

Software Dependencies

17-313 Fall 2024

Foundations of Software Engineering

<https://cmu-313.github.io>

Michael Hilton and Rohan Padhye

Administrivia

- P4 final due Friday night
- P5 released (select projects by Tue, Nov 19th; checkpoint slides due Sun, Nov 25th—presentation next day)
- Midterm 2 next week (Thu, Nov 21st)

Left-pad (March 22, 2016)

OBSESSIONS

QUARTZ

NPM ERR!

How one programmer broke the internet by deleting a tiny piece of code

THE VERGE

TECH

REVIEWS

REPORT TECH

How an irate developer briefly broke JavaScript

Unpublishing 11 lines of code brought down an open source house of cards

By Paul Miller | @futurepaul | Mar 24, 2016, 4:29pm EDT



SIGN IN

The Register

{* SOFTWARE *}

How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript

Code pulled from NPM – which everyone was using

Left-pad (March 22, 2016)

npmjs.org tells me that left-pad is not available (404 page) #4

 Closed silkenrance opened this issue on Mar 22, 2016 · 193 comments



silkenrance commented on Mar 22, 2016

When building projects on travis, or when searching for left-pad on npmjs.com, both will report that the package cannot be found.

Here is an excerpt from the travis build log

```
npm ERR! Linux 3.13.0-40-generic
npm ERR! argv "/home/travis/.nvm/versions/node/v4.2.2/bin/node" "/home/travis/.nvm/versions/node/v4.2.2/bin/npm
npm ERR! node v4.2.2
npm ERR! npm v2.14.7
npm ERR! code E404
npm ERR! 404 Registry returned 404 for GET on https://registry.npmjs.org/left-pad
npm ERR! 404
npm ERR! 404 'left-pad' is not in the npm registry.
npm ERR! 404 You should bug the author to publish it (or use the name yourself!)
npm ERR! 404 It was specified as a dependency of 'line-numbers'
npm ERR! 404
npm ERR! 404 Note that you can also install from a
npm ERR! 404 tarball, folder, http url, or git url.
npm ERR! Please include the following file with any support request:
npm ERR!   /home/travis/build/coldrye-es/pingo/npm-debug.log
make: *** [deps] Error 1
```

And here is the standard npmjs.com error page <https://www.npmjs.com/package/left-pad>

However, if I remove left-pad from my local npm cache and then reinstall it using npm it will happily install left-pad@0.0.4.

 88

 3

Left-pad (Docs)

left-pad

String left pad

build unknown

Install

```
$ npm install left-pad
```

Usage

```
const leftPad = require('left-pad')

leftPad('foo', 5)
// => "  foo"

leftPad('foobar', 6)
// => "foobar"

leftPad(1, 2, '0')
// => "01"

leftPad(17, 5, 0)
// => "00017"
```

Install

```
> npm i left-pad
```

Repository

github.com/stevemao/left-pad

Homepage

github.com/stevemao/left-pad#readme

Weekly Downloads

2,962,641



Version

1.3.0

License

WTFPL

Unpacked Size

9.75 kB

Total Files

10

Issues

3

Pull Requests

7

Last publish

4 years ago

Left-pad (Source Code)

17 lines (11 sloc) | 222 Bytes

```
1  module.exports = leftpad;
2
3  function leftpad (str, len, ch) {
4    str = String(str);
5
6    var i = -1;
7
8    if (!ch && ch !== 0) ch = ' ';
9
10   len = len - str.length;
11
12   while (++i < len) {
13     str = ch + str;
14   }
15
16   return str;
17 }
```

See also: isArray

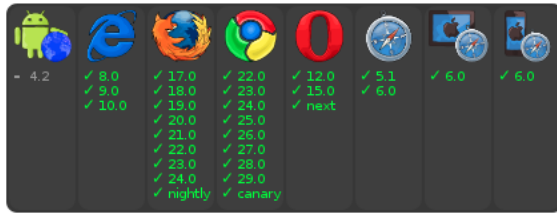
5 lines (4 sloc) | 133 Bytes

```
1 var toString = {}.toString;
2
3 module.exports = Array.isArray || function (arr) {
4   return toString.call(arr) === '[object Array]';
5 };
```

isarray

Array#isArray for older browsers and deprecated Node.js versions.

build passing downloads 227M/month



Just use `Array.isArray` directly, unless you need to support those older versions.

