

Surviving in Dependency Hell 🥵

c0c0n 2023 | Kumar Ashwin

About Me

Kumar Ashwin

- Security Engineer
- Deals in Web, Cloud & Software Supply Chain Security
- Talks & Trainings - c0c0n, x33fcon, ...

@0xCardinal (<https://0xcardinal.com>) on socials

Agenda

Premise

Strategies



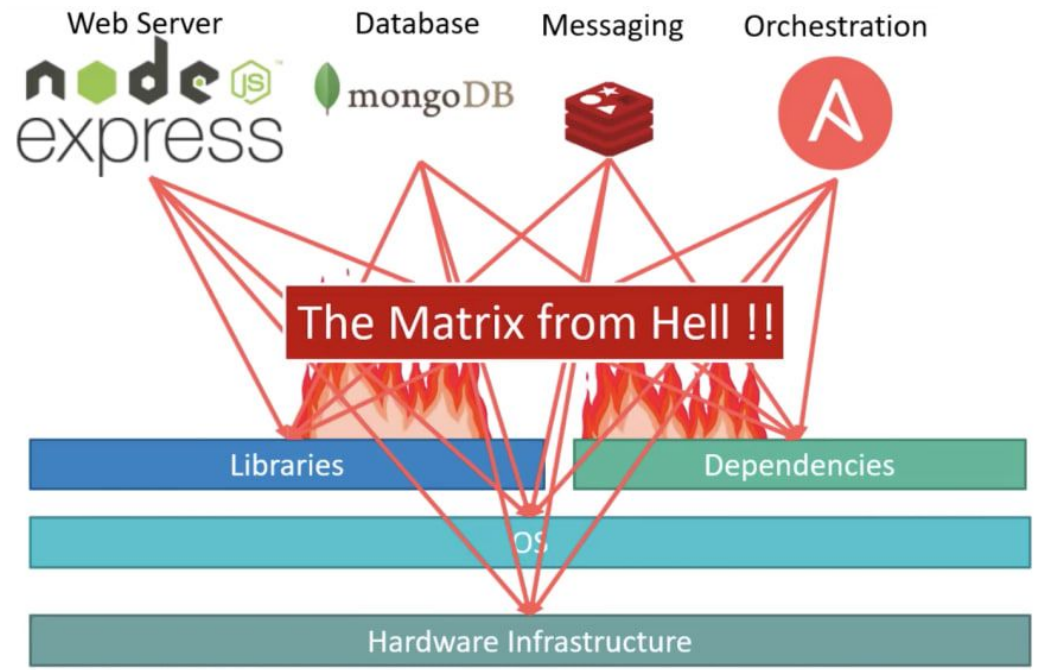
Conclusion

Disclaimer

The research was conducted within the Node Ecosystem, but the strategies discussed in the slides can be applied to other package ecosystems as well.

***We will not** be talking about creating **Dependency Heaven**,
but **will** talk about how to be the **Lucifer in the hell**.*

