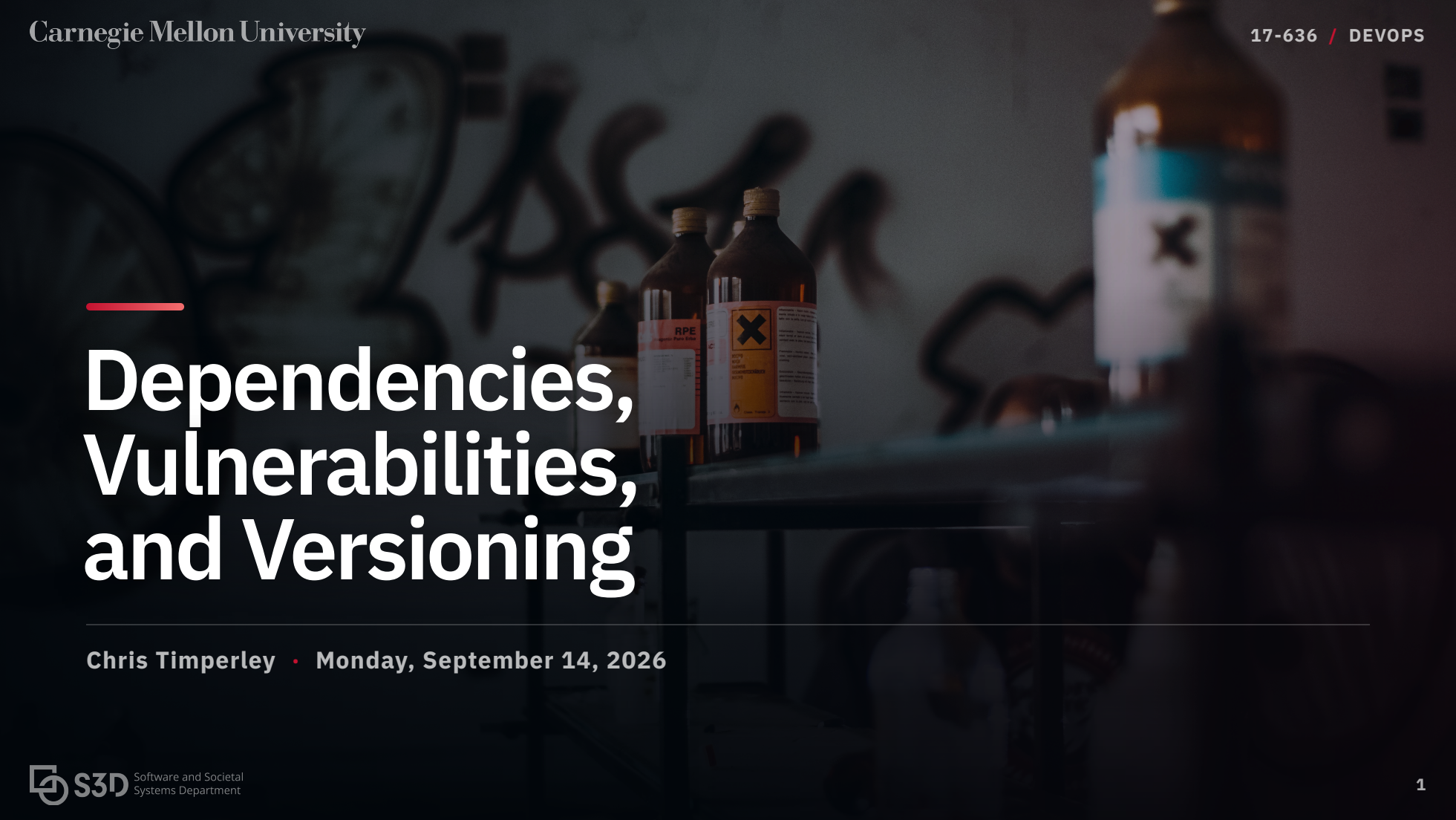


Dependencies, Vulnerabilities, and Versioning

Chris Timperley · Monday, September 14, 2026

Administrivia

- **Homework 2**
  - Checkpoint A was due last night (September 13th)
  - Checkpoint B is due September 20th
  - Goal: build a fast CI pipeline that is able to catch developer mistakes
  - Bonus points for speed and ability to spot particularly tricky mistakes
- **Lab: Syft and Grype**
  - producing SBOMs and scanning for vulnerabilities

🎯 Learning Goals

- Trace installed versions through manifests, dependency paths, and lock files.
- Explain compatibility risk using the public API, version labels, and tests.
- Justify a vulnerability repair and identify the checks still needed.
- Compare update policies and conditions for revisiting exceptions.
- Justify adopting, deferring, or replacing a dependency for an application.

Projects depend on packages distributed through registries



Dependency

Software our project relies on

Examples: React, NumPy, Lodash.



Package

A named, versioned unit of reusable software

Package names: react, numpy, lodash.



Registry

A service that distributes packages

Examples: npm, PyPI, Maven Central.

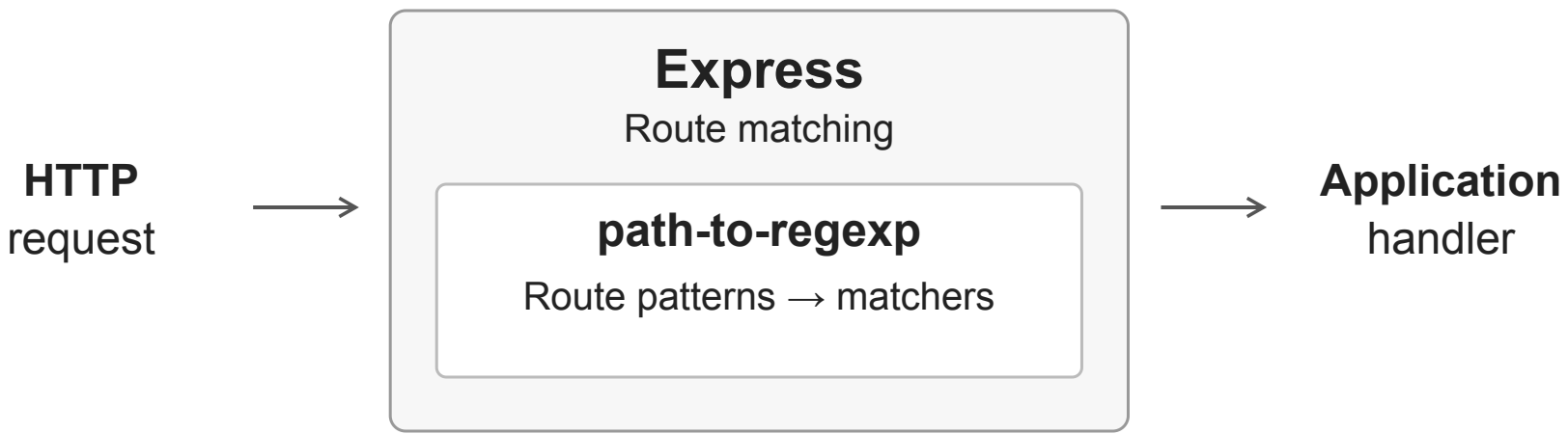
Dependencies create ongoing responsibility

- Dependencies are essential building blocks for our software
- Reusing them saves us implementation work
- Each dependency also adds **ongoing maintenance work:** handling behavior changes, vulnerabilities, and the end of support
  - **Update the code?** Existing behavior may break
  - **Keep the existing code?** A known defect may remain

Running Example: Express

We'll follow a historical repair to weigh a **vulnerability fix** against the risk of **breaking existing behavior**.