Usage

```
var isArray = require('isarray');

console.log(isArray([])); // => true
console.log(isArray({})); // => false
```

Install

```
> npm i isarray
```

Repository

github.com/juliangruber/isarray

Homepage

github.com/juliangruber/isarray

Weekly Downloads

50,913,317

Version License

2.0.5 MIT

Unpacked Size Total Files

3.43 kB 4

Issues Pull Requests

4 3

How do software projects manage third-party dependencies on reusable libraries?

- It's hard
- It's mostly a mess (everywhere)
- But it's critical to modern software development

What is a Dependency?

- Core of what most build systems do
 - “Compile” and “Run Tests” is just a fraction of their job
- Examples: Maven, Gradle, NPM, Bazel, ...
- **Foo->Bar**: To build Foo, you may need to have a built version of Bar
- Dependency Scopes:
 - **Compile**: Foo uses classes, functions, etc. defined by Bar
 - **Runtime**: Foo uses an abstract API whose implementation is provided by Bar (e.g. logging, database, network or other I/O)
 - **Test**: Foo needs Bar only for tests (e.g. JUnit, mocks)
- Internal vs. External Dependencies
 - Is Bar also built/maintained by your org or is it pulled from elsewhere using a package manager?

Examples of dependency views

NodeBB / install / package.json

Code Blame 201 lines (201 loc) · 6.13 KB

```
44 "autoprefixer": "10.4.20",
45 "bcryptjs": "2.4.3",
46 "benchpressjs": "2.5.1",
47 "body-parser": "1.20.3",
48 "bootbox": "6.0.0",
49 "bootstrap": "5.3.3",
50 "bootswatch": "5.3.3",
51 "chalk": "4.1.2",
52 "chart.js": "4.4.4",
53 "cli-graph": "3.2.2",
54 "clipboard": "2.0.11",
55 "colors": "1.4.0",
56 "commander": "12.1.0",
57 "compare-versions": "6.1.1",
58 "compression": "1.7.4",
59 "connect-flash": "0.1.1",
60 "connect-mongo": "5.1.0",
61 "connect-multiparty": "2.2.0",
62 "connect-pg-simple": "10.0.0",
63 "connect-redis": "7.1.1",
64 "cookie-parser": "1.4.6",
65 "cron": "3.1.7",
```

```
github.com/CMU-313/Teedy/blob/main/pom.xml
152 <dependencyManagement>
153 <dependencies>
154 <dependency>
155 <groupId>com.sismics.docs</groupId>
156 <artifactId>docs-core</artifactId>
157 <version>${project.version}</version>
158 </dependency>
159
160 <dependency>
161 <groupId>com.sismics.docs</groupId>
162 <artifactId>docs-web-common</artifactId>
163 <version>${project.version}</version>
164 </dependency>
165
166 <dependency>
167 <groupId>com.sismics.docs</groupId>
168 <artifactId>docs-web-common</artifactId>
169 <type>test-jar</type>
170 <version>${project.version}</version>
171 </dependency>
172
173 <dependency>
174 <groupId>com.sismics.docs</groupId>
175 <artifactId>docs-web</artifactId>
176 <version>${project.version}</version>
177 </dependency>
178
179 <dependency>
180 <groupId>org.eclipse.jetty</groupId>
181 <artifactId>jetty-server</artifactId>
182 <version>${org.eclipse.jetty.jetty-server.version}</version>
183 </dependency>
184
185 <dependency>
186 <groupId>org.eclipse.jetty</groupId>
187 <artifactId>jetty-webapp</artifactId>
188 <version>${org.eclipse.jetty.jetty-webapp.version}</version>
189 </dependency>
```

