> WHITEPAPER

OPEN SOURCE SOFTWARE ONDER CONTROLE

ZONDER VENDOR LOCK-IN
VAN BELEID NAAR ACTIE

Door: Marcel Kornegoor & Marco Lammers

AT COMPUTING

ATYPICAL OPEN SOURCE GURUS

versie: 12/25

TL;DR

Bij vrijwel iedere organisatie vormt open source software (OSS) een onzichtbare spil in de ICT-infrastructuur. Toch blijft vaak een zekere terughoudendheid bestaan: “We willen een leverancier met contract, want anders lopen we onaanvaardbare risico’s en nemen we onze verantwoordelijkheid niet”, is een geluid dat regelmatig klinkt. Dat is begrijpelijk, maar het berust op een verouderd denkmodel. Tijd om het anders te zien!

> DE KERN VAN HET PROBLEEM

Veel bedrijven redeneren als volgt: "We zijn geen softwarebedrijf, dus we willen een contract (SLA) met een softwareleverancier - we willen deze leverancier verantwoordelijk maken en kunnen bellen als er iets mis gaat.” Dat klinkt logisch, maar het creëert vooral schijnzekerheid.

Geen enkel contract garandeert veiligheid. In een contract of overeenkomst worden vooral afspraken gemaakt over te nemen maatregelen en reactiesnelheid bij een incident of verstoring. Soms is een contract voorzien van claim-modules die recht geven op financiële compensatie wanneer een probleem of storing langer duurt dan overeengekomen. Ook dat zijn geen harde garanties dat de kwaliteit van de afgenomen software op orde is, het zegt alleen iets over inspanningen die worden gedaan op gebied van kwaliteit of bepaalde kaders en richtlijnen die gehanteerd worden bij het vervaardigen van de software. Een contract is dus feitelijk niet meer dan een vorm van risicobeheersing die een risico vermindert. Het is geen maatregel die een risico daadwerkelijk wegneemt.

>

DE STRATEGISCHE KRACHT VAN OPENHEID

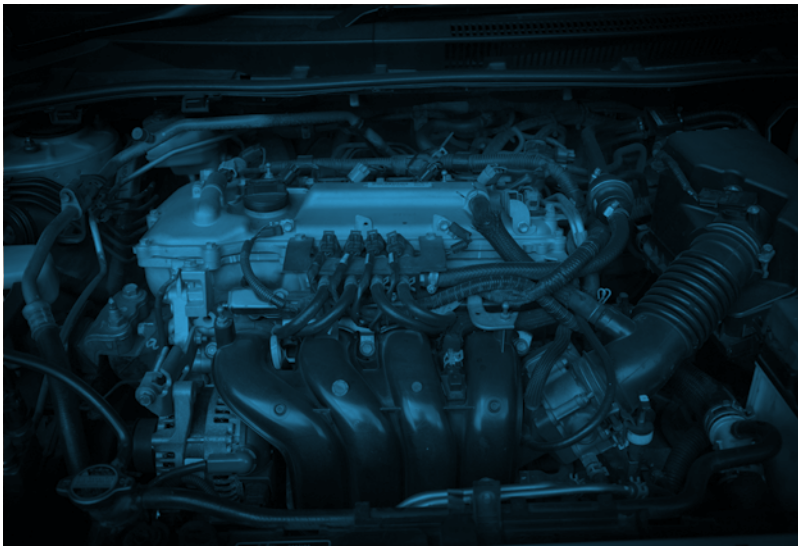
Bij closed-source (propriëtaire) software weet je nooit precies wat je in huis haalt; je vertrouwt volledig op de blauwe ogen van de vendor. Alleen de vendor kan “onder de motorkap” en weet als enige hoe de software intern precies werkt. Bij OSS kun je wel onder de motorkap. Sterker nog: je kunt tot in het kleinste detail zien hoe de (elektro) motor precies in elkaar steekt. Dit biedt strategische voordelen die veel verder gaan dan alleen het besparen van licentiekosten. Het belangrijkste voordeel is ***digitale soevereiniteit***: je wordt bevrijd van vendor lock-in. Je bent niet langer afhankelijk van de grillige roadmap, prijsverhogingen of (vijandelijke) overnames van een leverancier, maar houdt zelf de controle over de levenscyclus van je software. Daarnaast is de ***innovatiesnelheid*** binnen actieve communities vaak vele malen hoger dan binnen de muren van een enkele softwarefabrikant. Doordat je de broncode kunt inzien, inspecteren en aanpassen, verandert jouw rol van passieve consument naar een organisatie die zelf aan het stuur kan zitten van haar IT-landschap. Je hebt invloed op de software en kunt op diverse manieren bijdragen aan het verbeteren ervan. Zo pak je pas echt eigenaarschap over je IT-risico’s!

