Dieses Dokument ist lizenziert für HS Augsburg, uw00411t.
Alle Rechte vorbehalten. © c't - magazin für computertechnik (CT). Download vom 31.05.2023 08:02 von www.wiso-net.de.

Die Abhängigkeitshölle

Wie Abhängigkeiten Software gefährden und was Sie dagegen tun sollten

Bild: Albert Hulm

Dieses Dokument ist lizenziert für HS Augsburg, uw00411t.
Alle Rechte vorbehalten. © c't - magazin für computertechnik (CT). Download vom 31.05.2023 08:02 von www.wiso-net.de.

In modernen Applikationen fußt Software Schicht um Schicht auf anderer Software. Muss man sich das wie ein fest gefügtes Mauerwerk oder mehr wie einen wackeligen Stapel vorstellen, der jeden Moment die ganze Firma begraben kann? Wer das nicht weiß, lebt gefährlich! Gegen das Unwissen helfen Werkzeuge, um die eigenen Abhängigkeiten aufzudecken und ihre Zuverlässigkeit zu beurteilen.

Von Sylvester Tremmel

Sieben Terrassen hat der Läuterungsberg in Dantes Göttlicher Komödie. Auf Terrasse Nummer vier des Fegefeuers büßen die Faulen in immerwährender Hetze. Irgendwo dort werden sich vielleicht auch die Programmierer einfinden müssen, die gedankenlos Abhängigkeiten anhäufen, um zwei oder drei Zeilen Code zu sparen – wenn sie Glück haben.

Wenn Sie Pech haben, holen Ihre Sünden sie noch in diesem Leben ein. So dürften sich im Dezember 2021 viele IT-Mitarbeiter gefühlt haben, als sie von Abteilung zu Abteilung hetzten. Es galt zu verhindern, dass ihr Betrieb einer Sicherheitslücke in Log4j zum Opfer fällt – vor allem aber mussten sie erst mal herausfinden, an wie vielen Stellen man die Java-Bibliothek Log4j überhaupt nutzte. „Schnell, schnell, daß nicht die Zeit verlorengehe“, hätten sie zusammen mit Dantes Büßern schreien mögen.

Daraus darf man aber nicht den Schluss ziehen, Open Source zu verteufeln oder ganz auf Bibliotheken zu verzichten. Die Zeiten, in denen man jedes Projekt als frei stehenden Haufen Assembler- oder C-Code implementieren konnte, sind vorbei: Anwendungen sollen im Browser laufen oder auf virtuellen Maschinen, die Bytecode interpretieren, und sie sollen möglichst architektur- und betriebssystemübergreifend funktionieren.

Zum anderen wird aus der krampfhaften Vermeidung von Abhängigkeiten allzu leicht das „Not invented here Syndrom“. Dabei erfindet jeder Entwickler jedes Rad neu – noch dazu oft mit mehr Ecken, als ein Rad haben sollte. Das ist nicht nur ineffizient, sondern auch ineffektiv: Das globale Software-Ökosystem blüht und gedeiht genau deshalb so prächtig, weil man auf Vorhandenes aufbaut, kooperiert und wechselseitig die Arbeit anderer nutzt. Ganz besonders gilt das für Open-Source-Projekte.

Es ist also gut und richtig, dass Log4j in Unmengen von Java-Projekten steckt: Fast jede Software sollte Log-Dateien schreiben und wer auf eine vorhandene Bibliothek setzt, der profitiert, wenn andere die Bibliothek verbessern. Weniger gut und richtig ist, wie unkoordiniert und versteckt die Abhängigkeiten oft wuchern und wie gedankenlos sie in Projekte geladen werden.

Wucherungen

Dieser Wildwuchs nimmt mitunter absurde Ausmaße an, man könnte vom „Already invented there Syndrom“ sprechen: NPM ist ein Katalog für JavaScript-Pakete, nach eigener Aussage das weltgrößte Softwareverzeichnis – und ein Schauplatz solcher pathologischen Abhängigkeiten. Immer wieder fallen auch große Projekte auf die Nase, weil sie sich indirekt und blind auf irgendein kleines Trivialpaket verlassen – und dessen Entwickler ihr Werk depublizieren, sabotieren, einen Fehler machen oder einfach nicht mehr pflegen.

Im März 2016 traf es zum Beispiel den JavaScript-Transpiler Babel, der modernen JavaScript-Code in älteren Code für ältere Browser übersetzt. Babel nutzte das Paket left-pad, dessen frustrierter Entwickler es im Zuge eines Namensstreits depublizierte. Das brachte Babel ins Straucheln, ebenso wie viele andere Projekte, die left-pad nutzten, zahllose weitere Projekte, die Babel nutzten, und so weiter.

left-pad ist keine hilfreiche Bibliothek, die komplizierte Arbeit abnimmt, sondern besteht aus einer Handvoll Zeilen Code. Damit kann man Zeichenketten auf der linken Seite bis zu einer bestimmten Länge auffüllen. Diese Trivialität war eine Abhängigkeit, auf die sich alle möglichen Projekte blind verließen.

Dergleichen passiert immer wieder, etwa im April 2020, aufgrund eines Problems mit dem Paket is-promise, das im Grunde aus einer einzigen Zeile Code besteht. NPM ist voll von derartigen Skurrilitäten: Das Paket is-even prüft, ob eine Zahl gerade ist. Dafür nutzt es die Abhängigkeit is-odd, die auf das Gegenteil prüft. Von beiden Paketen hängen Dutzende andere Pakete ab – und is-odd benötigt das Paket is-number.

Auf Seite 64 lesen Sie, worauf man stößt, wenn man die Abhängigkeiten des Frontend-Frameworks Vue.js beforscht: ein wichtiges Projekt, das zahlreiche Größen der IT-Branche benutzen. Hinter Vue.js selbst steht aber keine solche Größe und mit dem Projekt alleine ist es auch nicht getan. Denn Vue benötigt zum Beispiel das erwähnte Kompatibilitätstool Babel. Und was ist mit den Projekten, auf die wiederum Babel aufbaut? Hinter jeder neuen Abhängigkeit kann sich eine Perle der Open-Source-Kultur, der Morast von verwahrlostem Code oder der Wahnwitz sinnloser Minimalpakete verbergen.

So interessant diese Überraschungen für NPM-Forschungsreisende sind: In eigenem Code will man sie nicht haben. Auf Seite 68 zeigen wir Ihnen deshalb, wie SBOMs helfen, den Überblick zu bewahren. In so einer „Software Bill of Materials“ finden sich alle Abhängigkeiten fein säuberlich aufgelistet. Mit den richtigen Tools wissen Sie dann immer, auf welchen Säulen ihre Systeme ruhen – und welche dieser Säulen eventuell bröckeln.

Wenn dann eine Sicherheitslücke entdeckt wird oder ein Projekt den Geist aufgibt, versinken Sie nicht in hektischem Chaos, sondern wissen gleich, welche Bereiche betroffen sind und was Sie beruhigt ignorieren können. Schnell reagieren müssen Sie im Fall der Fälle immer noch, aber Sie haben einen Plan und büßen weder im Diesseits noch im Jenseits für Ihre Trägheit. *(syt@ct.de)* c't