Software Ecosystem Dependency Management : Current and Future Challenges

Tom Mens
tom.mens@umons.ac.be

Software Engineering Lab
Faculty of Sciences

UMONS
Université de Mons

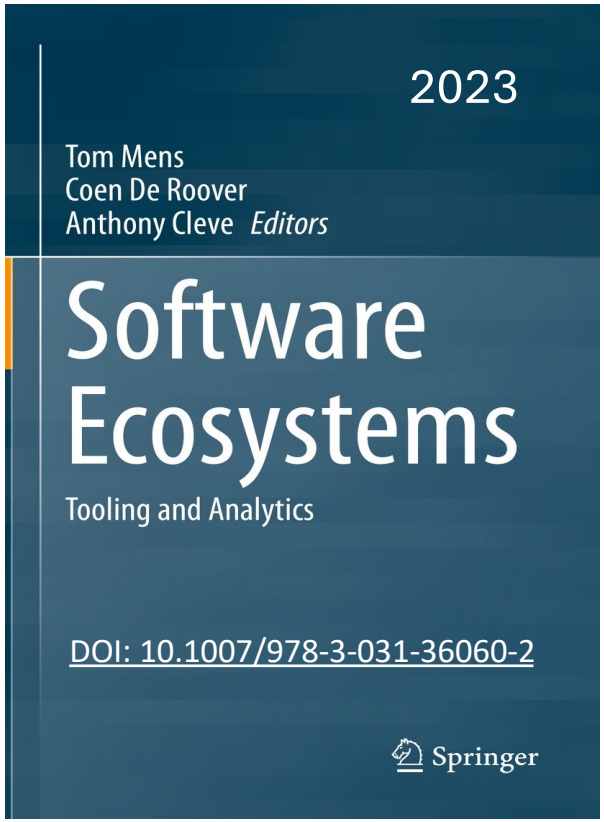
Rethinking Software Ecosystems, 12-16 January 2026, Lorentz center, Leiden, The Netherlands

Disclaimer

The scientific references used in this slide show is inevitably incomplete, given that the research domain I am focusing on is very large. The cited references should therefore be considered merely as illustrations. I truly apologise in advance if your research may not have been cited, mainly because of space considerations.

My recent software ecosystems research

- collaborative development
- GitHub workflow automation
- package dependency management



TITLE	CITED BY	YEAR
A Dataset of Contributor Activities in the NumFocus Open-Source Community Y Hourri, A Decan, T Mens 2025 IEEE/ACM 22nd International Conference on Mining Software Repositories …	2	2025
Observing bots in the wild: A quantitative analysis of a large open source ecosystem N Chidambaram, T Mens 2025 IEEE/ACM International Workshop on Bots in Software Engineering (BotSE …		2025
A bot identification model and tool based on GitHub activity sequences N Chidambaram, A Decan, T Mens Journal of Systems and Software 221, 112287	5	2025
On the evolution of direct dependencies in npm packages S Ebrahimi-Kia, JM Gonzalez-Barahona, D Moreno-Lumbreras, G Robles, ...		2025
An Empirical Analysis of the GitHub Actions Language Usage and Evolution AT Bardsiri, A Decan, T Mens		2025
An Analysis of Code Clones in GitHub Actions Workflows G Cardoen, A Decan, T Mens		2025
An Overview and Catalogue of Dependency Challenges in Open Source Software Package Registries T Mens, A Decan arXiv preprint arXiv:2409.18884	7	2024
Mitigating security issues in github actions HO Delicheh, T Mens Proceedings of the 2024 ACM/IEEE 4th International Workshop on Engineering …	10	2024
RABBIT: A tool for identifying bot accounts based on their recent GitHub event history N Chidambaram, T Mens, A Decan Proceedings of the 21st International Conference on Mining Software …	9	2024
gawd: A Differencing Tool for GitHub Actions Workflows PR Mazrae, A Decan, T Mens 21st International Conference on Mining Software Repositories	2	2024
A dataset of github actions workflow histories G Cardoen, T Mens, A Decan Proceedings of the 21st International Conference on Mining Software …	12	2024
Quantifying Security Issues in Reusable JavaScript Actions in GitHub Workflows H Onsori Delicheh, A Decan, T Mens 21st International Conference on Mining Software Repositories	14	2024
On the outdatedness of workflows in the GitHub Actions ecosystem A Decan, T Mens, HO Delicheh Journal of Systems and Software 206, 111827	26	2023
An Introduction to Software Ecosystems T Mens, C De Roover Software Ecosystems: Tooling and Analytics, 1-29	7	2023
The GitHub Development Workflow Automation Ecosystems M Wessel, T Mens, A Decan, PR Mazrae Software Ecosystems: Tooling and Analytics, 183-214	22	2023
A dataset of bot and human activities in GitHub N Chidambaram, A Decan, T Mens 2023 IEEE/ACM 20th International Conference on Mining Software Repositories …	13	2023
On the usage, co-usage and migration of CI/CD tools: A qualitative analysis P Rostami Mazrae, T Mens, M Golzadeh, A Decan Empirical Software Engineering 28 (2), 52	66	2023

Software Ecosystems

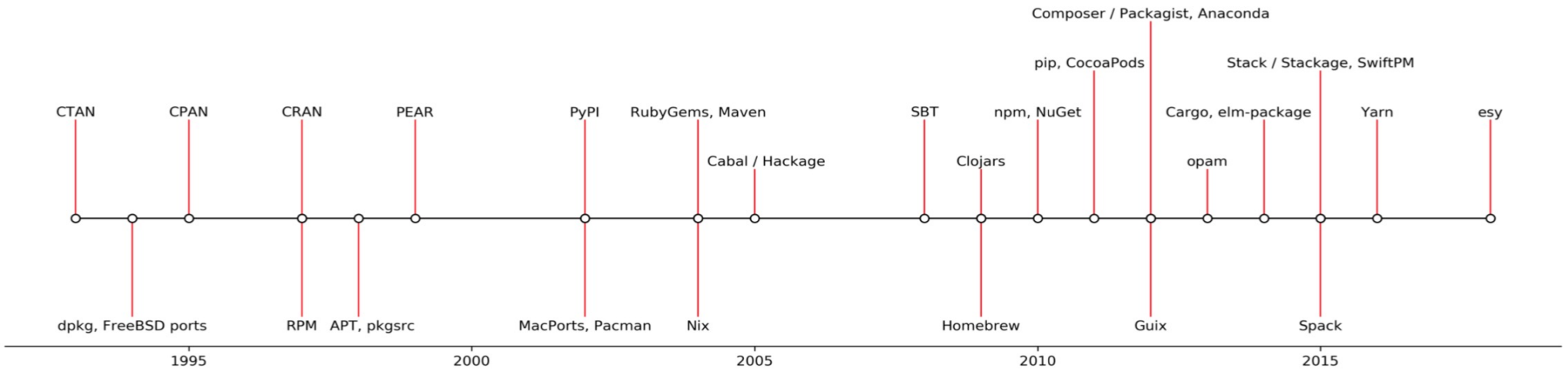
"a collection of software ~~projects~~ *components* which are developed and co-evolve in the same environment"

[Mircea Lungu 2009, PhD thesis]