Express is a JavaScript framework for handling HTTP requests in Node.js services.



Running Example: Express (v4.21.2)

What's Changed

- Add funding field (v4) by **@bjohansebas** in #6065
- deps: path-to-regexp@0.1.11 by **@blakeembrey** in #5956
- deps: bump path-to-regexp@0.1.12 by **@jonchurch** in #6209
- Release: 4.21.2 by **@UlisesGascon** in #6094

Full Changelog: `4.21.1...4.21.2`

GitHub release notes · Express 4.21.2

- **Claim:** fixes a vulnerability in the path-to-regexp matcher
- **Local evidence:** existing routing tests pass before and after the update

What evidence would justify accepting this update?

Ask four questions before accepting an update



Dependency resolution
Which package versions will the project actually use?



Compatibility
Will the behavior we rely on still work?



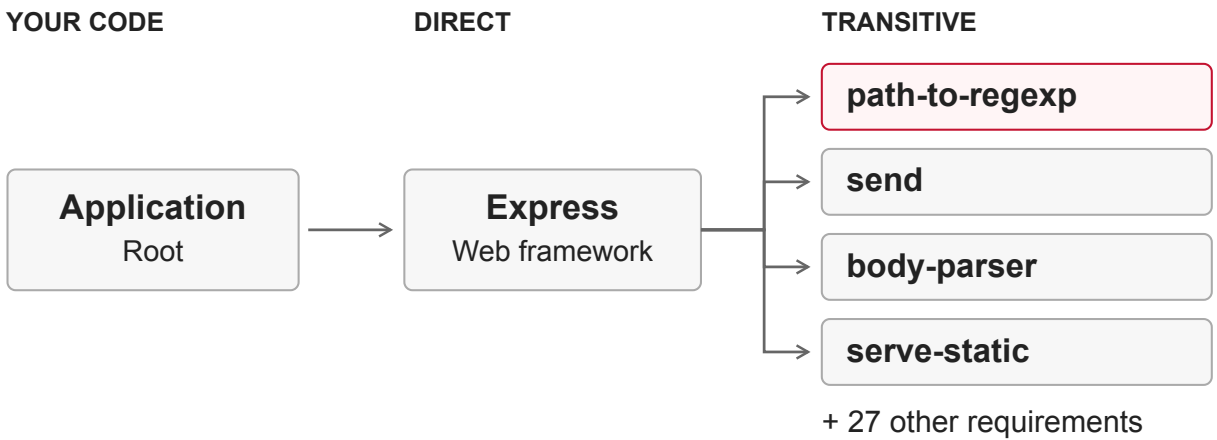
Vulnerability diagnosis
Does this update address the vulnerability affecting our project?



Supply-chain trust
Where does the code come from, and what access will it have?

Dependencies

Your dependencies have dependencies



- **Direct dependency:** a package your project requires (e.g., Express)
- **Transitive dependency:** a package brought in by another dependency (e.g., path-to-regexp through Express)

A manifest says what your project needs

A **manifest** describes a project and its requirements. npm uses `package.json`.

PACKAGE.JSON

```
{
  "name": "web-app",
  "version": "1.0.0",
  "dependencies": {
    "express": "^4.21.1"
  }
}
```



Parent: web-app
Any package that declares a dependency is that dependency's parent.



Allowed: ^4.21.1
The manifest permits a set of Express versions.



What’s wrong with this requirements.txt?

```
requests
flask
```

No versions → fresh installs can change ("it works on my machine")
New APIs, defaults, or transitive dependencies.

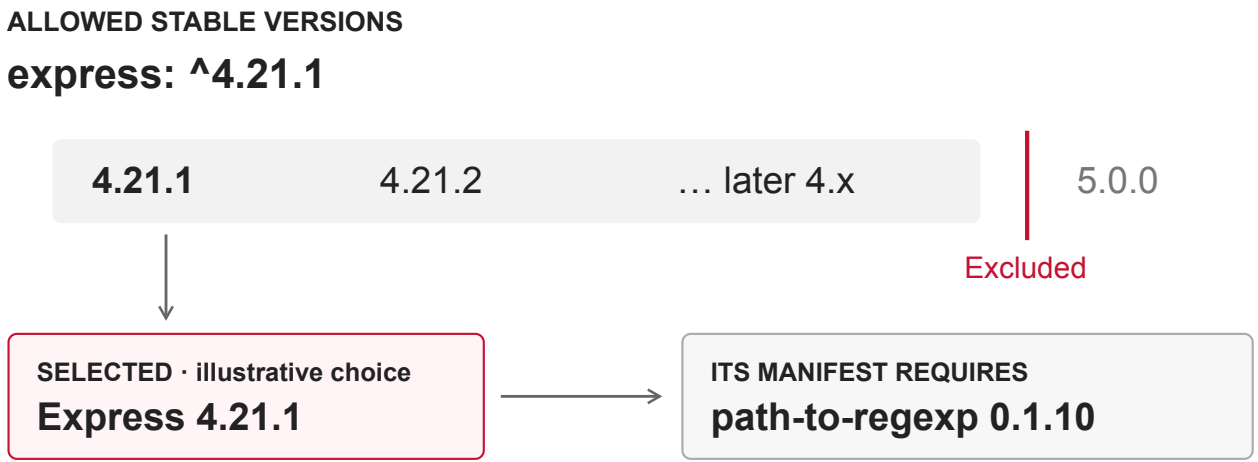
Better: Specify the versions you allow

```
requests>=2.32,<3
flask==3.1.0
```

- **Range:** 2.32 up to, but not including, 3
- **Pin:** exactly 3.1.0
- Beware direct pins still leave **transitive versions** open

Which allowed version actually gets installed?

Dependency resolution chooses concrete versions that satisfy the requirements.



A **range** says what is allowed; a **resolved version** says what was chosen. Repeat for each package's dependencies; existing lock selections can guide the choice.

⚠ Dependency requirements can conflict

One Python environment

FastAPI 0.99.1 → **Pydantic 1.x**
`>=1.7.4, <2.0.0`
`excluding 1.8 and 1.8.1`

pydantic-settings 2.0.3 → **Pydantic ≥2.0.1**
`>=2.0.1`

No shared version. Resolution fails!

⚠ One update can force another

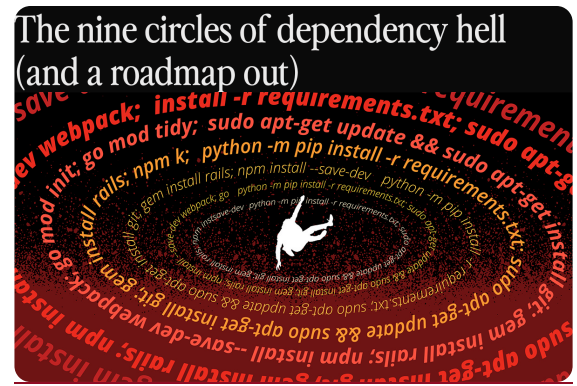
Use pydantic-settings 2.0.3 → Requires **Pydantic 2.x** → **Upgrade FastAPI to 0.100.0**

Or keep **Pydantic 1.x** and its built-in settings support.