De praktijk laat zien dat grote leveranciers, ondanks een afgesloten contract, óók last hebben van storingen/uitval en problemen. Denk aan de recente GitLab-breach bij IBM’s Red Hat, waarbij gevoelige gegevens van deze IT-reus en haar klanten werden buitgemaakt. Of aan storingen en grote security-incidenten bij Microsoft of Amazon Web Services door de jaren heen — van Exchange-lekken tot het wereldwijde CrowdStrike-incident. Ook CDN-providers zoals CloudFlare en andere cloudaanbieders die veelal werken op basis van miljoenencontracten met grootzakelijke klanten hebben grote verstoringen gehad.

Deze voorbeelden tonen één ding aan: een vendor-contract neemt het risico niet *gegarandeerd* weg. Uiteindelijk blijf jij eindverantwoordelijk voor de geleverde diensten en daarmee voor zaken als monitoring, configuratie, beschikbaarheid, integriteit, vertrouwelijkheid (BIV) én continuïteit. Niet de aanwezigheid van een contract, maar de adequate uitvoering van (onderhouds) werkzaamheden bepaalt in grote mate de veiligheid en stabiliteit van software en IT-systemen.

Het veiligheidsdenken van de in Nederland geboren professor Sidney Dekker laat zien dat veiligheid ontstaat door te focussen op hoe werk écht wordt gedaan (work-as-done), niet slechts op wat beleidsmakers verzinnen (work-as-imagined). Wanneer je OSS als bedrijf ziet als iets waarvoor een leverancier moet zorgen — en jij er verder niets aan kunt/wilt doen — dan mis je de vraag die ertoe doet: hoe zorgen we dat het voor ons veilig en betrouwbaar werkt?

Het omdenken gaat nog een stap verder, want de uitdagingen om OSS verantwoord in te zetten zijn simpelweg anders dan bij propriëtaire software. Het zorgt echter ook voor kansen en mogelijkheden die propriëtaire software niet biedt. Laten we hier in de volgende paragraaf eens op inzoomen.



AANDACHTSPUNTEN BIJ OSS-GEBRUIK

We kijken naar drie belangrijke aandachtspunten die vaak over het hoofd worden gezien, maar essentieel zijn wanneer je OSS professioneel en verantwoord wilt gebruiken.

1. ECOSYSTEEM EN ONDERHOUD

Bij commerciële software kun je denken: er is een leverancier, dus het is geregeld. Bij OSS is dat fundamenteel anders — de afhankelijkheid ligt bij een gemeenschap (*community*) van ontwikkelaars, in veel gevallen met een bijbehorend project: dit zijn de mensen die de software onderhouden. Niet ieder project of iedere community is even volwassen of actief. Het is daarom verstandig om selectiecriteria te hanteren, zoals bijvoorbeeld:

- Is het project actief (van wanneer is de laatste update?)?
- Heeft het project een levendige community?
- Zijn er regelmatig releases?
- Is er een roadmap?
- Is er een stabiele vorm van funding (een commercieel verdienmodel of sponsoring door individuen of bedrijven)

Dit soort vragen geeft zicht op het onderhouds- en continuïteitsrisico.

2. LICENTIE-, COMPLIANCE- EN SUPPORT-VRAAGSTUKKEN

Soms wordt de uitspraak gedaan dat “vendor contracts” nodig zijn om aan compliance-frameworks te voldoen. Dat is onjuist. In werkelijkheid gaat het erom dat je de risico’s voor je bedrijfsvoering beheerst door middel van adequate maatregelen. Hoe je dit realiseert wordt in compliance-frameworks niet afgedwongen: het kan dus op meerdere manieren. Dit hoeft dus niet per se een contract te zijn. Je kunt OSS dus prima inzetten, zolang je maar goed nadenkt over wat je aan OSS gebruikt en hoe je het gecontroleerd/verantwoord gebruikt. Dit betekent dus dat je ervoor kunt kiezen om zelf zaken als patching, onderhoud, (security) incidenten en operationele verantwoordelijkheid op je te nemen en dat niet aan een externe leverancier over te laten. Mocht je dit toch niet zelf willen doen, dan kun je bij OSS het onderhoud ook “los” uitbesteden, zonder daarmee aan een vendor vast te zitten. Dit is vergelijkbaar met onderhoud aan een auto. Als je een auto koopt dan ben je niet verplicht om het onderhoud van de auto bij de fabrikant of dealer te doen. Je kunt ook voor een onafhankelijk garagebedrijf kiezen. Om de risico’s onder controle te houden, is het dus (wederom) belangrijk om kritisch te kijken welke OSS je inzet. Volwassen OSS projecten geeft je enorm veel inzicht in hun status. Denk aan licentie, SBOM (software bill of materials, oftewel de lijst met "ingrediënten" waaruit de software bestaat), project governance, security tests, bekende kwetsbaarheden en bugs, code reviews etc. Deze vorm van transparantie maakt feitelijk dat je veel beter weet waar je aan begint en dus welke risico’s je loopt. Organisaties zoals de CNCF (Cloud Native Compute Foundation) hebben voor projecten die onder hun auspiciën vallen duidelijke richtlijnen op gebied van governance.

