Nationaal Cyber Security Centrum
Ministerie van Justitie en Veiligheid

Nieuwe ontwikkelingen in statische analyse

Onderzoeksrapport

Jan Rooduijn

Static application security testing (SAST) is een methode om automatisch kwetsbaarheden in software te ontdekken zonder die software te draaien. Het gebruik van SAST-tools wordt door vrijwel alle richtlijnen voor veilige softwareontwikkeling voorgeschreven. Maar hoe maak je een keuze uit de grote verscheidenheid aan beschikbare SAST-tools? En hoe waardeer je de bewering van ontwikkelaars van moderne tools dat die effectiever zouden zijn dan meer gevestigde tools? Tegen welke obstakels lopen ontwikkelaars in de praktijk aan bij het gebruik van SAST-tools? Dit onderzoek heeft als doel om Nederlandse softwareontwikkelaars verder te helpen bij het beantwoorden van deze vragen.

Samenvatting

De hoofdvraag van dit onderzoek luidt: hoe kunnen Nederlandse organisaties profiteren van nieuwe ontwikkelingen in statische analyse? Eerst zijn op basis van een aantal criteria drie nieuwe SAST-tools geselecteerd: CodeQL, Infer en Semgrep. Deze tools zijn vergeleken met de meer gevestigde tool SonarQube.
De belangrijkste resultaten zijn:

- Zowel CodeQL als Semgrep ondersteunt het schrijven van eigen detectieregels. Deze nieuwe ontwikkeling biedt verschillende voordelen. Organisaties kunnen bijvoorbeeld profiteren van detectieregels die worden bijgedragen door derde partijen. Eigen regels maken het ook makkelijker om SAST-tools optimaal te benutten door ze specifiek te configureren naar de eigen organisatie.
- Uit verschillende vergelijkingen in de academische literatuur blijkt dat de nieuwe SAST-tools accurater zijn in het detecteren van kwetsbaarheden dan de gevestigde referentietool. Het is daarom de moeite waard voor organisaties om te experimenteren met nieuwe tools.
- Verschillende SAST-tools dekken verschillende gebieden. Het kan daarom de moeite waard zijn om meerdere SAST-tools te gebruiken.

Het tweede deel van dit onderzoek richt zich op obstakels die softwareontwikkelaars ervaren bij het gebruik van SAST-tools. Op basis van academische literatuur zijn de volgende obstakels als de belangrijkste geïdentificeerd:

1. Te veel foutpositieven
2. Onvoldoende informatieve meldingen
3. Te weinig hulp bij het oplossen van meldingen
4. Onvoldoende workflow integratie
5. Gebrek aan incorporatie van gebruikersfeedback

Door middel van een handmatige analyse is nagegaan in hoeverre de nieuwe tools deze obstakels adresseren. Sommige, maar lang niet alle obstakels worden door de nieuwe tools geadresseerd. Dit bevestigt dat het de moeite waard is voor organisaties om met de nieuwe tools te experimenteren, maar toont ook aan dat er nog veel ruimte voor verbetering is in de ontwikkeling van SAST-tools.

In het laatste deel van dit onderzoek wordt specifiek de Nederlandse situatie onder de loep genomen. Hiertoe is een enquête uitgevoerd onder mensen die aan (veilige) software-ontwikkeling werken bij organisaties die onderdeel zijn van de Rijksoverheid, de vitale sector, of bij softwareleveranciers van deze organisaties. De belangrijkste bevindingen zijn:

- De nieuwe SAST-tools worden in Nederland relatief weinig gebruikt in verhouding tot andere landen.
- SAST-tools zijn bij alle respondenten zeer belangrijk voor het ontdekken van kwetsbaarheden.
- Organisaties in de publieke sector ervaren meer obstakels bij het gebruik van SAST-tools dan die in de financiële sector en de IT-sector.

De bevindingen van dit onderzoek geven aanleiding voor de volgende aanbevelingen:

- Organisaties die software ontwikkelen zouden SAST-tools moeten gebruiken om kwetsbaarheden te detecteren.
- Het kan voor organisaties die al lang dezelfde SAST-tool gebruiken de moeite waard zijn om met een nieuwe tool te experimenteren. Met name tools die het mogelijk maken eigen regels te schrijven bieden verschillende voordelen.
- Het kan de moeite waard zijn om meerdere SAST-tools tegelijkertijd te gebruiken.
- Het is nuttig om kennis in huis te ontwikkelen om eigen regels voor SAST-tools te schrijven. Kennisuitwisseling tussen verschillende Nederlandse sectoren kan het gebruik van SAST-tools verbeteren.

Dank aan Natalia Kadenko (NCSC) voor het vervullen van een klankbordrol, aan Rik van Dijk (NCSC) voor review van dit manuscript, en aan Erik Poll (Radboud Universiteit) voor review, literatuursuggesties en hulp bij het vinden van respondenten voor de enquête.

Inleiding

Kwetsbaarheden in software vormen de basis van veel cyber-aanvallen. Neem bijvoorbeeld de bekende log4j-casus. Daarbij hadden ontwikkelaars onvoldoende controle ingebouwd van de input die een gebruiker kan geven. Aangezien er zeel veel gebruik gemaakt werd van log4j, leidde dit tot een grote hoeveelheid kwetsbare systemen, en dus ook cyberaanvallen.