Finding compatible versions may require changes across several packages.

The nine circles of dependency hell

1. **Limbo:** installed packages ≠ recorded packages
2. **Lust:** adding packages without vetting them
3. **Gluttony:** too many dependencies
4. **Greed:** competing package managers
5. **Wrath:** needed package or version unavailable
6. **Heresy:** monkey-patching dependencies
7. **Violence:** breaking minor or patch releases
8. **Fraud:** circular dependencies
9. **Treachery:** conflicting shared-library requirements



Matt Rickard · Sourcegraph · 2021

Read the article ↗

npm can install multiple versions

Inside the Express repository



Separate copies satisfy both parents, but add code and versions to maintain.

Node finds the nearest package copy

Inside the Express repository

INSTALLED PACKAGE FILES

```
express/  (repository root)
└── node_modules/
    ├── debug/  (2.6.9)
    ├── send/
    └── cookie-session/
        └── node_modules/
            └── debug/  (3.2.7)
```

Express + send
Search upward → debug 2.6.9

cookie-session
Nested copy first → debug 3.2.7

Python typically uses **one library version per environment**.

Group dependencies by when you need them

```
"dependencies": {
  "path-to-regexp": "0.1.10"
},
"devDependencies": {
  "mocha": "10.2.0"
}
```

Other entries omitted.

NPM'S TWO MAIN GROUPS



Application dependencies

The route matcher handles application routes.



Development tools

Mocha runs CI tests. Other dev tools run builds.

`npm ci --omit=dev` omits development dependencies from a production install.

Other ecosystems support finer groups, such as **tests**, **documentation**, or **optional features**.

The lock records direct and transitive versions

Inside the Express repository

Share the lock to repeat the resolver's selections.

Express package.json → `"send": "0.19.0"`

CAPTURED PACKAGE-LOCK.JSON

```
"node_modules/send": {
  "version": "0.19.0",
  "resolved": "…/send-0.19.0.tgz",
  "integrity": "sha512-dW41u5Vf…"
}
```

Selected fields; URL and hash abbreviated.



version

The selected version.



resolved

Registry tarball to fetch.



integrity

Hash of the expected bytes.

Commit the manifest and lock together







1. Resolve

```
$ npm install
```

Update selections to satisfy changed requirements.

2. Review and commit

`package.json`

`package-lock.json`

Commit requirements and reviewed selections together.

3. Install in CI

```
$ npm ci
```

Install from the lock; fail if it disagrees with the manifest.

A changed requirement needs a new lock

Inside the Express repository: change **path-to-regexp** from **0.1.10 → 0.1.12**.

REQUIREMENT EDITED

Manifest and lock disagree

Manifest requires 0.1.12

Lock selects 0.1.10

npm ci fails: 0.1.10 is not allowed

npm install

LOCK REGENERATED

Both select the repair

Manifest requires 0.1.12

Lock selects 0.1.12

npm ci succeeds: installs 0.1.12

Review and commit the requirement and lock changes together.

Why don’t we just pin every dependency?



In the lock ≠ in the shipped image

A **software bill of materials (SBOM)** inventories components for a stated scope.



Selected

`package-lock.json`

Mocha present

Remains in the saved lock.



Installed

`npm ls`

Mocha absent

After a production install:

```
npm ci --omit=dev
```



Shipped

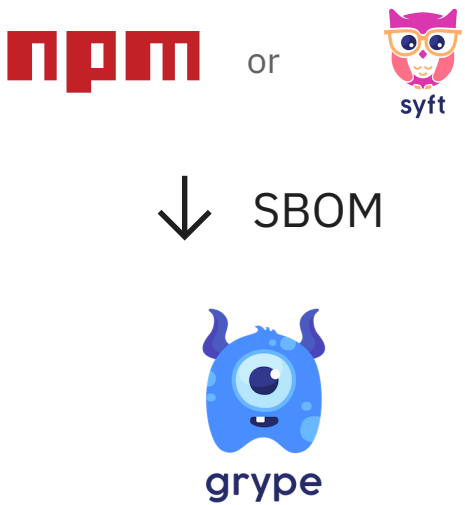
Image inventory / SBOM

Inspect the image

The lock alone cannot establish what it contains.

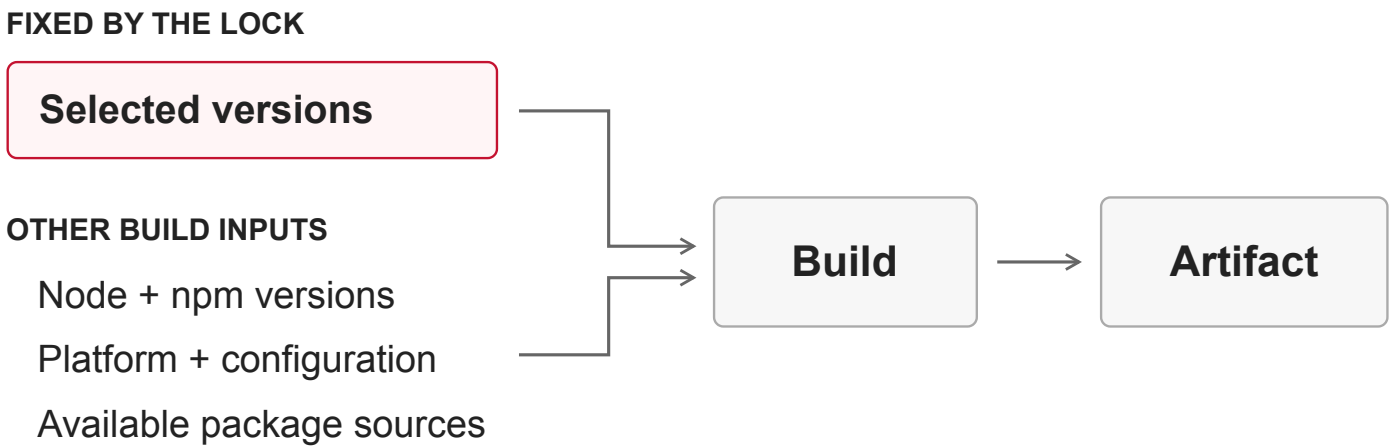
Inventory packages, then check for known vulnerabilities

- **Record packages and versions**
  - npm: project dependencies
  - Syft: OS + application packages
- **Match against advisories:** Grype
- **New advisory?** Scan again.
  - **New build?** Regenerate the inventory.



⚠ A lock fixes only some build inputs

Dependency reproducibility means obtaining the same dependency selections again.



Changing Node or enabling install scripts changes the build inputs—even with the same lock.

Versioning

The public API is a promise to callers

The **public API** is the interface and behavior a package promises to callers.

```
const name = formatName('Ada', 'Lovelace');
// 'Ada Lovelace'
const heading = name.toUpperCase();
// 'ADA LOVELACE'
```

This illustrative package accepts names and returns a **string**.