Package: git (1:2.17.1-1ubuntu0.9)

fast, scalable, distributed revision control system

Other Packages Related to git

- depends recommends suggests enhances
- git-man (<< 1:2.17.0-) [not amd64, i386]
fast, scalable, distributed revision control system (manual pages)
- git-man (<< 1:2.17.1-) [amd64, i386]
- git-man (>> 1:2.17.0) [not amd64, i386]
- git-man (>> 1:2.17.1) [amd64, i386]
- libc6 (>= 2.16) [not arm64, ppc64el]
GNU C Library: Shared libraries
also a virtual package provided by libc6-udeb
- libc6 (>= 2.17) [arm64, ppc64el]
- libcurl3-gnutls (>= 7.16.2)
easy-to-use client-side URL transfer library (GnuTLS flavour)
- liberror-perl
Perl module for error/exception handling in an OO-ish way
- libexpat1 (>= 2.0.1)
XML parsing C library - runtime library
- libpcre3
Old Perl 5 Compatible Regular Expression Library - runtime files
- perl
Larry Wall's Practical Extraction and Report Language
- zlib1g (>= 1:1.2.0)
compression library - runtime
- less
pager program similar to more
- patch
Apply a diff file to an original
- ssh-client
virtual package provided by openssh-client

Where are the dependencies hosted?

- Typically downloaded from dependency servers:
 - Maven Central (<https://repo.maven.apache.org/maven2/>)
 - Ubuntu Packages for Apt (<https://packages.ubuntu.com/>)
 - Python Package Index (<https://pypi.org/>)]
 - NPM Public Registry (<https://registry.npmjs.org/>)
- Packages need a unique identifier
 - Typically a package name (sometimes owner name) and version
- Custom repositories allowed by most package managers
 - Often used for company-internal packages or cache mirroring
 - Note problems with duplicates (same pkg name in different repositories; some priority order is needed)
- Somebody needs to manage repositories
 - Availability: Repository needs to be running
 - Access Control: Packages should only be published by owners
 - Integrity: Packages should be signed or otherwise verifiable
 - Uniqueness and archival: Only one artifact per version
 - Traceability: Packages can have metadata pointing to source or tests
 - Security: ???



Demo: Deps.dev

[open](#) / [source](#) / [insights](#)

npm package

express

5.0.0

Overview

Dependencies

Dependents

Compare

Versions

Filter dependencies by name, license, security advisory and more

Package

Notes

accepts
2.0.0

body-parser
2.0.2

content-disposition
1.0.0

content-type
1.0.5

cookie
0.6.0

1 ADVISORY

cookie-signature
1.2.2



master

express / package.json

Code

Blame

101 lines (101 loc) · 2.69 KB

```
32     "api"
33   ],
34   "dependencies": {
35     "accepts": "^2.0.0",
36     "body-parser": "^2.0.1",
37     "content-disposition": "^1.0.0",
38     "content-type": "~1.0.4",
39     "cookie": "0.7.1",
40     "cookie-signature": "^1.2.1",
41     "debug": "4.3.6",
42     "depd": "2.0.0",
43     "encodeurl": "~2.0.0",
44     "escape-html": "~1.0.3",
45     "etag": "~1.8.1",
46     "finalhandler": "^2.0.0",
47     "fresh": "2.0.0",
48     "http-errors": "2.0.0",
49     "merge-descriptors": "^2.0.0",
50     "methods": "~1.1.2",
51     "mime-types": "^3.0.0",
52     "on-finished": "2.4.1",
53     "once": "1.4.0",
54     "parseurl": "~1.3.3",
55     "proxy-addr": "~2.0.7",
```

Dependency Pinning vs. Floating

- Pinning: "I depend on libFoo 1.5.0"
 - Declares a specific version of the dependency. Frozen in time.
- Floating: "I depend on libFoo-latest"
 - Each build will pull the latest available libFoo version
 - (Other forms available, e.g. libFoo 1.5.x)
- Activity (groups of 2-3; write names and Andrew ID)
 - 1 advantage of pinning over floating
 - 1 advantage of floating over pinning

Pinned dependencies requires manual updates in case of security issues

The image displays three screenshots of the npm website for the 'express' package, illustrating how pinned dependencies affect security updates.

