Publicatie : AG Connect
Datum : 30 sep 2020
cm2 : 1655
Pagina : 44
AVE : € 21000,00
Regio : Nederland
Frequentie : 10x per jaar
Oplage : 8000

Hoe ga je verstandig om met componenten van derde partijen in je code?

'JE WILT NIET AFHANKELIJK ZIJN VAN EEN DOOD PAARD'

Vrijwel alle ontwikkelaars gebruiken dependencies in hun code: een component dat je in je eigen project gebruikt en waar je afhankelijk van bent. Maar deze gewoonte kan ook problemen met zich meebrengen. Al die componenten moeten onderhouden en geüpdatet worden. Doe je dat niet, dan kunnen er serieuze problemen ontstaan.

door Eveline Meijer beeld Shutterstock

DAT HET GEBRUIK VAN DEPENDENCIES ALOMTEGENWOORDIG IS, BLIJKT WEL UIT HET 2020 OPEN SOURCE SECURITY AND RISK ANALYSIS REPORT VAN SYNOPSYS. Volgens dit rapport bevat 99% van alle commerciële software opensourcecomponenten, wat dus dependencies zijn.

Maar waarom zou je een component van een ander willen gebruiken in je eigen project? "Het is heel simpel. Wat een ander al gebouwd heeft, hoef je niet zelf te bouwen", vertelt Tijs van der Storm, senior researcher bij het Centrum Wiskunde & Informatica (CWI) en professor software engineering bij de Rijksuniversiteit Groningen. "Als een ander iets al gemaakt heeft, is het goedkoper om dat te gebruiken dan om het zelf te maken. Je bespaart daarmee niet alleen op het intypen van de code, maar ook op kennis. Je hergebruikt namelijk developmentkennis."

In het geval van opensourcecomponenten is het grootste deel ook nog eens gratis. De groei van open source in de afgelopen jaren draagt dan ook bij aan het gebruik van dependencies, stelt Tibor Lapikas, proposition manager bij

Publicatie : AG Connect
Datum : 30 sep 2020
cm2 : 1655
Pagina : 44
AVE : € 21000,00
Regio : Nederland
Frequentie : 10x per jaar
Oplage : 8000

70% VAN DE COMMERCIËLE SOFTWARE BLIJKT VEROUDERDE ONDERDELEN TE BEVATTEN

Software Improvement Group (SIG). "In de afgelopen tien jaar is er steeds meer goede open source op de markt gekomen, die heel mooie functionaliteiten biedt. Het bespaart je veel tijd als je die code gebruikt."

HET GAAT VAAK MIS

Maar het hergebruik van dit soort componenten komt ook met risico's. Zo blijkt 70% van de commerciële software onderdelen te bevatten die verouderd waren. Bijna de helft daarvan bevat beveiligingsproblemen met een 'hoog risico'-status.

Ook code die nog wel wordt onderhouden kan een securityprobleem bevatten. "In de meeste gevallen is het niet zo dat iets niet meer wordt onderhouden, maar dat de ontwikkelaars die de opensource-software gebruiken niet altijd de laatste versie gebruiken. Ze lopen dus niet goed genoeg mee met de updates", verklaart Lapikas. Wordt een patch niet geïnstalleerd, dan blijft een compleet project kwetsbaar door de fout in de dependency. Omdat die broncode openbaar is, kunnen kwaadwillenden op hun gemak naar dit soort gaten zoeken. Overigens worden in closedsource-software ook kwetsbaarheden gevonden, maar op andere manieren.

En dan zijn er ook nog gevallen waarin een component opeens van het internet verdwijnt. Een bekend voorbeeld daarvan is left-pad, een softwarecomponent van elf regels JavaScriptcode dat op de populaire dienst npm werd gepubliceerd. In 2016 ontstond er ruzie tussen de maker van het component en npm zelf, waarop de maker de code verwijderde. Snel daarna bleek dat deze elf regels code ontzettend veel werden gebruikt: het stukje code werd in een maand bijna 2,5 miljoen keer opgehaald. Maar alles wat left-pad gebruikte kreeg nu foutmeldingen, met wereldwijde chaos als gevolg. Pas na twee uur verscheen er een nieuwe versie van de code, waardoor de problemen opgelost werden.

Je kunt je dus snel gaan afvragen waarom ontwikkelaars zichzelf zo afhankelijk maken van iets dat ze zelf niet in beheer hebben. Waarom maken ze die componenten dan niet zelf? Daar is geen beginnen aan, vinden zowel Van der Storm als Lapikas. "Het scheelt echt behoorlijk veel werk. Als je dat allemaal zelf maakt, verdubbel je gemakkelijk je ontwikkeltijd." Daarnaast liggen de problemen niet bij de ontwikkelaar zelf als ergens een lek in blijkt te zitten. Het is niet aan hem om dat gat te dichten, maar aan de mensen die de dependency maken en updaten.