A package update can break your code



Backward-compatible
Existing application code keeps working unchanged.



Breaking change
Callers must adapt to the changed API.

Patch releases fix bugs

Semantic versioning (SemVer) labels changes as `MAJOR.MINOR.PATCH`.

Before · v2.1.3

```
formatName('Ada', '');
// 'Ada '
```

Unwanted trailing space.

After · v2.1.4

```
formatName('Ada', '');
// 'Ada'
```

Space removed as promised.

- The API promises no surrounding spaces
- This fix preserves supported calls and the string result

Minor releases add features

Before · v2.1.3

```
formatName(
  'Ada', 'Lovelace'
);
// 'Ada Lovelace'
```

After · v2.2.0

```
formatName(
  'Ada', 'Lovelace',
  { order: 'last-first' }
);
// 'Lovelace, Ada'
```

- Existing calls keep their behavior
- The new setting is optional

Major releases can require code changes

Before · v2.1.3

```
const name = formatName(
  'Ada', 'Lovelace'
);
// 'Ada Lovelace'
name.toUpperCase();
```

After · v3.0.0

```
const name = formatName(
  'Ada', 'Lovelace'
);
// { text: 'Ada Lovelace' }
name.text.toUpperCase();
```

- Version labels express the publisher’s (possibly flawed) assessment
- **Our tests still need to check our use**

Can fixing a bug be a breaking change?



Version ranges limit allowed updates

npm requirement	Patch `2.1.4`	Minor `2.2.0`	Major `3.0.0`
`2.1.3`	Excluded	Excluded	Excluded
`~2.1.3`	Allowed	Excluded	Excluded
`^2.1.3`	Allowed	Allowed	Excluded

- All allow `2.1.3`
- `~` allows later `2.1.x`
- `^` allows later `2.x`.

⚠ New releases do not update your lock file

Manifest requirement: `^2.1.3` · Locked version: **2.1.3**



Release published
2.2.0 satisfies the range.

CI runs
npm ci still installs locked 2.1.3.

Team updates
Select 2.2.0, test, and commit the lock.

The range sets permission; the lock records a selection.

Version labels guide us; tests check our application

Version labels can be wrong

- Patch and minor releases can accidentally break working code.
- A version number cannot guarantee compatibility with your application.

Test the behavior you depend on

- Run tests that exercise how your application uses the dependency.
- Check expected results—not just whether the application starts.

For our service

Does `GET /notes` still return `200` and `preview` after the update?

Example: An Express upgrade breaks this route

```
app.get('/*', (req, res) => res.send('preview'));
```

The wildcard `*` handles paths such as `/notes` in Express 4.

Express 4.21.2

Route registers

GET /notes → 200, preview

Express 5.1.0

Registration fails

“Missing parameter name”

The accepted route syntax changed; the method is still `app.get`.

Example: Test the behavior after repairing the route

Express 5 requires a **name after** `*`. Then test the same response:

```
const app = express5();
// path is the wildcard's name
app.get('/*path', (req, res) => res.send('preview'));
await request(app).get('/notes')
  .expect(200, 'preview');
```

Pass: /notes returns 200 and preview

Other required routes need their own tests.

Vulnerabilities

We upgrade dependencies for several reasons



Bug fixes
Correct behavior that does not work as intended.



New capabilities
Use features and performance improvements.



Compatibility and support
Keep working with supported runtimes and libraries.



Security fixes
Remove known weaknesses in dependencies.

Advisories describe known vulnerabilities

A **vulnerability advisory** describes a known flaw, its impact, and ways to respond.



path-to-regexp contains a ReDoS

High severity | GitHub Reviewed | Published on Dec 5, 2024 in **pillarjs/path-to-regexp** • Updated on Jun 3, 2025

Vulnerability details | Dependabot alerts 0

Package	Affected versions	Patched versions
path-to-regexp (npm)	< 0.1.12	0.1.12

Look for **affected versions**, **impact and conditions**, and **a fix or workaround**.

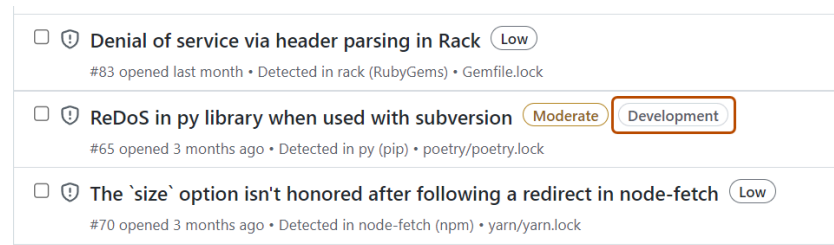
One finding, two identifiers: **CVE-2024-52798** · GHSA-rhx6-c78j-4q9w ↗

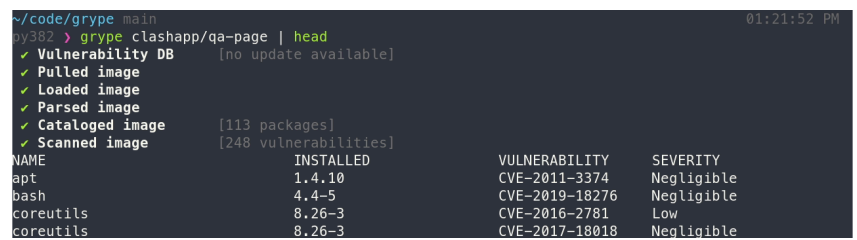
Scanners compare versions with advisories

npm audit
Checks an npm dependency tree.

Dependabot alerts
Matches GitHub's graph to reviewed advisories.

Grype
Scans images, directories, or an SBOM.





Inventory and advisory coverage limit what a scan can find.

An affected dependency may not be exploitable in our application



Vulnerability · the weakness
A flaw that can cause harm.

Example: certain inputs make a request matcher excessively slow.



Exploit · a way to trigger it
Technique that uses the flaw.

Example: an attacker sends a crafted request that ties up the service.

A scanner finds affected versions. Diagnosis checks whether an attacker can trigger the flaw here.

Diagnosis helps set repair priority



Affected version
Is our installed version in the advisory's affected range?



Reachability
Can our application run the vulnerable operation?



Exposure
Can an attacker supply the input or access needed to trigger it?

Investigate exposure to set urgency; keep the repair scheduled.

Each installed copy needs its own assessment

Inside the Express repository · 2026-09-09

```
$ npm explain path-to-regexp
path-to-regexp@0.1.10
node_modules/path-to-regexp
  path-to-regexp@"0.1.10"
  from the root project
```

Version: 0.1.10

Inside the advisory's affected range: <0.1.12.

Parent: Express

Root project here; requires this copy and calls its matcher.

- Several versions can coexist. **One fixed copy does not repair the others.**
- Identify the **package name, version, and dependency path** of each copy.

Vulnerable uses have specific conditions

- Exploitation may require particular **inputs or configuration**.
  - the system may not meet those conditions today — it could tomorrow
  - the system might reach vulnerable code indirectly via a framework (Express)
- An advisory's conditions tell us which uses to investigate.