Left Screenshot (express 5.0.0): Shows the 'Dependencies' tab. The 'cookie' dependency is listed as version 0.6.0. A callout bubble points to it with the text 'cookie 0.6.0'. A red badge indicates '1 ADVISORY'.

Middle Screenshot (Security Advisories): Shows a security advisory for 'cookie accepts cookie name, path, and domain with out of bounds characters' (LOW severity, GHSA-pxg6-pf52-xh8x).

Right Screenshot (express 5.0.1): Shows the 'Dependencies' tab. The 'cookie' dependency is updated to version 0.7.1. A callout bubble points to it with the text 'cookie 0.7.1'.

Is Pinning Sinning?

Pinning Dependencies (e.g. 1.5.3)

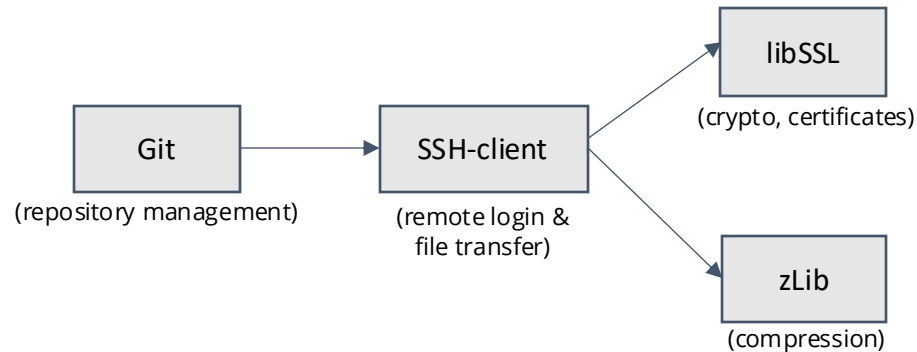
- ✓ Reproducible builds
- ✗ Can become vulnerable due to dependency bugs
- ✗ Have to keep updating dependents as dependencies evolve
- ✓ Stable network effects

Floating Dependencies (e.g. 1.x)

- ✗ Flaky builds (breaking changes)
- ✓ Latest security patches & bug fixes
- ✓ Less manual maintenance
- ✗ Floats leak transitively
(A pin to B floating C; then A still sees changing version of C)

Transitive Dependencies

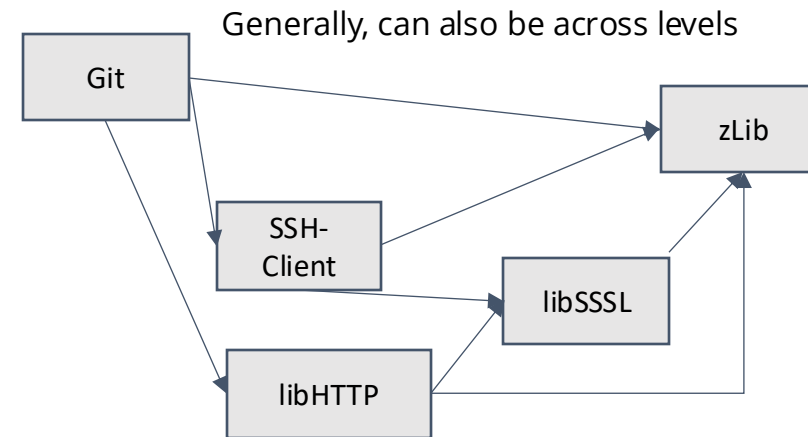
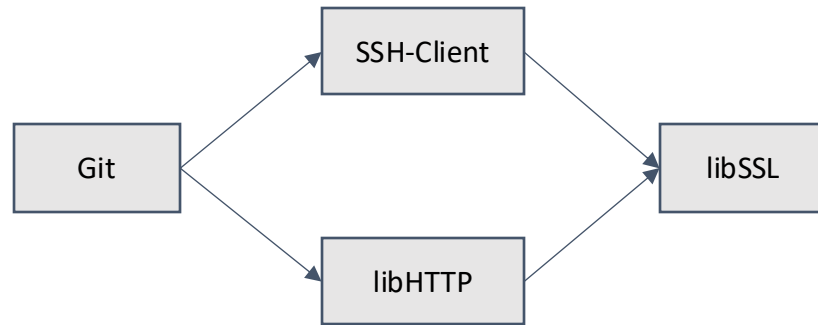
Packages can depend on other packages