A **software packaging ecosystem** is composed of one or more

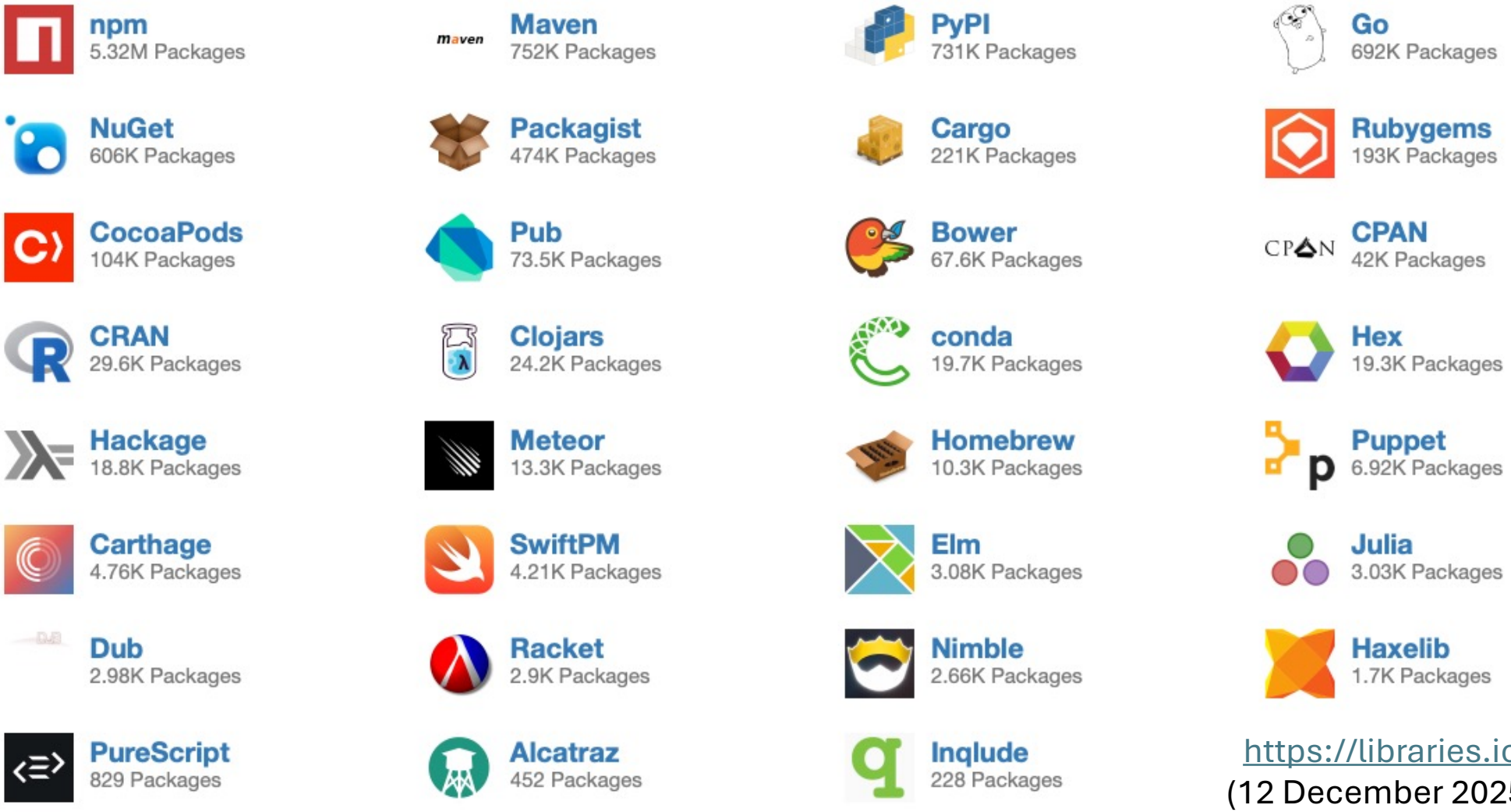
- **package registries**, i.e., collections of ***interdependent*** software packages (a.k.a. libraries)
- **package managers** to select, distribute and install packages from the registry

Packaging ecosystems tend to focus on a specific programming language, OS, application, ...



Software Packaging Ecosystems

https://libraries.io monitors **>10M** open source packages across **32** different package registries



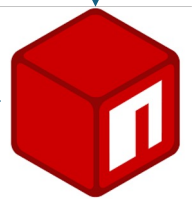
https://libraries.io
(12 December 2025)

Dependency Hell
A “simple” motivating example

```
const express = require('express’)
const app = express()
app.get('/', function (req, res) { res.send("Hello")} )
app.listen(3000)
```

package.json

```
{ "name": "HelloAPI",
  "version": "1.0.0",
  "main": "index.js",
  "dependencies":
    { "express": "4.18.2” }
}
```



express 4.18.2

Wide dependency tree
- 31 direct dependencies
- 17 development dependencies

Deep dependency tree
- 40 indirect dependencies

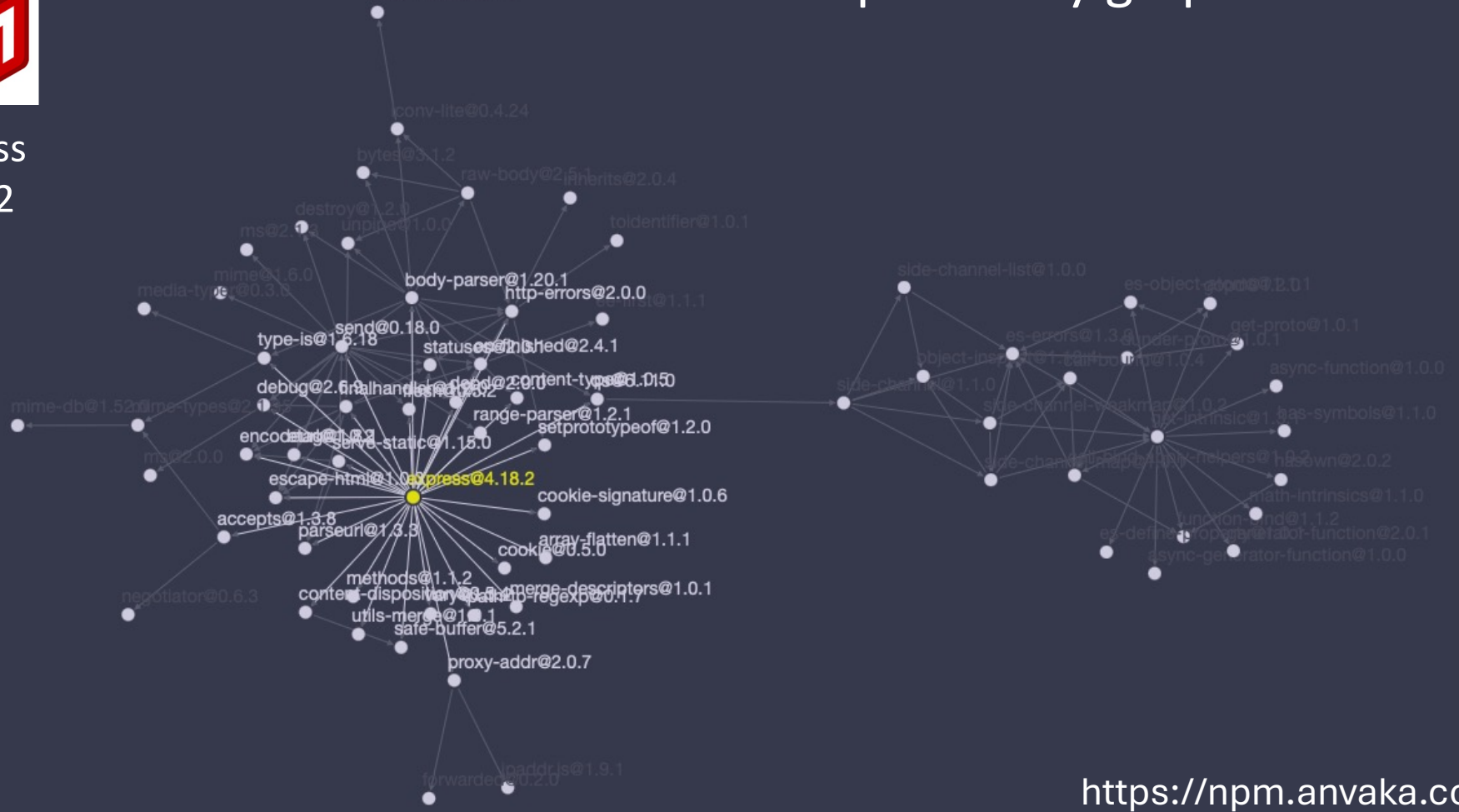
Vulnerable
- 2 known vulnerabilities

Outdated (on 1/12/2025)
- More recent patch, minor and major releases available

Dependency Hell
A “simple” motivating example

express
4.18.2

Direct and indirect dependency graph



https://npm.anvaka.com/

Dependency Hell
A “simple” motivating example

express
4.18.2

Vulnerability lag

- 2 ***direct*** vulnerabilities of medium severity that would be fixed by upgrading to a more recent minor release
  - cross-site sripting
    fix available in 4.20.0
  - open redirect
    fix available in 4.19.2
- There are likely many more ***indirect*** vulnerabilities as well

https://security.snyk.io/package/npm/express/4.18.2

Version	Published	Direct Vulnerabilities
4.22.1	1 Dec, 2025	0 C 0 H 0 M 0 L
5.2.1	1 Dec, 2025	0 C 0 H 0 M 0 L
5.2.0	1 Dec, 2025	0 C 0 H 0 M 0 L
4.22.0	1 Dec, 2025	0 C 0 H 0 M 0 L
5.1.0	31 Mar, 2025	0 C 0 H 0 M 0 L
4.21.2	5 Dec, 2024	0 C 0 H 0 M 0 L
5.0.1	8 Oct, 2024	0 C 0 H 0 M 0 L
4.21.1	8 Oct, 2024	0 C 0 H 0 M 0 L
4.21.0	12 Sep, 2024	0 C 0 H 0 M 0 L
5.0.0	10 Sep, 2024	0 C 0 H 0 M 0 L
4.20.0	10 Sep, 2024	0 C 0 H 0 M 0 L
5.0.0-beta.3	25 Mar, 2024	0 C 0 H 1 M 0 L
4.19.2	25 Mar, 2024	0 C 0 H 1 M 0 L
5.0.0-beta.2	21 Mar, 2024	0 C 0 H 2 M 0 L
4.19.1	20 Mar, 2024	0 C 0 H 2 M 0 L
4.19.0	20 Mar, 2024	0 C 0 H 2 M 0 L
4.18.3	29 Feb, 2024	0 C 0 H 2 M 0 L
4.18.2	8 Oct, 2022	0 C 0 H 2 M 0 L

Dependency Hell
A “simple” motivating example

express
4.18.2

Outdatedness on 1 Dec, 2025

- 1 patch version behind
  - 4.18.3 released on 29 Feb, 2024
- 4 minor versions behind
  - 4.22.1 released on 1 Dec, 2025
- 1 major version behind
  - 5.0.0 released on 10 Sep, 2024
  - 5.2.1 available on 1 Dec, 2025