Express example: a route has two parameters between the same pair of slashes, separated by something other than a period:

```
/users/:from-:to
```

Crafted requests to a matching route can cause excessive matching work, leading to a denial of service.

Exposure depends on the running system

Exposure depends on who can supply the triggering input and which controls stand between that input and the vulnerable operation.

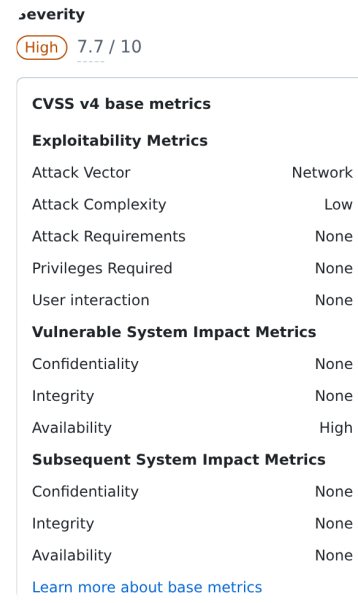
- A public route may let anyone send input to the matcher.
- A control protects an operation only if it runs **before that operation**.
- A local checkout may contain different code from the deployed artifact.

Do we need to fix this right now?



Severity alone does not set patch priority

- **Severity:** the flaw's **potential** harm and exploitation conditions
- **Risk:** the likelihood and consequences of **actual** exploitation in our system.
  - Example: a public, critical service makes an outage more consequential



Use severity, exposure, and likely harm to set repair priority.

Delaying a patch gives attackers more time

- Attackers can automate searches for systems with known flaws.
- **Log4Shell, 2021:** the FBI observed widespread scanning and attempted exploitation.
- AI is likely to be used to accelerate exploitation
- **Prompt patching reduces the time our system remains vulnerable.**

> By 2027, AI-enabled tools will almost certainly enhance threat actors' capability to exploit known vulnerabilities, increasing the volume of attacks against systems that have not been updated with security fixes. System owners already face a race in identifying and mitigating disclosed vulnerabilities before threat actors can exploit them. The time between disclosure and exploitation has shrunk to days and AI will almost certainly reduce this further. This will highly likely contribute to an increased threat to CNI or CNI supply chains, particularly any operational technology with lower levels of security.

NCSC · May 2025 · Forecast through 2027

Mitigation leaves the flaw in place

- A **mitigation** limits exploitation or its impact while the vulnerable code remains
  - this buys time for more extensive and delicate repairs
  - verify that the mitigation is effective — beware that this may have side effects
  - **Example (Express):** restrict route parameters or cap URL length.
- A **patch** changes the code to fundamentally address the flaw
  - regardless of current exposure, you should always **schedule the patch:** new routes, configuration, or access can defeat today's mitigation.

Transitive fixes may require a parent update



If no suitable parent update exists:

- **Override** the transitive version; test compatibility.
- **Replace** the parent dependency; adapt the application.
- **Mitigate** temporarily; keep the repair scheduled.

Regression tests reduce the cost of patching

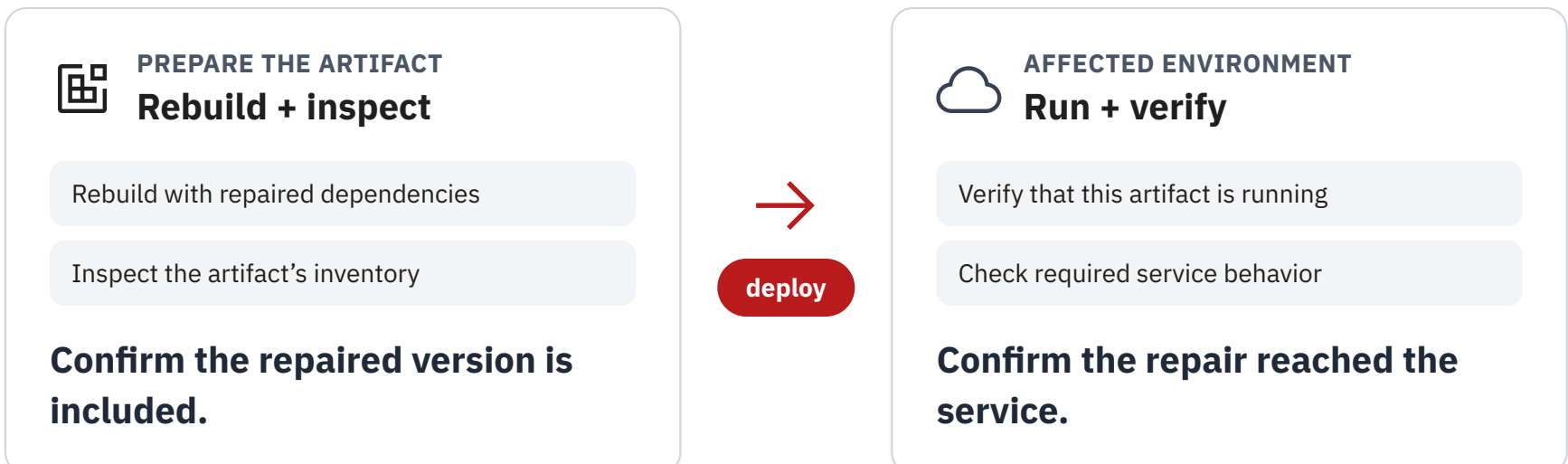
- **Behavior tests** check that the app still works
  - adapt code if the upgrade requires it
- **Security tests** reproduce the flaw
  - this is ideal, but not always practical
  - fail before the fix; pass after it.
- Maintain behavior tests so **urgent updates are easier to verify**.

Routing tests · saved output excerpt

```
✔ should denote an optional capture
group
✔ should denote a format
✔ should continue lookup
✔ should jump to next route
140 passing (63ms)
```

A patch protects only systems that run it

- Changing the repository leaves existing deployments unchanged.
- The repair must reach the exposed system.

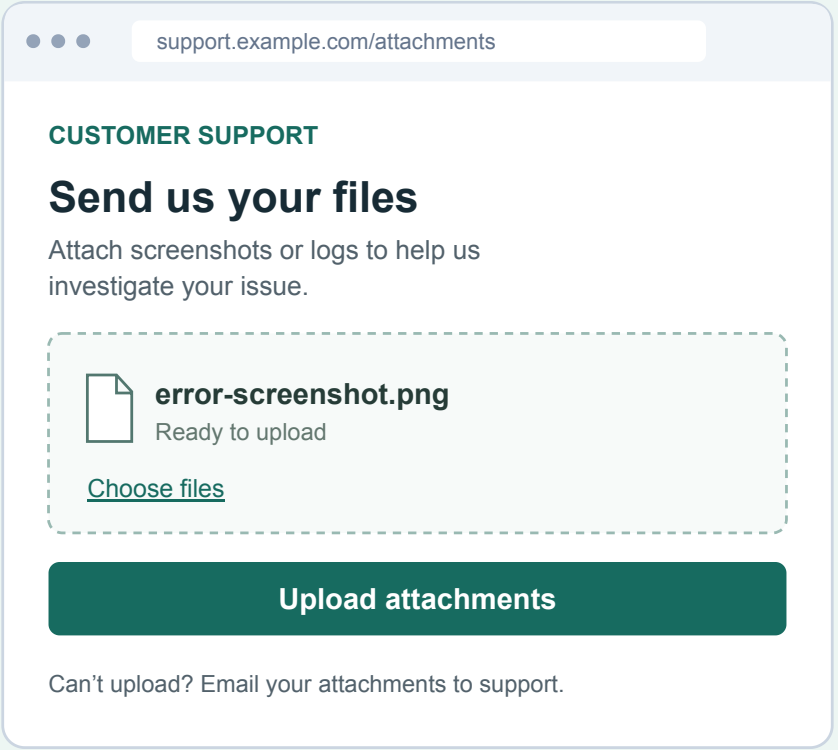


Our upload service has an advisory

- Support needs **screenshots and logs**
- Uploads are **publicly accessible**
- **Email is the temporary fallback**

The service runs the supplied code and **locked dependencies**.