3. VAN BELEID NAAR ACTIE

Beleidsstukken vol met hoofdstukken en paragrafen zijn mooi, maar wanneer ze een slaapplaats vinden op het intranet dan brengen ze weinig effect teweeg. Wat telt is: wie doet wat, wanneer, waarmee, en hoe meten we dat? Dekker zou zeggen: richt je op wat mensen werkelijk doen, niet op wat in het beleidsboek staat.Creëer daarom liever een beperkte set aan concrete maatregelen, met eenvoudige checklists, duidelijke eigenaren en teken het proces uit. Zorg dat je begint met uitvoering! Beschrijf daarna wat je daadwerkelijk doet, in plaats van andersom.

MOGELIJKE OPLOSSINGEN IN DE PRAKTIJK

Er zijn twee concrete routes die je kunt bewandelen om OSS-gebruik beter te beheersen — zonder de “als we een vendor hebben dan is het veilig” illusie.

A. PAS HET “90% PRINCIPE” TOE

Je hoeft niet elk risico tot op het hoogste detailniveau met beleid af te dichten. 100% veilig of 100% risicovrij bestaat simpelweg niet. Focus daarom op de essentiële 90% van controls die het grootste effect hebben:

- Inventarisatie: weet welke OSS-componenten je gebruikt (SBOM, dependency-lijst). Deze SBOM kun je opslaan in bijvoorbeeld een CMDB of in je versiebeheersysteem als onderdeel van je software-project.
- Evaluatie: hanteer een checklist met o.a. kwaliteitscriteria, toegestane licenties en project-gezondheid
- Processen: definieer hoe nieuwe OSS wordt gekozen, getest, geïmplementeerd en onderhouden
- Continuïteit: bepaal wat je doet als er binnen een project (“upstream”) onvoorziene veranderingen optreden (stopzetten, wisseling van de wacht, of een kritieke kwetsbaarheid ontstaat)
- Eigenaarschap: wijs mensen aan die verantwoordelijk zijn voor de betreffende OSS
- Tegenprestatie: een veel vergeten onderdeel van het gebruik van OSS is dat je er als gebruiker (“klant”) ook op diverse manieren aan kunt bijdragen. Kom je erachter dat een project heel interessant is, maar er een SBOM ontbreekt? Help het project door een SBOM op te stellen. Is de documentatie niet op orde? Begin met schrijven. Ook een donatie (geld, serverruimte/hosting, resources/uren) kan het project helpen en gelijktijdig risico’s verlagen/wegnemen. Het is soms bijzonder dat bedrijven er geen enkel probleem mee hebben om €100.000 aan licenties te betalen bij een vendor, maar moeilijk doen om €1.000 aan een OSS project te doneren….

Met deze pragmatische aanpak vermijd je het eindeloos verfijnen van beleid en begin je echt met het onder controle krijgen van je risico’s. Wil je een volledigere raamwerk met criteria voor het selecteren van een OSS project? Hanteer dan onze OSS Checklist.

B. VERSTERK HET INTERNE EIGENAARSCHAP

Stel jezelf de vraag: “Welk deel van ons software-landschap is niet gekoppeld aan een leverancier?” Je komt dan vermoedelijk uit bij zaken zoals (KPI) dashboards, low-code apps, integraties/koppelingen en een berg (automatiserings)scripts (PowerShell, Bash, Perl, Python, VB…). Het is daarom niet realistisch om te zeggen: “We zijn geen softwarebedrijf dus we willen geen verantwoordelijkheid.” Je hebt als bedrijf vrijwel altijd een vorm van software-ontwikkeling in huis. En juist dát maakt het noodzakelijk om op dit vlak actief te zijn. Zorg ervoor dat je:

- interne competenties voor OSS onderhoud, net zoals die competentie nodig is voor onderhoud van eigen apps, code en scripts
- procedures hebt voor vulnerability-monitoring en updates, zodat risico’s adequaat worden gesignaleerd.
- een fallback-plan hebt voor OSS-componenten waarbij de upstream stopt. Dit is feitelijk “gewoon” een business continuity plan, maar dan met OSS software als scope.

Deze aanpak maakt het verschil of je OSS-gebruik een bedrijfsrisico vormt of dat je het als een strategisch voordeel kunt inzetten.