Version	Published
4.22.1	1 Dec, 2025
5.2.1	1 Dec, 2025
5.2.0	1 Dec, 2025
4.22.0	1 Dec, 2025
5.1.0	31 Mar, 2025
4.21.2	5 Dec, 2024
5.0.1	8 Oct, 2024
4.21.1	8 Oct, 2024
4.21.0	12 Sep, 2024
5.0.0	10 Sep, 2024
4.20.0	10 Sep, 2024
5.0.0-beta.3	25 Mar, 2024
4.19.2	25 Mar, 2024
5.0.0-beta.2	21 Mar, 2024
4.19.1	20 Mar, 2024
4.19.0	20 Mar, 2024
4.18.3	29 Feb, 2024
4.18.2	8 Oct, 2022

- Using semantic versioning constraints reduces outdatedness
- “Technical lag” framework can help in assessing outdatedness

On the evolution of technical lag in the npm package dependency network
Decan et al. 2018, ICSME conference
DOI: 10.1109/ICSME.2018.00050

A formal framework for measuring technical lag in component repositories
Zerouali et al. 2019, Journal of Software: Evolution and Process
DOI: 10.1002/smr.2157

Dependency Hell
A “simple” motivating example

express
4.y.z

“Hidden” Complexity of transitive dependency grap tends to increase over time

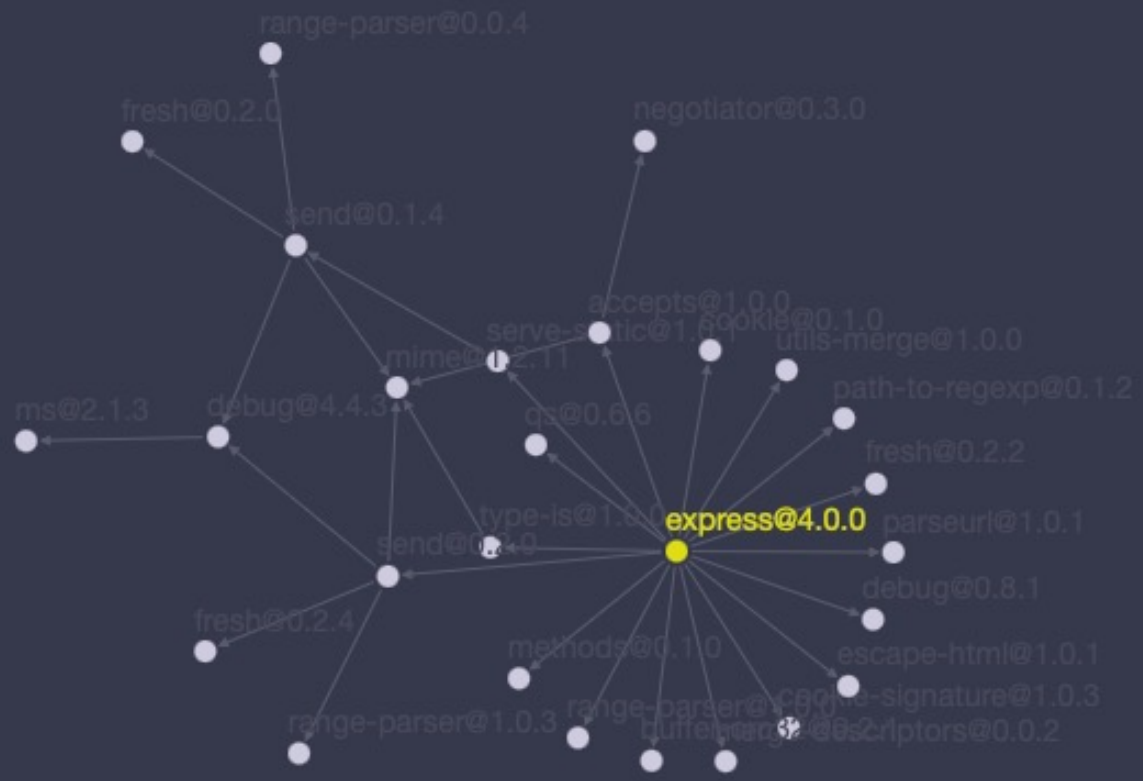
release	date	#transitive deps.
4.0.0	2014-04-09	27
4.1.0	2014-04-25	23
4.5.0	2014-07-05	34
4.10.0	2014-10-24	36
4.15.0	2017-03-01	46
4.18.2	2022-10-08	71
4.20.0	2024-04-09	74
4.22.1	2025-12-01	75