What should our team do today?



Investigate, then recommend an action

A teammate says:

> “We already limit each request to 12 files and handle upload errors, so we can leave uploads running until next week.”

Read the GitHub advisory ↗ Code repository ↗

- **Evaluate the claim:** do the file limit and error handling address this advisory?
- **Decide what to do today:** explain the tradeoffs and changes
- **Plan verification:** check behavior, the security fix, and deployment

Would you deploy this repair?

- **Claim:** would you accept your teammate's reasoning? Why?
- **Action:** keep uploads running, or use the fallback while patching?
- **Verification:** "The upload form still works." Is that enough?

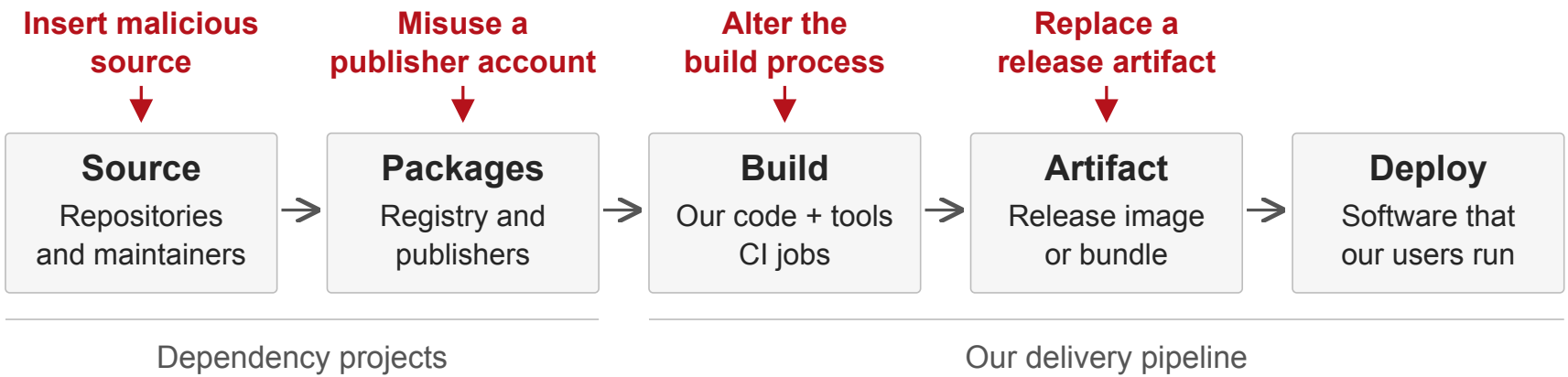
Supply-Chain Trust

Our software comes through a supply chain



- **Software supply chain:** the people and systems that produce and deliver software
- Each dependency brings its own supply chain

Attackers can target any stage of the chain



- A **supply-chain attack** compromises this chain to introduce malicious behavior

A trusted dependency can become malicious

event-stream incident · November 2018



New dependency → targeted development environment → changed application

- A new maintainer introduced a malicious dependency.
- It targeted Copay builds to steal wallet information and private keys.

Original diagram based on npm's incident account.

Why not trust packages we've used for years?



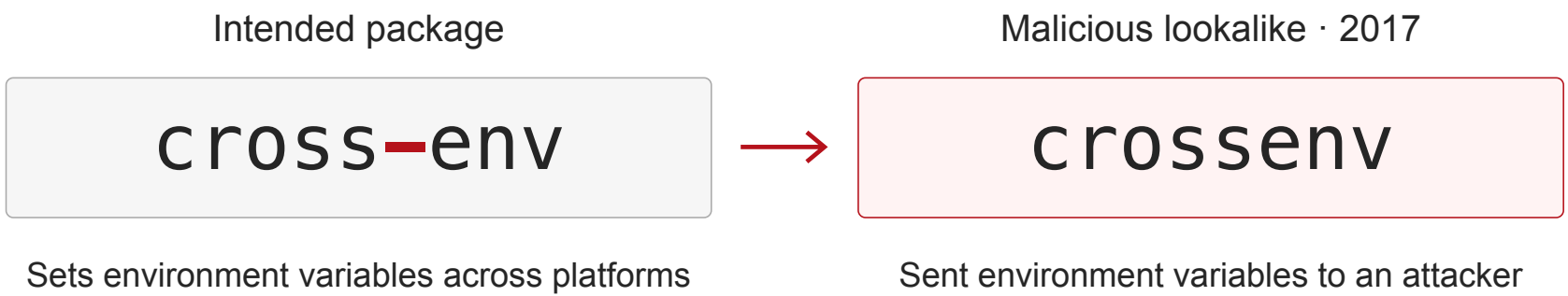
Passing checks does not establish trust

Check	What it tells us
Integrity hash	Downloaded bytes match the expected bytes.
Functional tests	Tested behavior works in the tested environment.
Vulnerability scan	Whether covered packages match known reports.

- A locked malicious version can pass an integrity check
- Targeted malicious behavior can escape tests and scans

Similar names can hide a different package

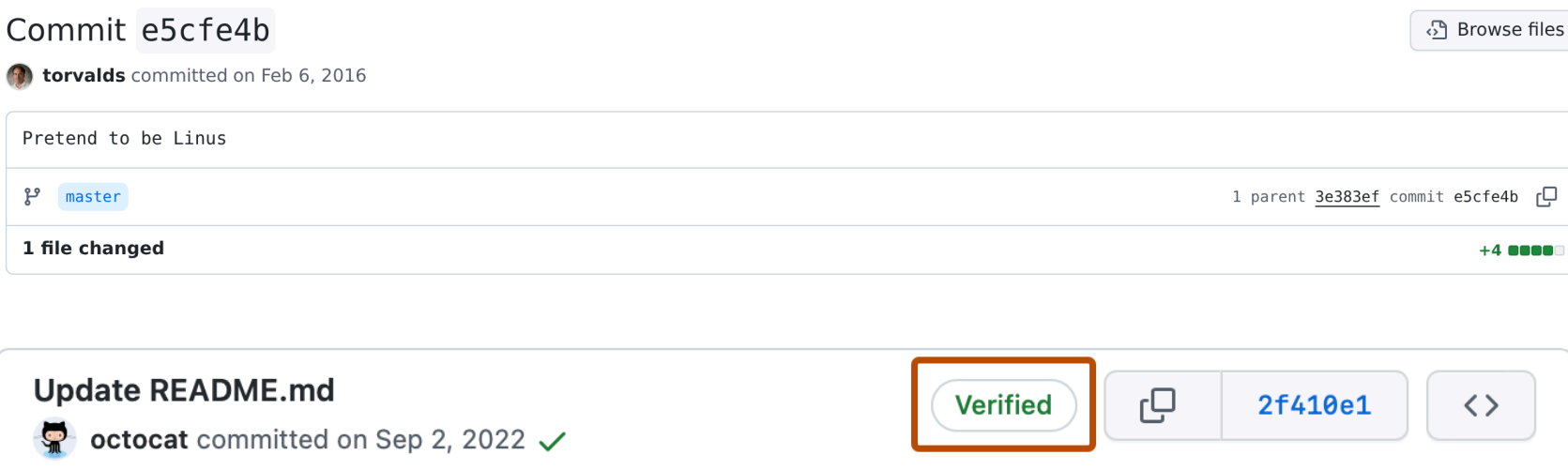
- **Check the exact name, repository, and publisher in the registry.**
- **Typosquatting** uses similar names to attract mistaken installs.