WAAROM IS DIT BELANGRIJK

- Er zijn steeds vaker audits, leveranciersbeoordelingen en compliance-vragen waarbij OSS onderdeel is van de keten.
- Projecten zoals de Cyber Resilience Act vergroten de druk op transparantie in softwareleveranciers en componenten.
- OSS maakt zo'n groot deel uit van een moderne software-stack dat je het niet mag negeren. Onderzoek heeft naar voren gebracht dat iedere vorm van software wereldwijd voor 70-90% uit OSS bestaat. Het zit dus per definitie in de haarvaten van iedere organisatie.
- Door te focussen op uitvoering en niet op complete perfectie, versnel je de adoptie en verlaag je risico's.

VERGELIJKING VAN RISICOFACTOREN TUSSEN OPEN SOURCE EN PROPRIËTAIRE SOFTWARE

Risicofactor	Open Source Software	Propriëtaire Software
Afhankelijkheid van leverancier	Afhankelijkheid is beperkt. Grote projecten worden vaak beheerd door een stichting of community (steward ownership).	Hoge afhankelijkheid van één leverancier. Roadmap en prioriteiten liggen extern.
Toegang tot broncode	In principe altijd beschikbaar, tenzij specifieke licentievoorwaarden beperkingen opleggen.	Niet beschikbaar. Volledige "black box".
Overstappen naar andere oplossing	Meestal goed mogelijk, zeker bij gebruik van open standaarden en open dataformaten.	Mogelijk, maar vaak complex door databinding, gesloten dataformaten of vendor lock-in.
Beschikbaarheid van updates	Grote projecten hanteren een vaste releasecyclus. Kleinere projecten volgen een community-driven model zonder strikte ritmiek.	Afhankelijk van de planning, prioriteiten en commerciële belangen van de leverancier.
Oppakken van issues en incidenten	Grote projecten hebben formele processen; bij kleinere projecten varieert dit sterk.	Meestal afgehandeld via SLA's of betaalde supportcontracten.
Licentie- of supportmodel	Licenties zijn doorgaans permissief of copyleft. Commerciële support is vaak beschikbaar via derden, zeker bij grote projecten.	Licentie is een verplicht onderdeel van het product. Supportcontracten zijn vaak onderdeel van het businessmodel.
Transparantie van de ontwikkelcyclus	Volledige transparantie (issues, PR's, commits).	Geen inzicht. Je bent afhankelijk van wat de leverancier communiceert.
Continuïteitsrisico	Community kan doorgaans doorgaan, behalve bij projecten met busfactor 1. Bij meningsverschillen kan een project worden geforked (bijv. MariaDB/MySQL, Forgejo/Gitea).	Bij verkoop of overnames kan alles veranderen (licenties, prijzen, roadmap), zoals bijvoorbeeld bij de overname van VMware door Broadcom.
Zichtbaarheid van security-kwetsbaarheden	Volledig inzicht in broncode. Onafhankelijke audits zijn mogelijk.	Geen zichtbaarheid. Afhankelijk van leverancier of externe audits die door de leverancier worden besteld.
Innovatietempo	Doorgaans hoog, zeker in grote ecosystemen (Linux, Kubernetes). Kleinere projecten kunnen worden beperkt door capaciteit.	Afhankelijk van de strategische en commerciële belangen van de leverancier.
Beschikbaarheid van expertise	Voor grote projecten ruim beschikbaar (Linux, Python, Kubernetes). Voor kleinere projecten soms schaars.	Expertise vaak via leveranciers en partner-ecosystemen beschikbaar. Meestal beperkte keuzevrijheid en vaak hogere kosten.

Tot slot

Open source software biedt enorme kansen — transparantie, flexibiliteit, innovatie, lagere kosten, geen/minder vendor lock-in — maar kent ook risico's. Deze zijn niet zozeer groter dan commerciële software, maar fundamenteel anders. Ze lijken veel meer op de risico's voor eigen ontwikkelde tools. De vraag is daarom niet: Heb ik een contract met een vendor? De vraag is: Heb ik een plan?

Over AT Computing

Veiligheid koop je niet af met een contract, maar borg je met kennis en vakmanschap. Dat is precies waar AT Computing voor staat. Als specialist in Linux, Python en open source software zoals Kubernetes, Docker, Ansible en Git helpen wij organisaties al sinds 1985 om qua IT op eigen benen te staan.

Wij geloven niet in vendor lock-in of 'black boxes', maar in transparantie en kennisdeling. Of we nu adviseren over complexe infrastructuren of jouw engineers opleiden via onze trainingen: het doel is altijd dat jullie zelf stevig aan het stuur zitten en wij zo snel mogelijk weer overbodig worden.

Arnhemsestraatweg 33
6881 ND Velp (GLD)
Nederland

www.atcomputing.nl