https://npm.anvaka.com/

Dependency Hell
A “simple” motivating example

express
4.0.0

Complexity of transitive dependency graph tends to increase

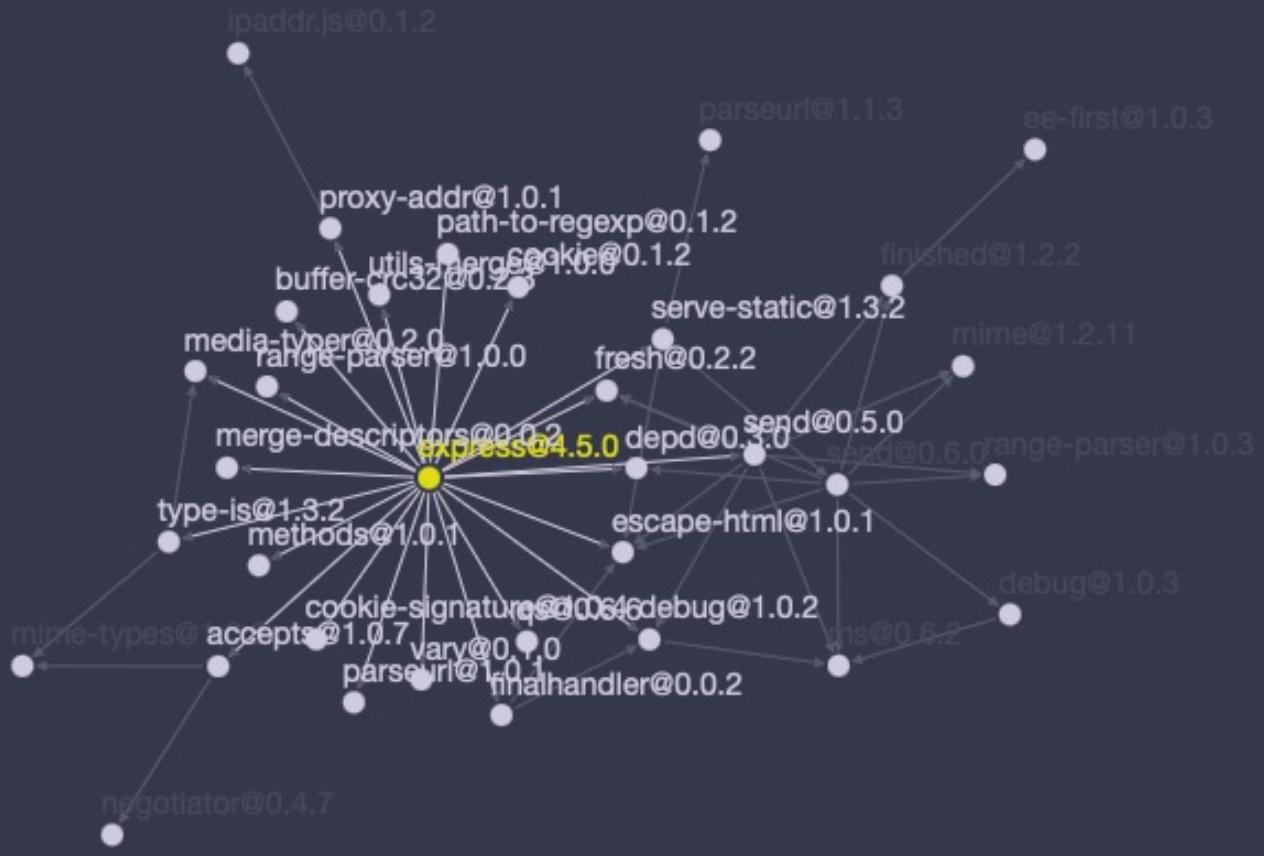


27 transitive dependencies

https://npm.anvaka.com/

Dependency Hell
A “simple” motivating example

express
4.5.0

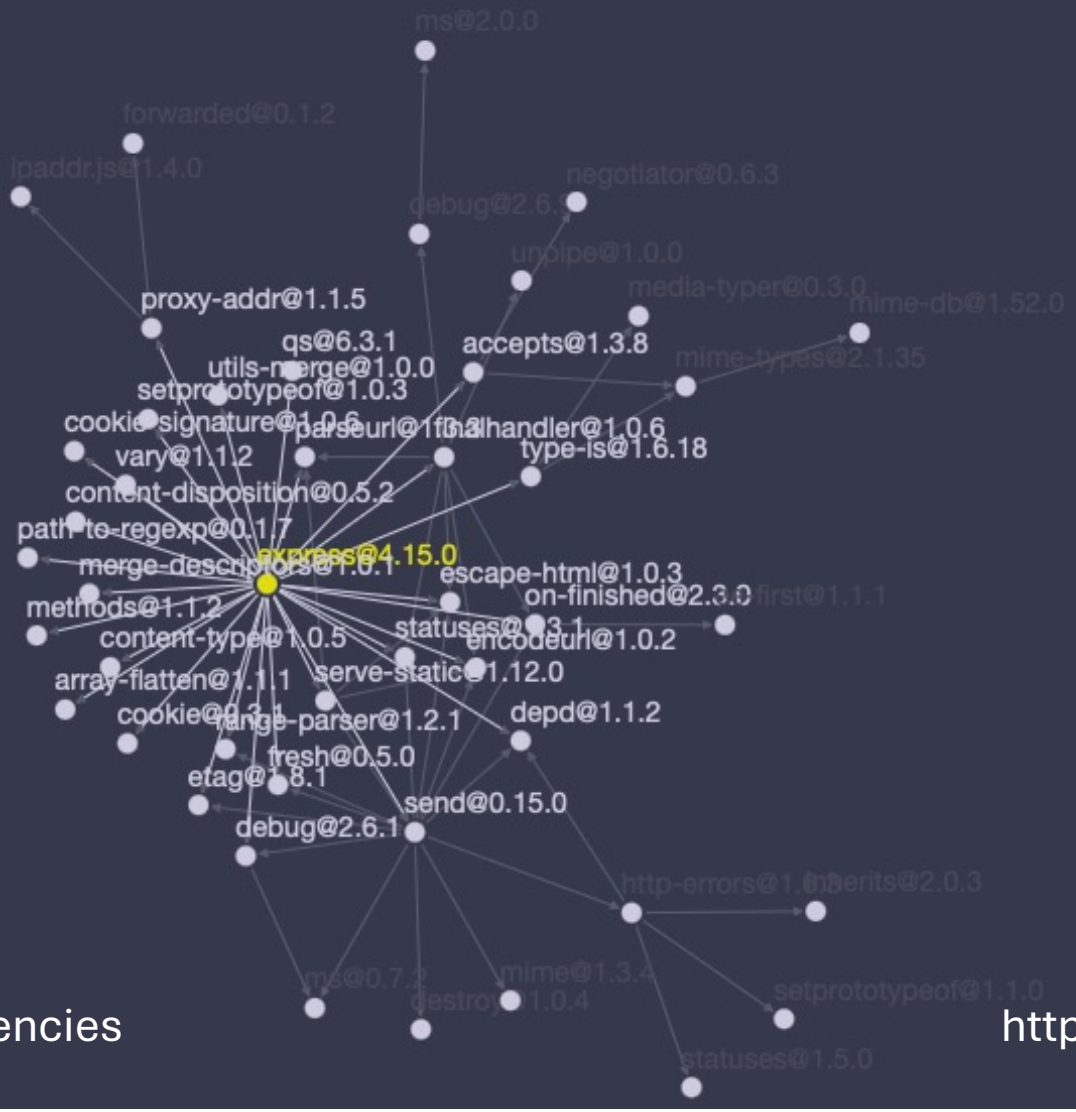


34 transitive dependencies

https://npm.anvaka.com/

Dependency Hell
A “simple” motivating example

express
4.15.0

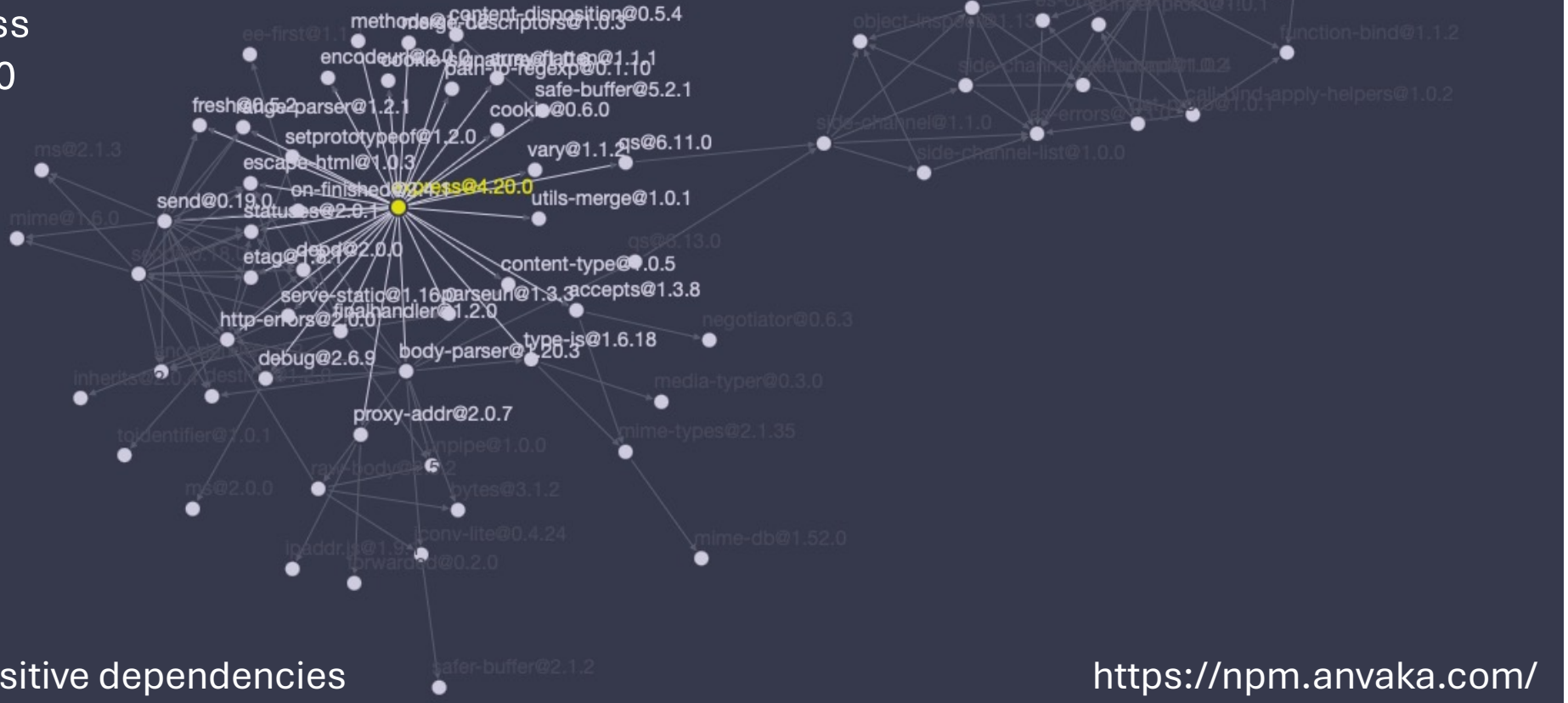


46 transitive dependencies

https://npm.anvaka.com/

Dependency Hell
A “simple” motivating example

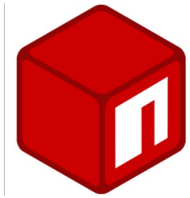
express
4.20.0



74 transitive dependencies

https://npm.anvaka.com/

Dependency Hell
Another Example



request 2.88.2
February 2020

This package has been deprecated

Author message:

`request has been deprecated, see https://github.com/request/request/issues/3142`

request

2.88.2 • Public • Published 6 months ago

Readme | Explore BETA | 20 Dependencies | 50,119 Dependents | 126 Versions

Deprecated!

As of Feb 11th 2020, request is fully deprecated. No new changes are expected land. In fact, none have landed for some time.

For more information about why request is deprecated and possible alternatives refer to this issue.

Request - Simplified HTTP client

npm install request
20 dependencies version 2.88.2 updated 6 months ago
1,506 ★

build passing | coverage 96% | coverage 97% | dependencies up to date | vulnerabilities 0 | gitter join chat

Super simple to use

Request is designed to be the simplest way possible to make http calls. It supports HTTPS and follows redirects by default.