npm's 2017 incident report · Original illustration of the package-name difference.

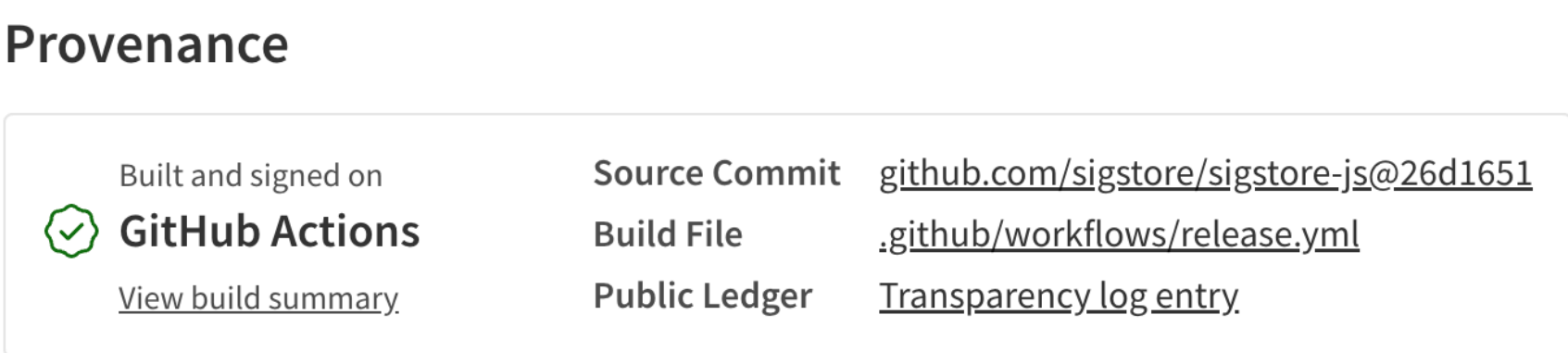
An author name is easy to fake

- **Git author names and emails are self-reported.**
- Inspect the signature and expected signer; still review the code.



Connect the package to its source and build

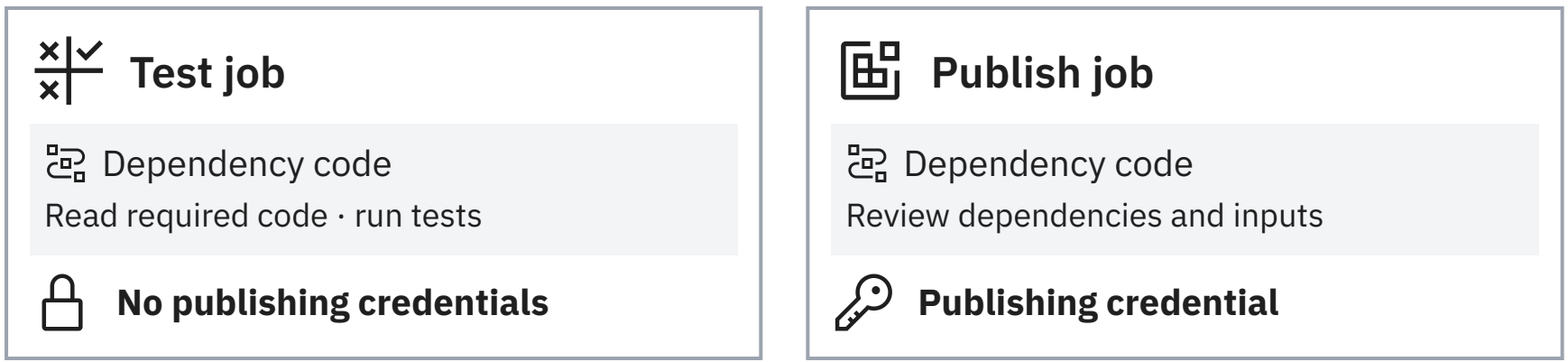
- A **provenance attestation** is a signed record of a package's source and build.
  - e.g., verify with `npm audit signatures`; inspect the source and workflow.



npm documentation example

Dependency code inherits the CI job's access

Dependency code can run during **installation, builds, and tests**.



Test dependencies need no publishing access.

Code running here can use this access.

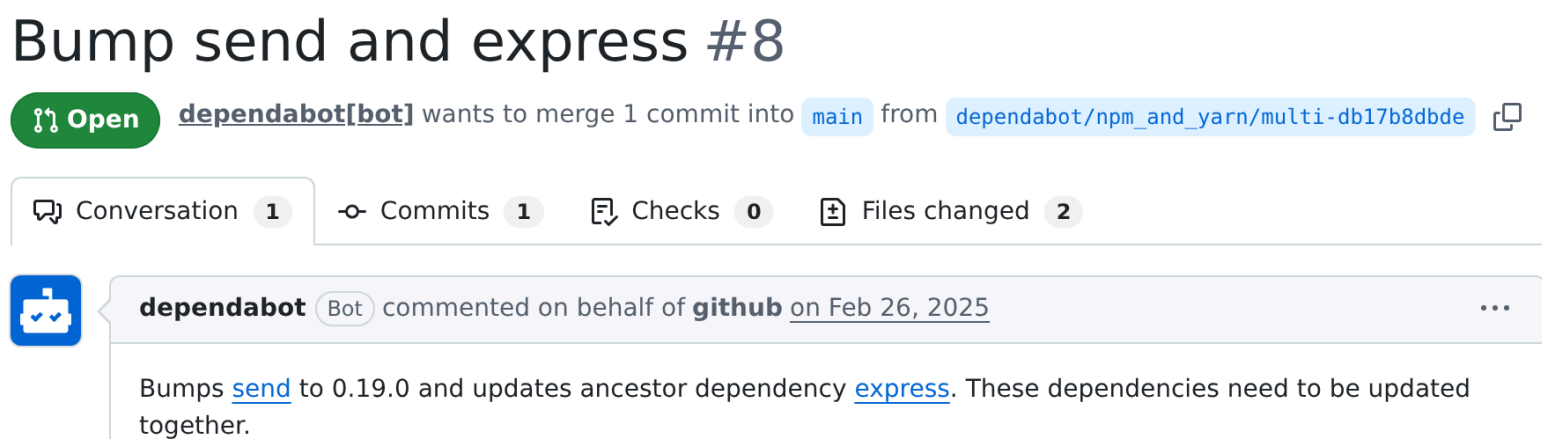
Recall: Apply the principle of Least Privilege: grant only the access each job needs.