Matrix Of Hell



Source: KodeKloud

Dependencies

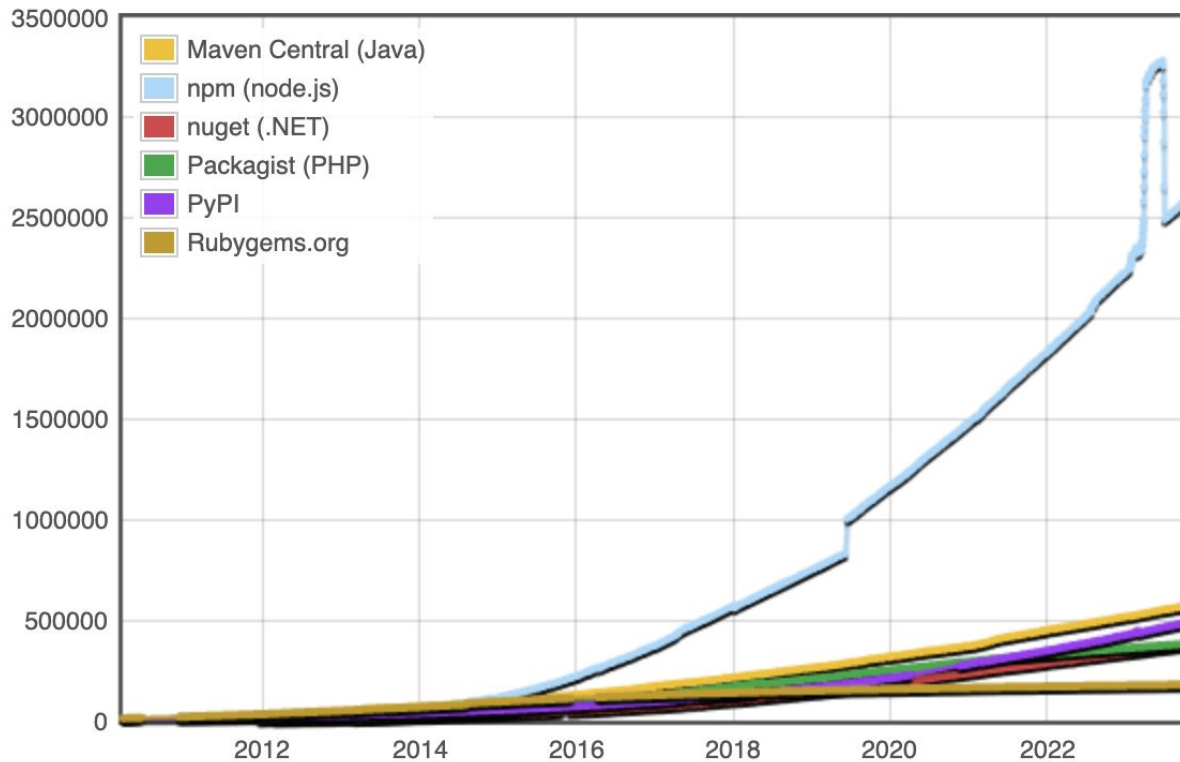
Dependency is a term used when your code depends on someone else's code usually someone external.