Install

```
> npm i request
```

Weekly Downloads
19,340,243

Version	License
2.88.2	Apache-2.0

Unpacked Size	Total Files
209 kB	17

Issues	Pull Requests
136	46

Homepage
github.com/request/request#readme

Repository
github.com/request/request

Dependency Hell
Another Example

request 2.88.2
February 2020

Vulnerability	Vulnerable Version
M Server-side Request Forgery (SSRF)	
request is a simplified http request client. Affected versions of this package are vulnerable to Server-side Request Forgery (SSRF) due to insufficient checks in the `lib/redirect.js` file by allowing insecure redirects in the default configuration, via an attacker-controller server that does a cross-protocol redirect (HTTP to HTTPS, or HTTPS to HTTP). **NOTE:** `request` package has been deprecated, so a fix is not expected. See https://github.com/request/request/issues/3142. How to fix Server-side Request Forgery (SSRF)? A fix was pushed into the `master` branch but not yet published.	*

request / request

Code | Issues 136 | Pull requests 46 | Actions | Security | Insights

- Pulse
- Contributors
- Community
- Commits
- Code frequency
- Dependency graph

Dependency graph

Dependencies | Dependents

Repositories that depend on **request**

5,561,468 Repositories | 55,129 Packages

Observations Dependency Risk

BLACKDUCK®

2025 Open Source Security and Risk Analysis Report

- 97% of today's software applications depend on OSS components
- 911 OSS components on average per application
- Majority of OSS components originates from package registries; npm accounts for 28%
- Majority of dependent components (64%) are *transitive* dependencies
- Dependencies are a frequent source of security vulnerabilities
- Dependencies regularly have licensing conflicts

https://www.blackduck.com/resources/analyst-reports/open-source-security-risk-analysis.html

Observations Dependency Risk

- npm accounted for 4.5 trillion package requests in 2024
  - **up 70%** year-over-year
- PyPI accounted for 530 billion package requests in 2024
  - **up 87%** year-over-year
- Over 512,847 reported malicious packages in 2024
  - **up 156%** year-over-year
- 80% of application dependencies did not upgrade vulnerable versions for over a year
  - even though safer alternatives are readily available
- Only 10.5% of available OSS packages are actively used
- Even small applications have unmanageable workflows due to their number of dependencies

https://www.sonatype.com/state-of-the-software-supply-chain

Observations Dependency Risk

Five main risks need to be addressed to improve health and security of OSS packages:

1. Lack of a standardized naming schema for software components.
   ➔ New SWHID standard (ISO/IEC 18670, April 2025)
2. Complexities associated with package versions.
3. Many widely used OSS components are developed by only a handful of contributors.
4. Increasing importance of individual developer account security.
5. “Legacy” OSS packages persist due to their sheer usage (cf. “request” example)
   1. Older major package versions are not replaced by more recent versions (due to backward incompatibility or upgrade effort)
   2. Unmaintained packages are not replaced by actively maintained alternatives with similar functionality.

https://openssf.org/resources/census-ii-of-free-and-open-source-software-application-libraries/

Catalogue of Dependency Challenges

Packaging ecosystems are **fragile to single points-of-failure** due to wide and deep transitive dependency networks

1. Outdated dependencies
2. Breaking changes
3. Incompatible dependencies
4. Bloated dependencies
5. Deprecated dependencies
7. Incompatible licenses
8. Vulnerable dependencies
9. Malicious dependencies and supply chain attacks
10. Abandoned/unmaintained dependencies

Adaptation based on https://ceur-ws.org/Vol-3941/BENEVOL2024_TECH_paper15.pdf

Software Packaging Ecosystems
Ecosystem comparisons

Packaging ecosystems tend to use ecosystem-specific formats, policies and tools

How to break an API: cost negotiation and community values in three software ecosystems
Bogart et al. 2016, FSE conference. DOI: 10.1145/2950290.2950325

An empirical comparison of dependency issues in OSS packaging ecosystems
Decan et al. 2017, SANER conference. DOI: 10.1109/SANER.2017.7884604

An empirical comparison of dependency network evolution in seven software packaging ecosystems
Decan et al. 2018, Empirical Software Engineering. DOI: 10.1007/s10664-017-9589-y

Dependency versioning in the wild
Dietrich et al. 2019, MSR conference. DOI: 10.1109/MSR.2019.00061

When and how to make breaking changes: Policies and practices in 18 open source software ecosystems
Kästner et al. 2021, ACM TOSEM. DOI: 10.1145/3447245

What do package dependencies tell us about semantic versioning?
Decan & Mens 2021, IEEE TSE. DOI: 10.1109/TSE.2019.2918315

Back to the past: Analysing backporting practices in package dependency networks
Decan et al. 2022, IEEE TSE. DOI: 10.1109/TSE.2021.3112204

Software Packaging Ecosystems
Empirical studies of specific ecosysems

npm

On the impact of security vulnerabilities in the npm package dependency network
Decan et al. 2018, MSR conference
DOI: 10.1145/3196398.3196401

On the evolution of technical lag in the npm package dependency network
Decan et al. 2018, ICSME conference
DOI: 10.1109/ICSME.2018.00050

An empirical study of dependency downgrades in the npm ecosystem
Cogo et al. 2021, IEEE TSE
DOI: 10.1109/TSE.2019.2952130

Deprecation of packages and releases in software ecosystems: A case study on npm
Cogo et al. 2022, IEEE TSE
DOI: 10.1109/TSE.2021.3055123

Other packaging ecosystems

A comprehensive study of bloated dependencies in the Maven ecosystem
Soto-Valero et al. 2021, EMSE journal
DOI: 10.1007/s10664-020-09914-8

Breaking bad? Semantic versioning and impact of breaking changes in Maven Central
Ochoa et al. 2022, EMSE journal
DOI: 10.1007/s10664-021-10052-y

An empirical study of yanked releases in the Rust package registry
Li et al. 2023, IEEE TSE
DOI: 10.1109/TSE.2022.3152148

Towards better dependency management: A first look at dependency smells in Python projects
Cao et al. 2023, IEEE TSE
DOI: 10.1109/TSE.2022.3191353

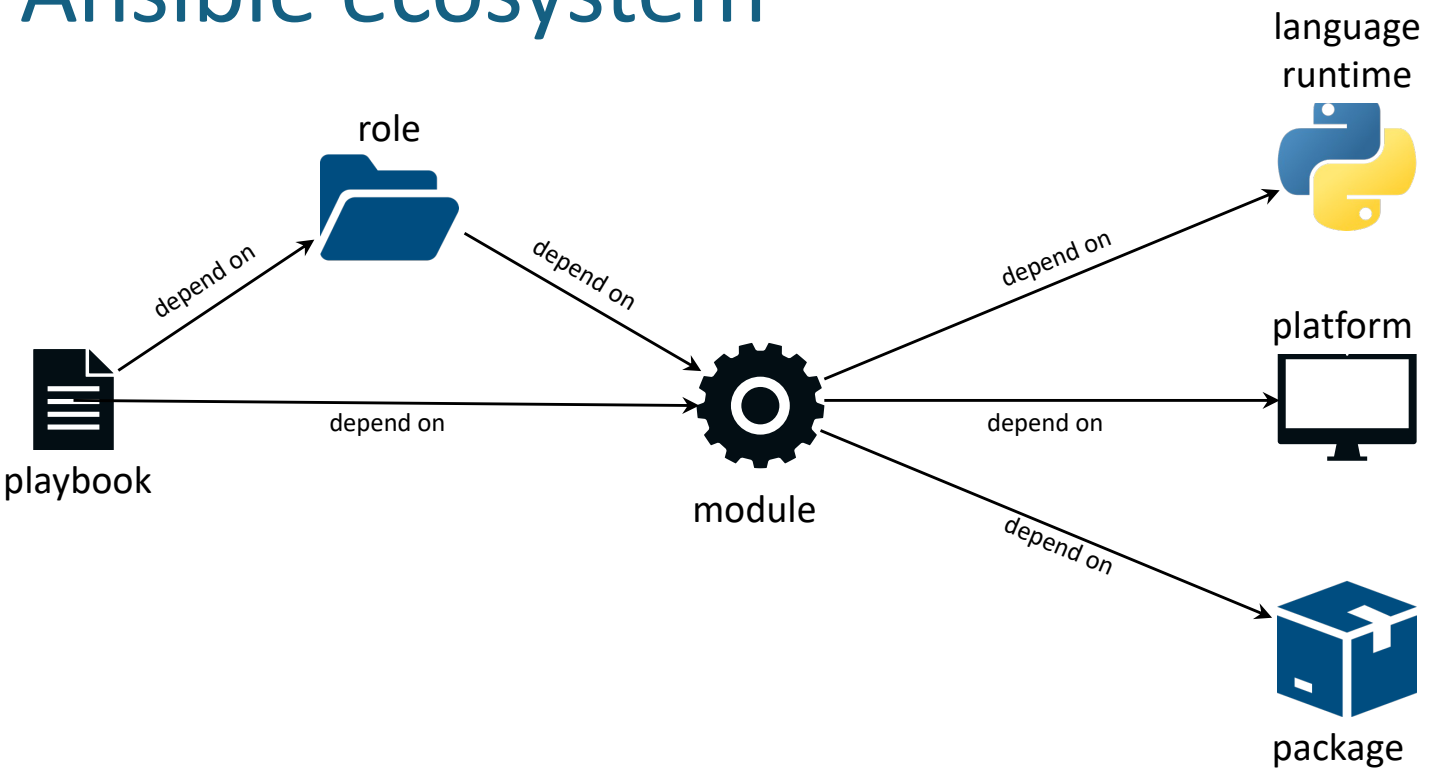
Beyond Packaging Ecosystems

- Configuration-as-Code (CaC) ecosystems
  - CI/CD workflows and pipelines and their dependencies
    (e.g. GitHub Actions, GitLab CI/CD, Bitbucket Pipelines, CircleCI orbs, Jenkins plugins)

- Infrastructure-as-Code (IaC) ecosystems
  - Ansible (plugin dependencies), Puppet, Terraform, Kubernetes (Helm charts), Pulumi, Docker