Q: Should Git be able to use exports of libSSL (e.g. certificate management) or zLib (e.g. gzip compression)?

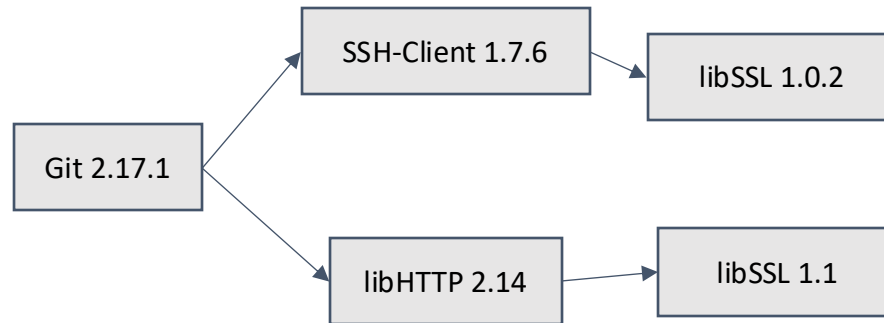
Diamond Dependencies

What are some problems when multiple intermediate dependencies have the same transitive dependency?



Diamond Dependencies

What are some problems when multiple intermediate dependencies have the same transitive dependency?



Resolutions to the Diamond Problem

1. Duplicate it!
 - Doesn't work with static linking (e.g. C/C++), but may be doable with Java (e.g. using ClassLoader hacking or package renaming)
 - Values of types defined by duplicated libraries cannot be exchanged across
2. Ban transitive dependencies; just use a global list with one version for each
 - Challenge: Keeping things in sync with latest
 - Challenge: Deciding which version of transitive deps to keep
3. Newest version (keep everything at latest)
 - Requires ordering semantics
 - Intermediate dependency may break with update to transitive
4. Oldest version (lowest denominator)
 - Also requires ordering semantics
 - Sacrifices new functionality
5. Oldest non-breaking version / Newest non-breaking version
 - Requires faith in tests or semantic versioning contract

Semantic Versioning

- Widely used convention for versioning releases
 - E.g. 1.2.1, 3.1.0-alpha-1, 3.1.0-alpha-2, 3.1.0-beta-1, 3.1.0-rc1
- Format: {MAJOR} . {MINOR} . {PATCH}
- Each component is ordered (numerically, then lexicographically; release-aware)
 - 1.2.1 < 1.10.1
 - 3.1.0-alpha-1 < 3.1.0-alpha-2 < 3.1.0-beta-1 < 3.1.0-rc1 < 3.1.0
- Contracts:
 - MAJOR updated to indicate breaking changes
 - Same MAJOR version => backward compatibility
 - MINOR updated for additive changes
 - Same MINOR version => API compatibility (important for linking)
 - PATCH updates functionality without new API
 - Ninja edit; usually for bug fixes
- Largely dependent on honor system. No easy way to automatically verify (can you solve it?)