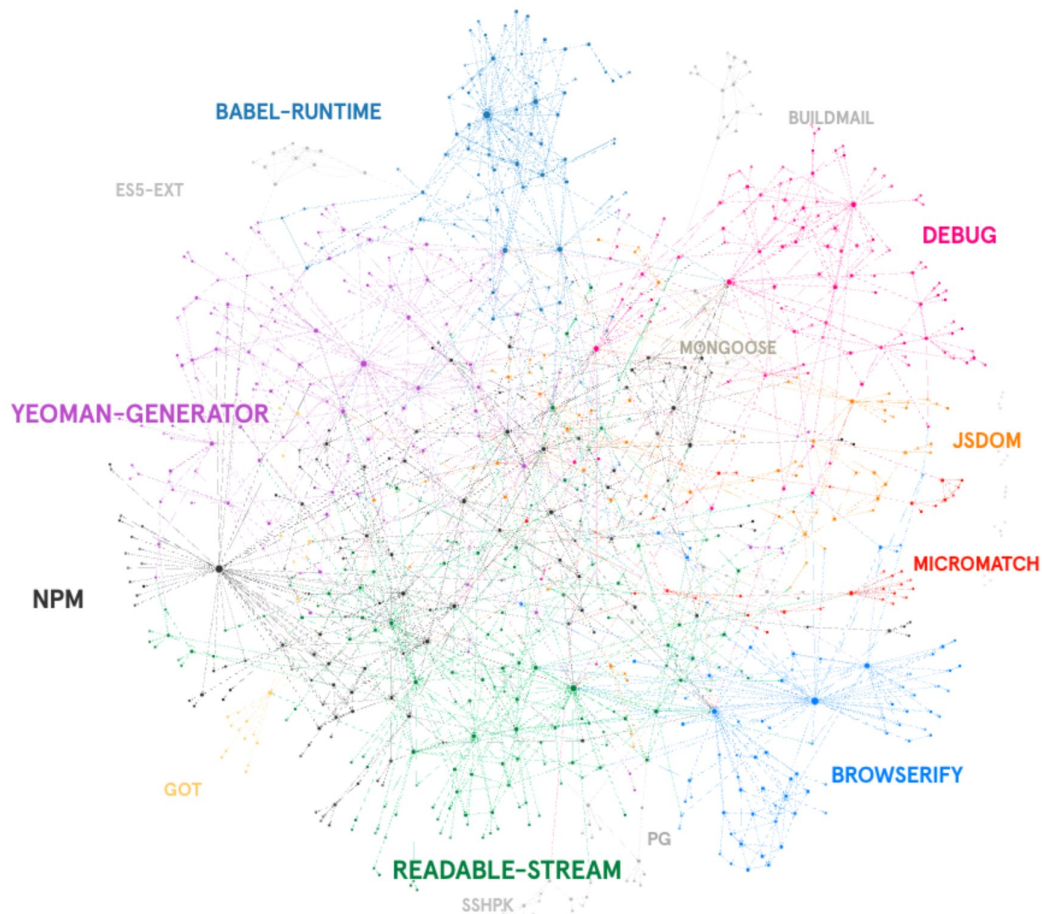
Dependencies

Types of Dependencies

- Direct
- Transitive

Packages

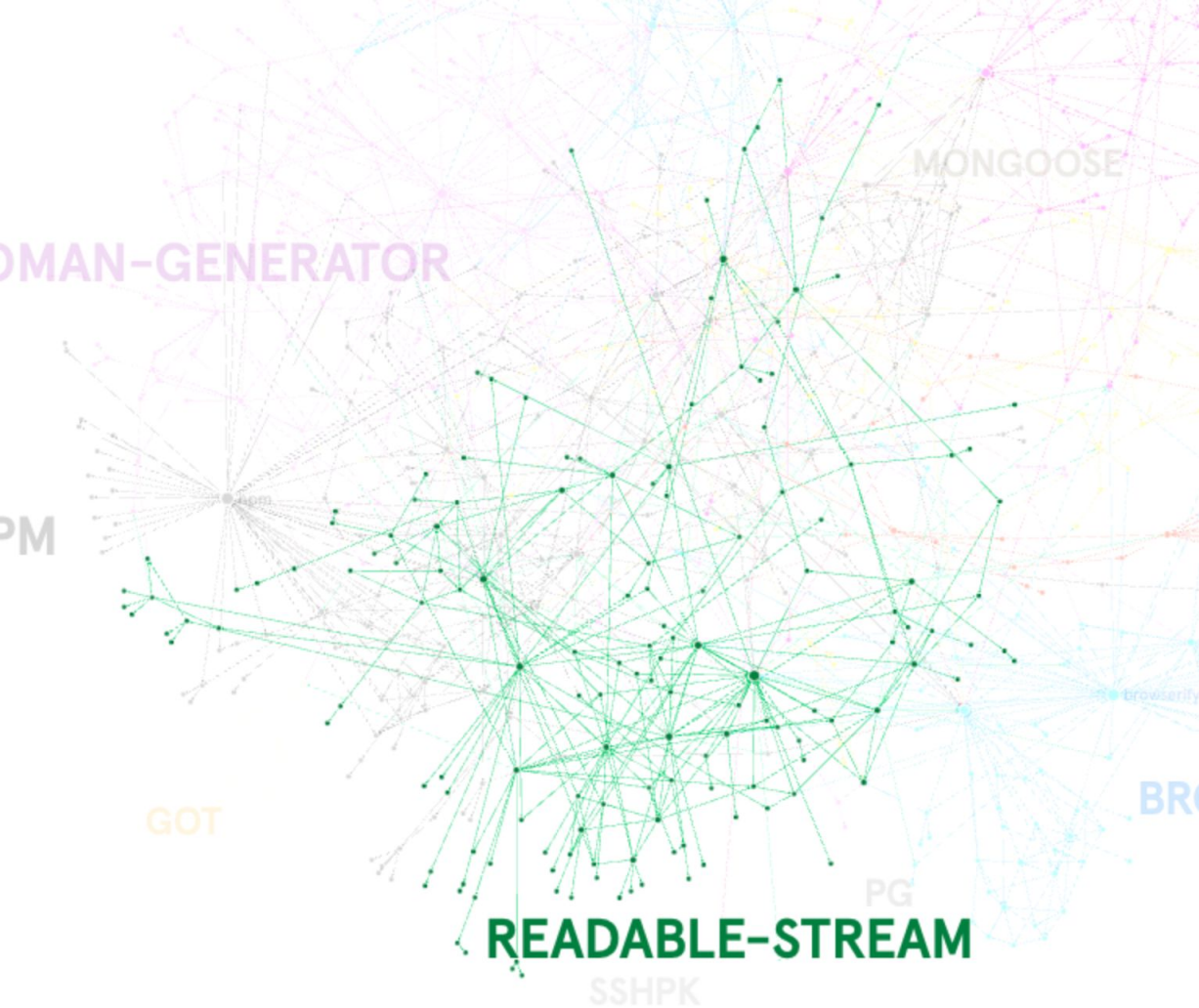




**Top 100 Projects having
4 level of transitive
dependencies.**

<https://medium.com/graph-commons/analyzing-the-npm-dependency-network-e2cf318c1d0d>

<https://legacy.graphcommons.com/graphs/a7ec343d-2a0c-47bb-9658-bb8315e8a096?show=analysis-cluster>



Graph of Readable-stream package
with around 144 nodes (containing 4
level deep dependencies)

Dependency Hell

OKAY, LET'S TALK STRATEGIES



9 Circles of Dependency Hell

- Problems with Package Management
 - Are my dependencies even correct?
 - Updating a new package and breaking something else.
 - Bloated bundles. Too many dependencies.
 - Multiple package managers.
 - The package or version you need isn't in your package manager.
 - Monkey patching a dependency.
 - Breaking changes on a minor or patch version.
 - Circular dependencies.
 - The diamond dependency problem.

<https://about.sourcegraph.com/blog/nine-circles-of-dependency-hell>

9 Circles of Dependency Hell //

Are my dependencies even correct?

Mismatched Manifests
Restrictive Licenses

9 Circles of Dependency Hell //

Are my dependencies even correct?

Set-up a proper dependency vetting process.

Dependency manifest(s) as single source of truth to avoid inconsistency.

Are my dependencies even correct?

- One should vet the dependency before using upon multiple factors -
 - Number of maintainers
 - Number of issues
 - Number of downloads
 - Longest open issue
 - Discussion on the issues, etc.
- Setup processes to identify the drift between the packages installed and the packages that are mentioned in the manifest.

<https://github.com/safedep/vet.git>

<https://www.mariokandut.com/how-to-check-unused-npm-packages/>

9 Circles of Dependency Hell //

Updating a new package and breaking something else.

No dependency vetting

3rd party author's trustworthiness

<https://evertpot.com/npm-revoke-breaks-the-build/>

<https://qz.com/646467/how-one-programmer-broke-the-internet-by-deleting-a-tiny-piece-of-code>

<https://www.bleepingcomputer.com/news/security/dev-corrupts-npm-libraries-colors-and-faker-breaking-thousands-of-apps/>

9 Circles of Dependency Hell //

Updating a new package and breaking something else.

Can't emphasize enough, setup a proper vetting process.