Beyond Packaging Ecosystems Infrastructure-as-Code ecosystems

The Ansible ecosystem

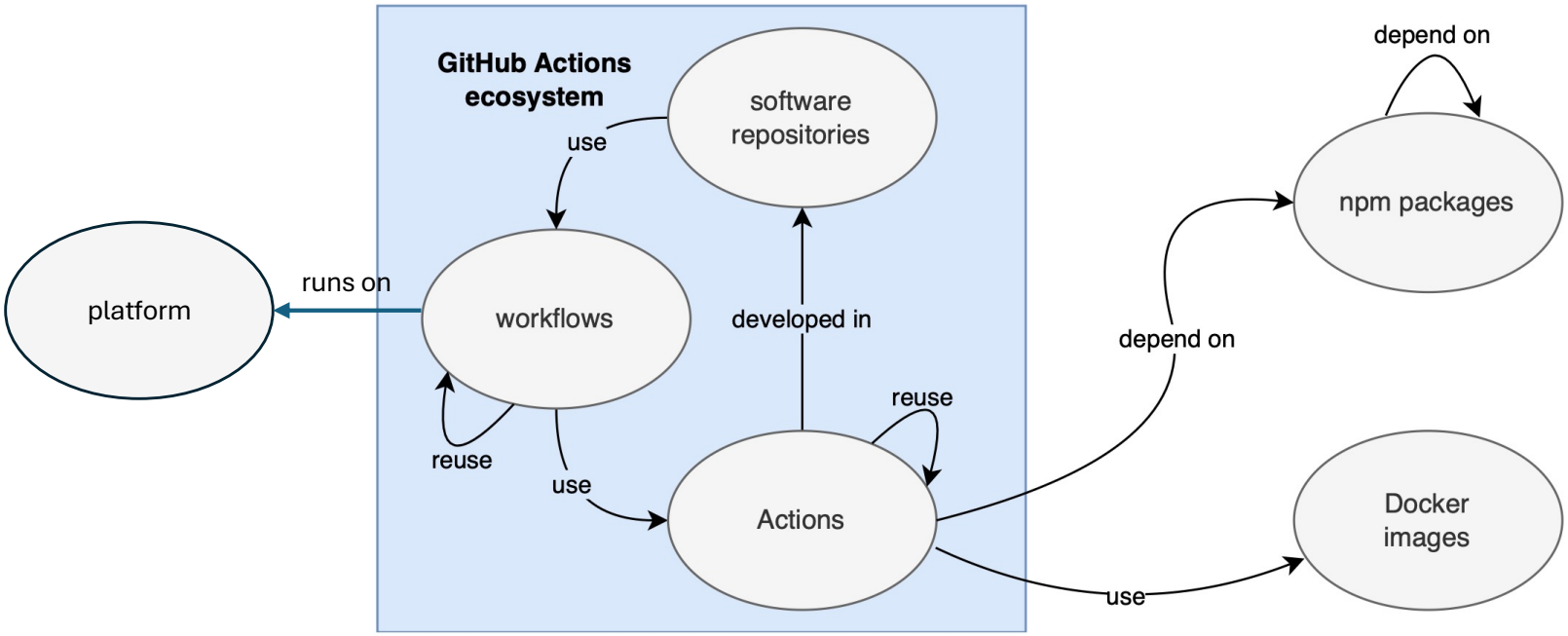


Analysing software supply chains of infrastructure as code: Extraction of Ansible plugin dependencies
Opdebeeck et al. 2025, SANER conference
DOI: 10.1109/SANER64311.2025.00025

Beyond Packaging Ecosystems
Configuration-as-Code ecosystems

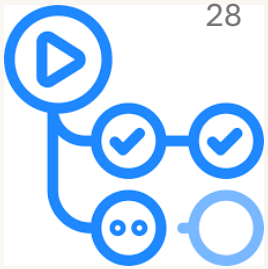
The **GitHub Actions ecosystem** of CI/CD workflow automation

- **worklows** depend on reusable Actions that may depend on external components (e.g. npm, Docker)
- This significantly **expands the attack surface and dependency risks** for GitHub repos, users and organisations.



The GitHub Development Workflow Automation Ecosystems
Wessel et al. 2023, In "Software Ecosystems: Tooling and Analytics"
DOI: 10.1007/978-3-031-36060-2_8

The GitHub Actions Ecosystem

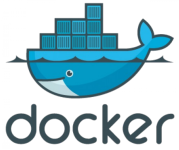


JavaScript Actions

setup-java / package.json

```
28      "dependencies": {
29        "@actions/cache": "^3.0.4",
30        "@actions/core": "^1.10.0",
31        "@actions/exec": "^1.0.4",
32        "@actions/glob": "^0.2.0",
33        "@actions/http-client": "^1.0.11",
34        "@actions/io": "^1.0.2",
35        "@actions/tool-cache": "^1.6.1",
36        "semver": "^7.3.4",
37        "xmlbuilder2": "^2.4.0"
38      },
```

Docker Actions



super-linter / Dockerfile

Code | Blame | 441 lines (381 loc) · 14.8 KB

```
10  FROM tenable/terrascan:1.18.1 as terrascan
11  FROM alpine/terragrunt:1.4.6 as terragrunt
12  FROM assignuser/chktex-alpine:v0.1.1 as chktex
13  FROM cljkondo/clj-kondo:2023.05.18-alpine as clj-kondo
14  FROM dotenvlinter/dotenv-linter:3.3.0 as dotenv-linter
15  FROM ghcr.io/awkbar-devops/clang-format:v1.0.2 as clang-format
16  FROM ghcr.io/terraform-linters/tflint-bundle:v0.46.1.1 as tflint
17  FROM ghcr.io/yannh/kubeconform:v0.6.1 as kubeconfrm
18  FROM golangci/golangci-lint:v1.52.2 as golangci-lint
19  FROM hadolint/hadolint:latest-alpine as dockerfile-lint
20  FROM hashicorp/terraform:1.4.6 as terraform
21  FROM koalaman/shellcheck:v0.9.0 as shellcheck
22  FROM mstruebing/editorconfig-checker:2.7.0 as editorconfig-checker
23  FROM mvdan/shfmt:v3.6.0 as shfmt
24  FROM rhysd/actionlint:1.6.24 as actionlint
25  FROM scalameta/scalafmt:v3.7.3 as scalafmt
26  FROM zricethezav/gitleaks:v8.16.3 as gitleaks
27  FROM yoheimuta/protolint:0.44.0 as protolint
28
29  ##################
30  # Get base image #
31  ##################
32  FROM python:3.11.3-alpine3.17 as base_image
```

Composite Actions

action / action.yml

Code | Blame | 78 lines (78 loc) · 2.37 KB

```
46  runs:
47    using: 'composite'
48    steps:
49      - uses: actions/setup-python@v4
50        with:
51          python-version: '3.10'
52      - run: |
53          pip install --upgrade pip
54          python3.10 -m venv env
55          source env/bin/activate
56          if [ ${{ inputs.version }} == "latest" ] ; then
57            pip install schemathesis
58          else
59            pip install schemathesis==${{ inputs.version }}
60          fi
61        shell: bash
62      - run: |
63          source env/bin/activate
64          schemathesis run \
65            ${{ inputs.schema }} \
66            --hypothesis-database=:memory: \
67            --hypothesis-max-examples=${{ inputs.max-examples }}
68            --checks=${{ inputs.checks }} \
69            ${{ inputs.args }} \
70        shell: bash
```

Beyond Packaging Ecosystems Infrastructure-as-Code ecosystems

On the Relation between Outdated Docker Containers, Severity Vulnerabilities, and Bugs
Zerouali et al. 2019, SANER conference
DOI: 10.1109/SANER.2019.8668013

The Docker Hub Image Inheritance Network: Construction and Empirical Insights
Opdebeeck et al. 2023, SCAM conference
DOI: 10.1109/SCAM59687.2023.00029

Helm Charts for Kubernetes Applications: Evolution, Outdatedness and Security Risks
Zerouali et al. 2023, MSR conference
DOI: 10.1109/MSR59073.2023.00078

Infrastructure-as-code ecosystems
Opdebeeck et al. 2023. In “Software Ecosystems: Tooling and Analytics”
DOI: 10.1007/978-3-031-36060-2_9

Analysing software supply chains of infrastructure as code: Extraction of Ansible plugin dependencies
Opdebeeck et al. 2025, SANER conference
DOI: 10.1109/SANER64311.2025.00025

Beyond Packaging Ecosystems
The GitHub Actions Ecosystem

Usage and Adoption

On the rise and fall of CI services in GitHub
Golzadeh et al. 2022, SANER conference
DOI: 10.1109/SANER53432.2022.00084

On the use of GitHub Actions in software development repositories
Decan et al. 2022, ICSME conference
DOI: 10.1109/ICSME55016.2022.00029

On the usage, co-usage and migration of CI/CD tools: A qualitative analysis
Rostami Mazrae et al. 2023, Empirical Softw. Eng. journal
DOI: 10.1007/s10664-022-10285-5

A dataset of Github Actions workflow histories
Cardoen et al. 2024, MSR conference
DOI: 10.1145/3643991.3644867

Reuse, Dependency and Security Analysis

On the outdatedness of workflows in the GitHub Actions ecosystem
Decan et al. 2023, Journal of Systems and Software
DOI: 10.1016/j.jss.2023.111827