Dependency Maintenance

How should we handle updates over time?

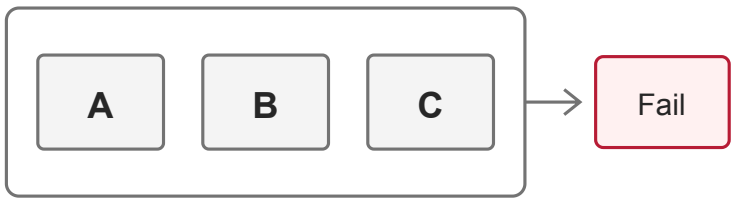
Bots propose updates; teams decide

- **Dependabot** and **Renovate** open update pull requests.
- Require tests and review; automate merging only for defined cases.



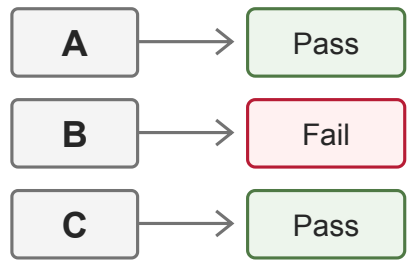
Small updates make failures easier to trace

One grouped proposal



Which change explains the failure?

Three small proposals



Investigate B

- Prefer smaller updates that are easier to apply and debug
- Group related changes that need testing together
- Review regularly; avoid a growing backlog of updates - integration becomes harder

Urgent fixes need a faster path

- Investigate relevant security alerts promptly.
- If postponing a repair, record **who, why, and when to revisit**.



Owner
Who will follow up?



Reason
Why can this wait?



Review date
When will we reassess?

Same code, newly discovered vulnerability

We updated to **path-to-regexp 0.1.12** to fix the earlier vulnerability.

path-to-regexp / Security / Advisories / GHSA-37ch-88jc-xwx2

path-to-regexp vulnerable to Regular Expression Denial of Service via multiple route parameters

High **UlisesGascon** published **GHSA-37ch-88jc-xwx2** on Mar 26

Package	Affected versions	Patched versions	Severity
path-to-regexp (npm)	<= 0.1.12	0.1.13	High 7.5 / 10

March 2026 advisory · Another vulnerability affects that same installed version.

Keep monitoring—even when dependencies have not changed.

Bonus: Choosing Dependencies

What evidence supports adopting a dependency for this use?

Do we need another dependency?

Do we need this capability, and do we already have it?



WORK A PACKAGE SAVES

Implementing the capability ourselves.



WORK WE TAKE ON

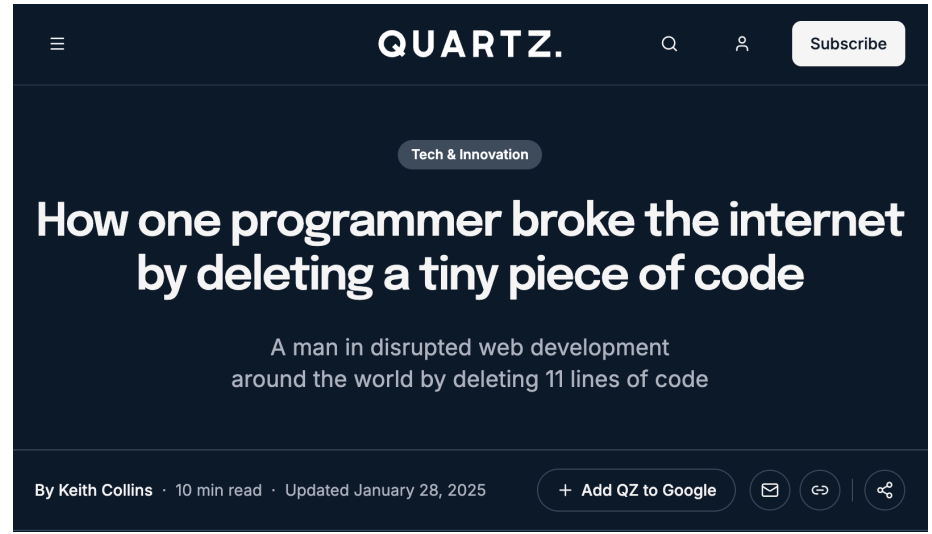
Integration and tests.
Updates and security fixes.

Adds characters to the start of a string:

```
leftPad('7', 3, '0')
// '007'
```

March 2016: package removal broke installs and builds.

≈2% of npm packages depended on it, directly or indirectly.



Quartz's account · npm's incident report

Was the code saved worth another dependency?

Inspect the project behind the package



README and docs
Required behavior and configuration?
Runtime support? API stability?



Releases and issues
Maintained? Problems addressed?



Tests and security policy
Automated checks? Reporting flaws?



Package metadata and license
Extra dependencies? Permitted use?

Reference: OpenSSF evaluation guide

Compare two Markdown parsers

Tiny Notes needs a **Markdown preview**: convert text such as `# Heading` to HTML.

- Render headings, lists, links, and fenced code.
- Show user HTML as text or remove it; reject unsafe links.
- Keep upkeep manageable for a small team.



Marked

markedjs/marked ↗



markdown-it

markdown-it/markdown-it ↗

Which would you choose—and why?

- **Recommend a candidate** for Tiny Notes
- **Cite two findings** from the repositories or docs
- **Name one uncertainty** and how you would test it

Choosing a package is just the start

- **Make it meet our requirements**
  - Configure HTML handling; add sanitization if needed
- **Test it in Tiny Notes**
  - Check the preview, unsafe input, and large notes
- **Assign someone to maintain it**
  - Track updates and security fixes; revisit the choice as needs change

Dependency Decisions

What can we conclude about the opening update now?

The evidence supports this historical repair

ESTABLISHED LOCALLY



Intended matcher installed

Express 4.21.2 → 0.1.12.



Selected advisory addressed

Fixes the selected 2024 finding.



140 routing tests pass

All selected checks still pass.

STILL TO VERIFY

- **Deployment:** verify the repaired artifact in the affected environment.
- **Maintenance:** investigate remaining findings.

Use the same reasoning beyond this example

1

IDENTIFY THE CODE

Requirements, selected versions, actual inventory.

2

EVALUATE THE CHANGE

Compatibility, vulnerability conditions, source trust.

3

VERIFY THE RESULT

Repaired version, required behavior, deployment.

4

MAINTAIN THE DECISION

Owner, update policy, reasons to reconsider.

These steps apply when **updating, adopting, deferring, or replacing** a dependency.

🚀 Today, we learned how to ...

- Trace installed versions through manifests, dependency paths, and lock files.
- Explain compatibility risk using the public API, version labels, and tests.
- Justify a vulnerability repair and identify the checks still needed.
- Compare update policies and conditions for revisiting exceptions.
- Justify adopting, deferring, or replacing a dependency for an application.