Keeping a local cached copy of the dependencies used (if you are big enough to maintain)

9 Circles of Dependency Hell //

Bloated bundles. Too many dependencies.

Slow builds

Older/unused dependencies in the environment

<https://www.darkreading.com/vulnerabilities-threats/on-shaky-ground-why-dependencies-will-be-your-downfall>

<https://bundlephobia.com/>

<https://www.bitovi.com/blog/why-your-angular-bundle-is-bloated>

9 Circles of Dependency Hell //

Bloated bundles. Too many dependencies.

Inventory & audit the dependencies.

```
npm list --depth 100  
npm audit
```

9 Circles of Dependency Hell //

Multiple package managers.

Slow builds

Package conflicts

9 Circles of Dependency Hell //

Multiple package managers.

Ideal scenario is to use one package manager per language.

9 Circles of Dependency Hell //

The package or version you need isn't in your package manager.

9 Circles of Dependency Hell //

The package or version you need isn't in your package manager.

Use Git Repositories to install packages locally.

● a technique used to dynamically update the behavior of a piece of code at run-time without altering the original source code.

9 Circles of Dependency Hell //

Monkey patching a dependency.

Difficult to upgrade
Malicious Monkey Patch

<https://arstechnica.com/information-technology/2009/05/mozilla-ponders-policy-change-after-firefox-extension-battle/>

9 Circles of Dependency Hell //

Monkey patching a dependency.

Again in an ideal world you should not monkey patch, but if you must then properly store and document.

9 Circles of Dependency Hell //

Breaking changes on a minor or patch version.

Patches

Non Semantic Versioning

<https://pypi.org/project/html5lib/#history>



0.999999999

Jul 15, 2016



0.99999999

Jul 15, 2016



0.9999999

Sep 10, 2015



0.999999

Jul 7, 2015



0.99999

May 1, 2015

9 Circles of Dependency Hell //

Breaking changes on a minor or patch version.

Keep the semantic versioning and follow the started ruleset
to define a version - MAJOR.MINOR.PATCH

<https://pypi.org/project/requests/#history>



9 Circles of Dependency Hell //

Circular Dependencies.

$A \rightarrow B \rightarrow A$

Unintended consequences.

<https://medium.com/@louismrc/fix-your-circular-dependencies-with-dependency-inversion-e22b6f4c9510>

<https://discuss.python.org/t/handling-installation-of-circular-dependencies/25531/8>

<https://spin.atomicobject.com/2018/06/25/circular-dependencies-javascript/>

9 Circles of Dependency Hell //

Circular Dependencies.

Identify & Avoid using circular dependencies.

Use different design pattern while working on the project.

<https://www.npmjs.com/package/madge>

9 Circles of Dependency Hell //

The diamond dependency problem.

Different version of same package

<https://docs.copado.com/articles/#!/copado-methodology-temp/the-diamond-dependency-problem>
<https://well-typed.com/blog/2008/08/solving-the-diamond-dependency-problem/>

9 Circles of Dependency Hell //

The diamond dependency problem.

Performs the deduplication of the dependency.

<https://docs.copado.com/articles/#!/copado-methodology-temp/the-diamond-dependency-problem>
<https://well-typed.com/blog/2008/08/solving-the-diamond-dependency-problem/>

Security Product

- Compliment each other.
- Reduce load from product teams by validating the issues.
- Understand the technical impact by reachability analysis and then propose the solution.

Closing Pointers

- Minimize dependencies.
- Standardize the package manager.
- Follow semantic versioning.
- Vet the dependency properly.
 - Check for the associated security issues.
 - Backward compatibility and lock file.
 - You Ain't Gonna Need It (YAGNI) Principle
 - Licensing & Legal Considerations
 - Duplicated functionality
 - Popularity
- Check for the unused or too complex dependencies.

Thank you!
Embrace the chaos!

Kumar Ashwin
[Oxcardinal.com](https://oxcardinal.com)