Quantifying Security Issues in Reusable JavaScript Actions in GitHub Workflows
Onsori Delicheh et al. 2024, MSR conference
DOI: 10.1145/3643991.3644899

Automation and Reuse Practices in GitHub Actions Workflows: A Practitioner's Perspective
Onsori Delicheh et al. 2026, ACM TOSEM
[Under Review]

An Analysis of Code Clones in GitHub Actions Workflows
Cardoen et al. 2026, SANER conference
DOI: TBD

Next-Gen Software Ecosystems

- Promptware ecosystems

  Villamizar et al. (2025). **Prompts as Software Engineering Artifacts: A Research Agenda and Preliminary Findings**. Product-Focused Software Process Improvement (PROFES), LNCS 16361. DOI: 10.1007/978-3-032-12089-2_32

- Agent(ic) ecosystems (cf. agentic development)
- Specification ecosystems (cf. spec-driven development)
- Domain-knowledge ecosystems (cf. ontologies / knowledge graphs / ...)
- AI model ecosystems
- “copy-paste-reuse” ecosystems (cf. Audris Mockus)

Challenge
Multi-ecosystem dependencies

- Dependencies spanning across multiple ecosystems (different languages / technologies / policies / ...)
- Ecosystems composed of sub-ecosystems
- Ecosystems that partially overlap
- Software+hardware ecosystems

Insight: Exploring Cross-Ecosystem Vulnerability Impacts
Xu et al. 2023, ASE conference. DOI: 10.1145/3551349.3556921

How to understand and improve their structure / governance / health / interactions?

- Lack of empirical (sociotechnical) studies, tools, ...

Catalogue of Dependency Challenges

1. Outdated dependencies
2. Breaking changes
3. Incompatible dependencies
4. Bloated dependencies
5. Deprecated dependencies
7. Incompatible licenses
8. Vulnerable dependencies
9. Malicious dependencies and supply chain attacks
10. Abandoned/unmaintained dependencies

Williams et al. Research directions in software supply chain security ACM TOSEM 2025

Adaptation based on https://ceur-ws.org/Vol-3941/BENEVOL2024_TECH_paper15.pdf

Catalogue of Dependency Challenges

Malicious dependencies and Supply chain attacks

Malicious PyPI, npm, and Ruby Packages Exposed in Ongoing Open-Source Supply Chain Attacks

Jun 04, 2025 Ravie Lakshmanan

Several malicious packages have been uncovered across the npm, Python, and Ruby package repositories that drain funds from cryptocurrency wallets, erase entire codebases after installation, and exfiltrate Telegram API tokens, once again demonstrating the variety of supply chain threats lurking in open-source ecosystems.

https://thehackernews.com/2025/06/malicious-pypi-npm-and-ruby-packages.html

Catalogue of Dependency Challenges
Malicious dependencies

- Malware is software **intentionally** designed to inflict harm
- Types of malware
  - **Viruses** attach to legitimate programs and replicate, spreading upon program execution.
  - **Worms** replicate themselves to spread across computing systems.
  - **Trojan horses** disguise themselves as legitimate software and subsequently perform malicious activities.
  - **Spyware** secretly monitors and collects user information.
  - **Ransomware** encrypts data, demanding a ransom for its release.
  - **Adware** displays unwanted advertisements, often without the user's consent.
  - **Botnets** act as networks of infected computers controlled remotely by threat actors.
  - **Rootkits** conceal the presence of other malware by modifying operating systems.
  - **Protestware**

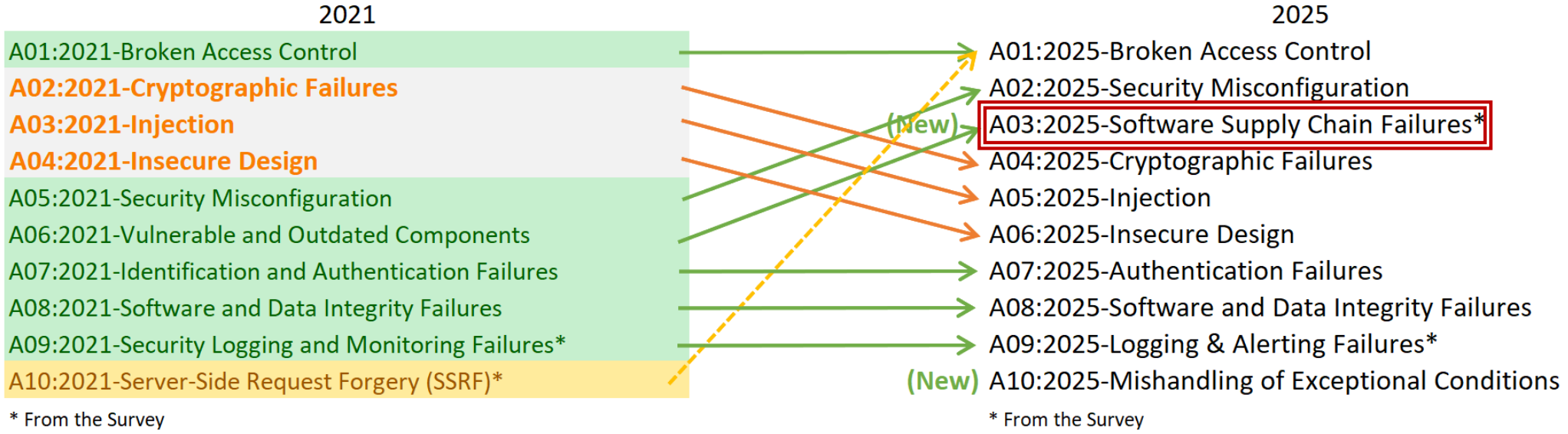
https://www.sonatype.com/resources/guides/understand-open-source-risks
https://www.sonatype.com/state-of-the-software-supply-chain/2024/scale

Software Supply Chain Risks

Top 10:2025 Web Application Security Risks

A03:2025 Software Supply Chain Failures

- An expansion of A06:2021-Vulnerable and Outdated Components to include a broader scope of compromises occurring within or across the entire ecosystem of software dependencies, build systems, and distribution infrastructure. This category was overwhelmingly voted a top concern in the community survey.



https://owasp.org/Top10/2025/

Software Supply Chain Risks
Top 10 CI/CD Security Risks (2022)

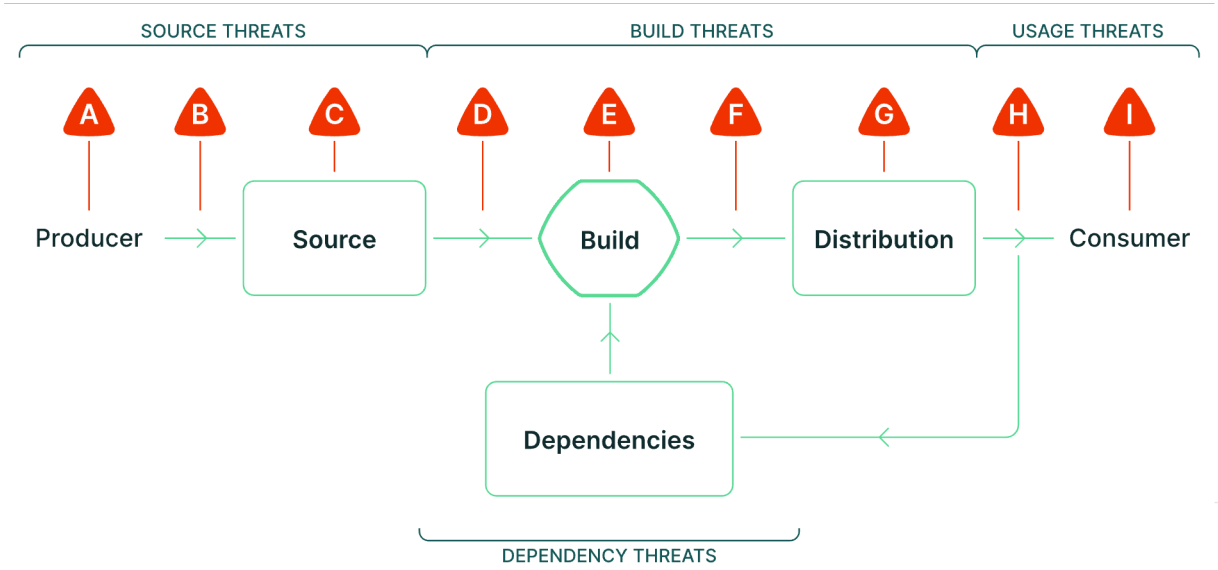
CICD-SEC-3 Dependency Chain Abuse

Abuse flaws relating to how build environments fetch code dependencies, to enable malicious packages to be fetched and executed locally.