<https://semver.org/>

[2.0.0](#) [2.0.0-rc.2](#) [2.0.0-rc.1](#) [1.0.0](#) [1.0.0-beta](#)

Semantic Versioning 2.0.0

Summary

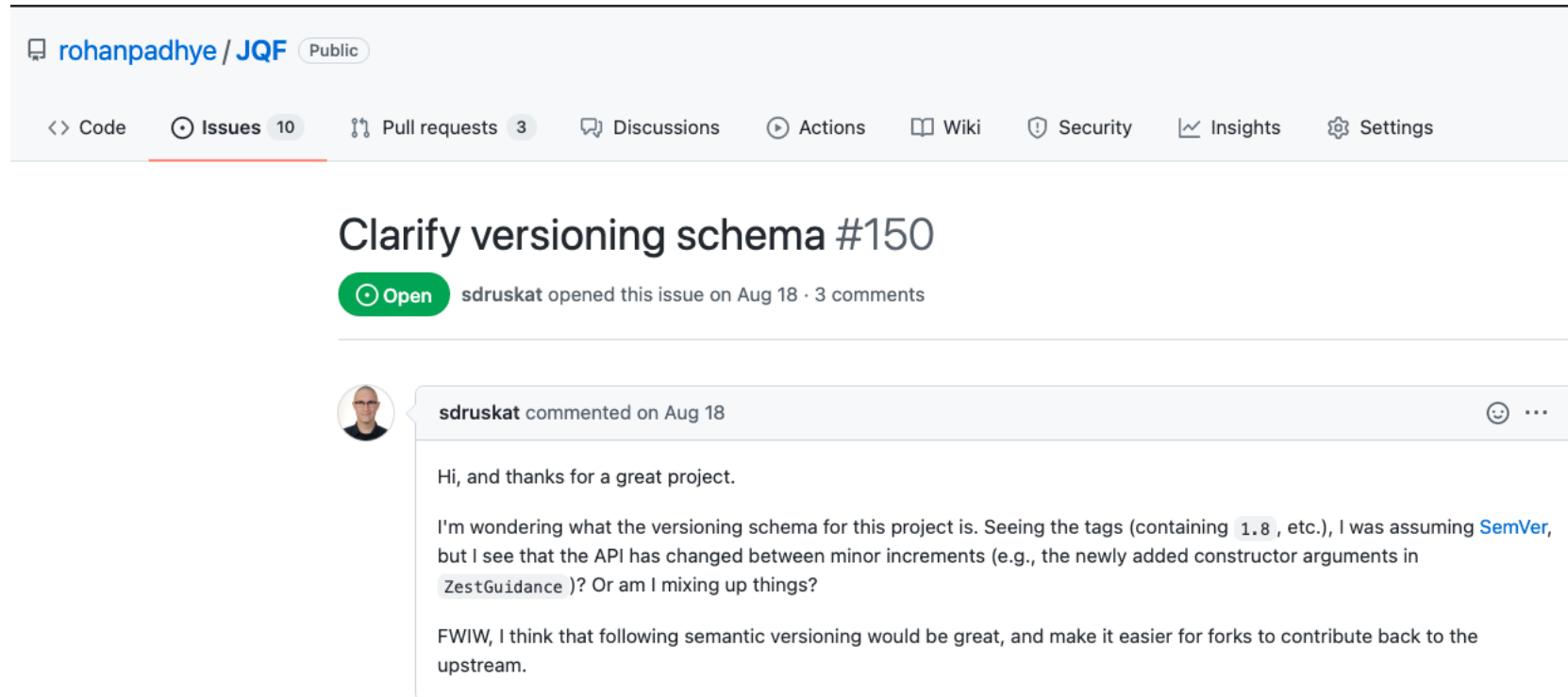
Given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make incompatible API changes,
2. MINOR version when you add functionality in a backwards compatible manner, and
3. PATCH version when you make backwards compatible bug fixes.

Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

People rely on SemVer contracts

(I got this "bug report" on one of my own research projects)



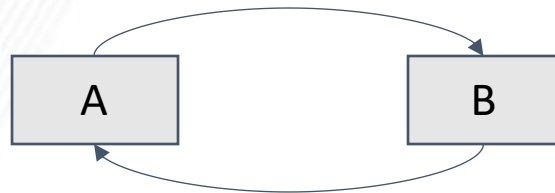
The screenshot shows a GitHub repository page for 'rohanpadhye / JQF' with a 'Public' label. The navigation bar includes 'Code', 'Issues 10', 'Pull requests 3', 'Discussions', 'Actions', 'Wiki', 'Security', 'Insights', and 'Settings'. The 'Issues' tab is selected. The issue title is 'Clarify versioning schema #150', opened by 'sdruskat' on Aug 18 with 3 comments. A comment from 'sdruskat' is shown, dated Aug 18, with a smiley face icon and a three-dot menu. The comment text reads: 'Hi, and thanks for a great project. I'm wondering what the versioning schema for this project is. Seeing the tags (containing 1.8, etc.), I was assuming SemVer, but I see that the API has changed between minor increments (e.g., the newly added constructor arguments in ZestGuidance)? Or am I mixing up things? FWIW, I think that following semantic versioning would be great, and make it easier for forks to contribute back to the upstream.'

Dependency Constraints

- E.g. Declare dependency on "Bar > 2.1"
 - Bar 2.1.0, 2.1.1, 2.2.0, 2.9.0, etc. all match
 - 2.0.x does NOT match
 - 3.0.x does NOT match
- Diamond dependency problem can be resolved using SAT solvers
 - E.g. Foo 1.0.0 depends on "Bar >= 2.1" and "Baz 1.8.x"
 - Bar 2.1.0 depends on "Qux [1.6, 1.7]"
 - Bar 2.1.1 depends on "Qux 1.7.0"
 - Baz 1.8.0 depends on "Qux 1.5.x"
 - Baz 1.8.1 depends on "Qux 1.6.x"
 - Find an assignment such that all dependencies are satisfied
 - Solution: Use Bar 2.1.0, Baz 1.8.1, and Qux 1.6.{latest}

Cyclic Dependencies

- A very bad thing
- Avoid at all costs
- Sometimes unavoidable or intentional
 - E.g. GCC is written in C (needs a C compiler)
 - E.g. Apache Maven uses the Maven build system
 - E.g. JDK tested using JUnit, which requires the JDK to compile



Cyclic Dependencies

- Bootstrapping: Break cycles over time
- Assume older version exists in binary (pre-built form)
- Step 1: Build A using an older version of B
- Step 2: Build B using new (just built) version of A
- Step 3: Rebuild A using new (just built) version of B
- Now, both A and B have been built with new versions of their dependencies
- Doesn't work if both A and B need new features of each other at the same time (otherwise Step 1 won't work)
 - Assumes incremental dependence on new features
- How was the old version built in the first place? (turtles all the way down)
 - Assumption: cycles did not exist in the past
 - Successfully applied in compilers (e.g. GCC is written in C)

Dependency Security

- Will you let strangers execute arbitrary code on your laptop?
 - Think about this every time you do “pip install” or “npm install” or “apt-get upgrade” or “brew upgrade” or whatever (esp. with sudo)
 - Scary, right? Who are you trusting? Why?
- Typo squatting (“pip install numpi”)
- Outright malice (search for the **xz-utils backdoor** incident)
- Genuine security vulnerabilities due to software bugs (e.g. **log4j**)

The XZ Backdoor: Everything You Need to Know

Details are starting to emerge about a stunning supply chain attack that sent the open source software community reeling.



<https://www.wired.com/story/xz-backdoor-everything-you-need-to-know/>

Dependabot alerts / #74

Deserialization of Untrusted Data in Apache Log4j #74

[Open](#) Opened 3 days ago on log4j:log4j (Maven) - pom.xml

Package	Affected versions	Patched version
log4j:log4j (Maven)	<= 1.2.17	None

CVE-2020-9493 Identified a deserialization issue that was present in Apache Chainsaw. Prior to Chainsaw V2.0 Chainsaw was a component of Apache Log4j 1.2.x where the same issue exists.

Users are advised to migrate from `log4j:log4j` to `org.apache.logging.log4j:log4j` for an updated version of the library.

[dependabot](#) bot opened this 3 days ago

Severity
Critical 9.8 / 10

CVSS base metrics	
Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	High

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

Weaknesses

CWE-502

Takeaways

- Dependency management is hard.