UPDATEN, UPDATEN, UPDATEN

Helaas kun je als ontwikkelaar weinig doen om je te beschermen tegen het echt verdwijnen van dependencies. Maar gelukkig komt dat ook nauwelijks voor.

Publicatie : AG Connect
Datum : 30 sep 2020
cm2 : 1655
Pagina : 44
AVE : € 21000,00
Regio : Nederland
Frequentie : 10x per jaar
Oplage : 8000

Ook code die nog wel wordt onderhouden kan een securityprobleem bevatten

Voor de rest van de problemen is de oplossing vaak hetzelfde als bij iedere andere vorm van software: zorg dat je up-to-date bent. Maar dat is bij dependencies een stuk ingewikkelder dan bij computersystemen.
"Het gaat hier om backwards compatability", legt Van der Storm uit. "Het kan zijn dat iets na een update niet goed meer werkt. Als je code dan niet meer compileert, dan weet je dat het stuk is en dat je hier wat aan moet doen.
Maar het komt ook voor dat de code nog wel compileert, maar niet meer werkt zoals bedoeld. Dat is een veel groter probleem."
Bovendien kan het lastig zijn om bij te houden welke componenten je allemaal gebruikt. Veel projecten gebruiken twintig of wel meer soorten libraries in een omgeving. Daarom is er de laatste jaren tooling ontstaan om hierbij te helpen. Zogeheten packagemanagers als npm en Maven zorgen bijvoorbeeld voor traceerbaarheid, vertelt Van der Storm.
"Je specificeert als ontwikkelaar wat je gebruikt en met een tool kun je dan alles updaten. Die wil je ook echt gebruiken, want als je dit handmatig gaat doen, is daar geen beginnen aan."

LEGACY LIBRARIES

Updaten heeft echter alleen nut als er ook daadwerkelijk nog aan een component gewerkt wordt. "Als een component verouderd is, moet je er op een gegeven moment toch van af. Je wilt niet afhankelijk zijn van een dood paard. Of je moet het forken en het zelf gaan onderhouden. Maar in de meeste gevallen is dat niet haalbaar", aldus Van der Storm.
Overigens zijn er wel uitzonderingen op deze regel. Er zijn ook projecten die nauwelijks veranderen, omdat ze heel stabiel zijn. Ze leven dus nog wel, maar je hoeft nauwelijks te updaten.
Een voorbeeld daarvan is het databasepakket SQL Lite. "Dat heeft soms alleen wat bugfixes. Driekwart van de code bestaat alleen al om het ding te testen. Dus dat is wel echt een stabiele dependency, ook al verandert er weinig aan de code zelf."

TOOLS VOOR DEPENDENCYONDERHOUD

Welke tools kun je goed gebruiken om je dependencies te onderhouden? Natuurlijk zijn er de packagemanagers die hierbij helpen, maar er is meer op de markt. Tijs van der Storm en Tibor Lapikas geven tips.

OSSMETER

Als je een opensourcedependency gebruikt, wil je natuurlijk wel weten hoe gezond deze is. "Collega's van mij op het CWI hebben meegewerkt aan een Europees onderzoeksproject, OSSMETER, om te onderzoeken hoe je aan repositories op GitHub kunt zien of een project gezond is. Dat zie je bijvoorbeeld als er veel patches zijn, veel pull requests geaccepteerd worden en als de mailinglijst actief is", aldus Van der Storm. Dergelijke informatie wordt door de OSSMETER verzameld en in een dashboard geplaatst. Aan de hand daarvan kunnen bedrijven beslissen of het verantwoord is om een repository te gebruiken. De tool werkt niet alleen voor GitHub, maar ook voor andere opensourcerepositories.

OPEN SOURCE HEALTH

Ook SIG heeft een tool ontwikkeld, gericht op de gezondheid van opensourcelibraries. Deze tool, Open Source Health, is onderdeel van Sigrid, het software-assuranceplatform van de organisatie. "We brengen daarmee een aantal risico's in kaart, bijvoorbeeld de kwetsbaarheden in de code", vertelt Lapikas. "Maar we kijken ook of het nog wordt onderhouden, of er nog actief aan wordt gewerkt en of er releaseversies zijn die problemen kunnen opleveren."
Het platform kijkt daarnaast naar welke licenties er gebruikt worden. "Opensourcesoftware heeft verschillende soorten licenties. Die licenties hebben impact op waar je je software mag gebruiken, en dat gaat nog weleens fout."
Door al dit soort risico's te verzamelen, ontstaat er een duidelijk beeld van welke libraries als eerste geüpdatet of op een andere manier aangepakt moeten worden.