- **Dependency confusion:** Publication of malicious packages in public repositories with the same name as internal package names, to trick clients into downloading the malicious package rather than the private one.
- **Dependency hijacking:** Obtaining control of the account of a package maintainer on the public repository, in order to upload a new, malicious version of a widely used package, with the intent of compromising unsuspecting clients who pull the latest version of the package.
- **Typosquatting:** Publication of malicious packages with similar names to those of popular packages in the hope that a developer will misspell a package name and unintentionally fetch the typosquatted package.
- **Brandjacking:** Publication of malicious packages in a manner that is consistent with the naming convention or other characteristics of a specific brand's package, in an attempt to get unsuspecting developers to fetch these packages due to falsely associating them with the trusted brand.

https://owasp.org/www-project-top-10-ci-cd-security-risks/CICD-SEC-03-Dependency-Chain-Abuse

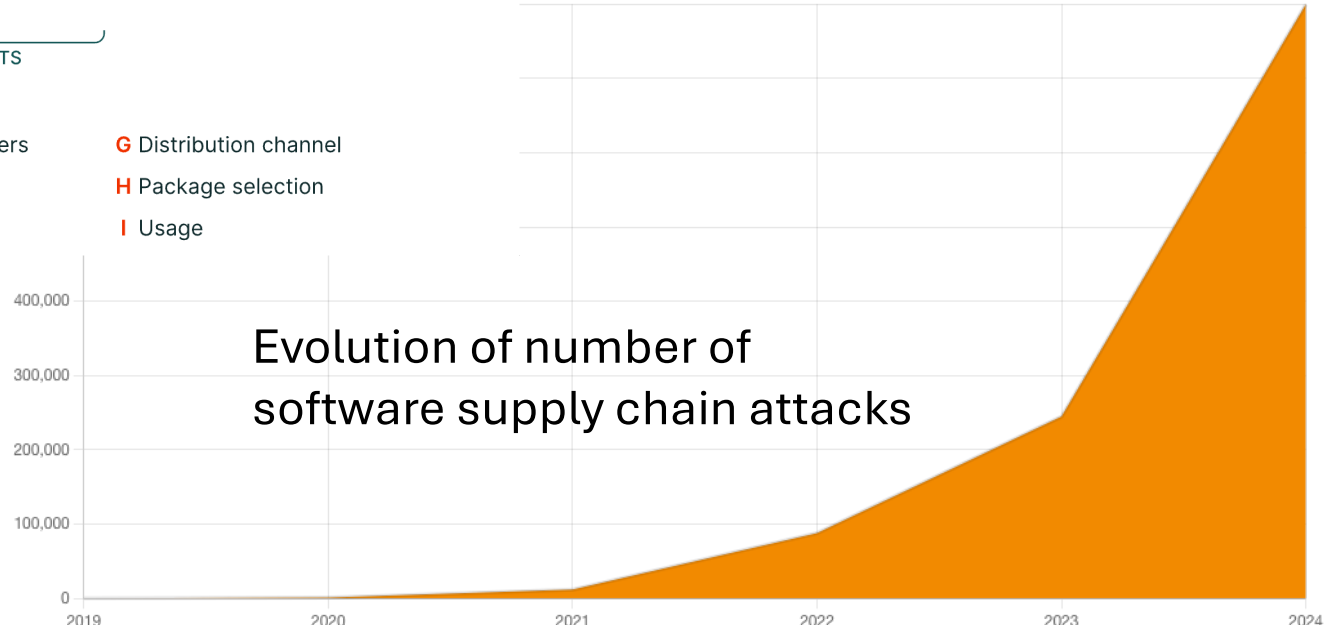
Software Supply Chain Risks
Many different threat vectors



A Producer (entity)
B Authoring & reviewing
C Source code management
D External build parameters
E Build process
F Artifact publication
G Distribution channel
H Package selection
I Usage

https://slsa.dev/spec/v1.2/threats

https://www.sonatype.com/state-of-the-software-supply-chain/2024/10-year-look



Evolution of number of software supply chain attacks

Software Supply Chain Risks Mitigation Strategies

- Software Composition Analysis (SCA) tools
  - Sonatype, Snyk, Synopsis (Black Duck), Mend, OWASP Dependency-Check, Endor Labs, ...
- OpenSSF Scorecard
  https://scorecard.dev
- Software Bill of Materials (SBOM)
  - formally structured lists of all software components present in a software product, including their licenses, versions, security vulnerabilities, and vendors
    - SPDX, CycloneDX
  - imposed or recommended by
    - US Executive Order 14028 https://www.federalregister.gov/d/2021-10460
    - EU Cyber Resilience Act https://www.cyberresilienceact.eu
- Supply chain Levels of Software Artefacts (SLSA)
  https://slsa.dev/spec/v1.2/threats
- Reproducible builds
  - a set of software development practices that create an independently-verifiable path from source to binary code
  - increases trust, security, transparency, resilience against attacks, regulatory compliance, licensing

  https://reproducible-builds.org

Bringing AI into the picture

Fundamental theorem of software engineering (FTSE):

“We can solve any problem by introducing an extra level of indirection”

"…except for the problem of too many indirections”

https://en.wikipedia.org/wiki/Fundamental_theorem_of_software_engineering

Risks of AI Usage

Top 10 security risks for LLM Applications 2025

GenAI SECURITY PROJECT
genai.owasp.org

LLM03:2025 Supply Chain Vulnerabilities

Common Examples of Risks

1. Traditional Third-party Package Vulnerabilities
2. Licensing Risks
3. Outdated or Deprecated Models
4. Vulnerable Pre-Trained Model
5. Weak Model Provenance
6. Vulnerable LoRA adapters
7. Exploit Collaborative Development Processes
8. LLM Model on Device supply-chain vulnerabilities
9. Unclear Terms and Conditions and Data Privacy Policies

All traditional dependency challenges and software supply chain risks also apply to LLM applications!

https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/

Risks of AI Usage

Top 10 security risks for Agentic Applications 2026

GenAI SECURITY PROJECT
genai.owasp.org

ASI04:025 Supply Chain Vulnerabilities

- Arise when agents, tools, and related artefacts they work with are provided by third parties and may be malicious, compromised, or tampered with in transit. These can be both static and dynamical sourced components, including models and model weights, tools, plug-ins, datasets, other agents, agentic interfaces - MCP (Model Context Protocol), A2A (Agent2Agent) - agentic registries and related artifacts, or update channels. These dependencies may introduce unsafe code, hidden instructions, or deceptive behaviors into the agent's execution chain.
- While **LLM03:2025** focuses on **static** dependencies, agentic ecosystems often compose capabilities at **runtime,** thereby increasing the attack surface. This distributed run-time coordination - combined with agentic autonomy - creates a live supply chain that can cascade vulnerabilities across agents. These shifts focus from manifest to run-time security of a diverse and often opaque component. Tackling this problem requires careful development-time tooling and runtime orchestration, where components are dynamically loaded, shared, and trusted.

https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/

Risks of AI usage

- LLMs (and agentic applications) are inherently insecurable
- Attackers can use AI to automate the finding and exploitation of flaws or vulnerabilities in software
- AI hallucinations fuel typosquatting
- AI tools can fake contributions to OSS projects, making it harder to spot bad actors. The ability of LLMs to mimic human interaction increases the chances of malicious code slipping through unnoticed through phishing and other attack vectors that can be automated.
- Malicious actors can tamper with LLM training data or the model itself, injecting harmful code or biases that result in misleading or malicious content.
- Improperly secured LLMs may leak sensitive information, either through generated text or attacks targeting the model's architecture.

https://openssf.org/blog/2025/01/23/predictions-for-open-source-security-in-2025-ai-state-actors-and-supply-chains

Risks of AI usage

- Responses on Stack Overflow's *2024 Developer Survey*
  - Code in software packages is likely to become (at least partly) generated by AI tools: the number of respondents using AI tools has increased from 44% to 62% in 2024.
  - Almost half (46%) of respondents stated they do not trust the accuracy of AI-generated code.
  - 61.7% said they often have security or ethical concerns about code generated by AI tools.
- The Hidden Costs of Coding with Generative AI (Anderson et al. 2025, MIT Sloan Management Review & Report)
  - Developers report about code duplications, integration problems, dependency conflicts, lack of context awareness, and a myriad of other problems that come with coding with AI.
  - Layering AI-generated code on top of legacy systems exacerbates technical debt through tangled dependencies that slow future development and destabilize systems even more.

https://survey.stackoverflow.co/2024/ai

https://tribunecontentagency.com/article/the-hidden-costs-of-coding-with-generative-ai/

Risks of AI Uage
Mitigation Strategies

- Use AI to fight AI
  - Use AI for automated detection and fixing of bugs and vulnerabilities, automated dependency management, improve code quality, reduce technical debt, flag suspicious contributor behaviour, and so on
  - But how to avoid these tools becoming compromised?
- Use AI BOM
  - https://spdx.dev/implementing-an-ai-bom/
- Avoid using AI in the software supply chain

- Ensure full transparency, provenance and reproducibility
  - Is this even possible in presence of AI components?

Final Reflections

- How to address/cope with the ecological footprint of AI? Can we make it more sustainable?
- Research needed that quantitatively assesses the effect/impact of AI on/in software ecosystems
  + provide solutions to reduce negative impact (e.g. code slop)
- How to teach/train/educate the next generation of CS/SE students and practitioners?
- What will future software ecosystems look like?
  - What will remain of what we have achieved in component-based/open-source/collaborative development over the last 30 years? Will we still need software code? Programming languages?
- How to deal with “unpredictability”, “loss of control”, “vendor lock-in”?