Het is dan ook essentieel om aandacht te besteden aan beveiliging tijdens het softwareontwikkelproces. Hier bestaan verschillende best practices voor, zoals bijvoorbeeld die in de Security Development Lifecycle van Microsoft.[1] Het NCSC heeft al eerder onderzoek gedaan naar verschillende van deze best practices, waaronder Threat Modeling[2] en SBOM's[3].

Een ander belangrijk onderdeel van veilig softwareontwikkeling is het uitvoeren van automatische testen. Dit wordt onder andere aangeraden in de richtlijnen van het NCSC.[4] Men verdeelt deze testen meestal onder in drie categorieën:

1. SAST (Static Application Security Testing): hierbij gaat het om het analyseren van software zonder die eerst te draaien.
2. DAST (Dynamic Application Security Testing): hierbij wordt de software juist wel gedraaid tijdens het testen. Een populaire vorm van DAST is *fuzzing*.
3. SCA (Software Composition Analysis): dit is gerelateerd aan de eerdergenoemde SBOM's. Het gaat om het automatisch herkennen van de alle software waar de gescande software afhankelijk van is, zodat er gewaarschuwd kan worden als een van die afhankelijkheden een kwetsbaarheid bevat.

Voor elk van deze categorieën is er een actieve industrie die werkt aan steeds bruikbaardere tools. Daarbij wordt geleund op nieuwe inzichten vanuit zowel de wetenschap als de praktijk.

Dit onderzoek beperkt zich tot tools van de eerste categorie: SAST-tools. Statische analyse is een relatief volwassen vorm van testen, die in het grootste deel van professionele softwareontwikkelingspijplijnen wordt gebruikt. Bekende voordelen van statische analyse ten opzichte van andere vormen van testen zijn: het detecteren van kwetsbaarheden vroeg in het ontwikkelproces, uitgebreide codedekking, en natuurlijke integratie in het CI/CD-proces.

Er zijn in het afgelopen decennium verschillende nieuwe SAST-tools beschikbaar gekomen, die meestal beweren effectiever te zijn dan de status quo.

De hoofdvraag van dit onderzoek luidt: **hoe kunnen Nederlandse organisaties profiteren van nieuwe ontwikkelingen in statische analyse?**

Doelgroep

Software Engineer
Quality Assurance/Test Engineer
Security Engineer
DevOps Engineer
IT/Infrastructure Manager
Security Analyst
System Administrator
Compliance Officer

[1] https://www.microsoft.com/en-us/securityengineering/sdl/practices
[2] https://www.ncsc.nl/documenten/publicaties/2024/mei/7/index
[3] https://www.ncsc.nl/documenten/publicaties/2023/juli/5/sbom-startersgids
[4] https://www.ncsc.nl/documenten/publicaties/2019/mei/01/beleids--en-beheersingsrichtlijnen-voor-de-ontwikkeling-van-veilige-software

Deelvragen en methode

Om de onderzoeksvraag meer behapbaar te maken, wordt die onderverdeeld in verschillende deelvragen. Elke deelvraag is zelf ook weer verder onderverdeeld.

Deelvraag 1: wat zijn de nieuwe ontwikkelingen in statische analyse?

Het moet concreet worden wat precies wordt verstaan onder "nieuwe ontwikkelingen in statische analyse". Omdat er elk jaar zeer veel nieuwe tools beschikbaar komen, wordt er in dit onderzoek een selectie gemaakt, onder andere op basis van populariteit.

DV1.1 Wat zijn de meest prominente nieuwe tools voor statische analyse?

Om goed te begrijpen hoe Nederlandse organisaties van de nieuwe tools kunnen profiteren, is het van belang om te begrijpen hoe die werken.

DV1.2 Op welke theoretische inzichten zijn deze tools gebaseerd?

Het is ook nuttig om te weten hoe de nieuwe tools zich tot elkaar verhouden en tot meer gevestigde tools, zodat organisaties kunnen inschatten of het de moeite waard is om over te stappen naar een nieuwe tool.

DV1.3 Hoe verhouden deze tools zich tot elkaar en tot meer gevestigde tools?

Deelvragen 1.1-1.3 worden beantwoord door middel van een literatuurstudie.

Deelvraag 2: in hoeverre adresseren deze nieuwe ontwikkelingen de behoeften van ontwikkelaars?

Uiteindelijk zijn nieuwe ontwikkelingen alleen relevant als ze bijdragen aan het doel om software veiliger te maken. Het is daarbij van belang om te weten waarom softwareontwikkelaars SAST-tools wel of niet gebruiken.

DV2.1 Wat zijn de belangrijkste obstakels die ontwikkelaars ervaren bij het gebruik van SAST-tools?

Vervolgens is het interessant om na te gaan of de nieuwe tools deze obstakels ook daadwerkelijk adresseren.

DV2.2 In hoeverre worden deze obstakels geadresseerd door de nieuwe tools?

Deelvraag 2.1 wordt beantwoord door middel van een literatuurstudie. Deelvraag 2.2 wordt beantwoord door de antwoorden op de eerdere deelvragen te vergelijken.

Deelvraag 3: hoe kunnen de nieuwe tools bijdragen aan de veiligheid van in Nederland ontwikkelde software?

De primaire doelgroep van dit onderzoek bestaat uit softwareontwikkelaars die werken bij de doelgroepen van het NCSC (op dit moment nog de Rijksoverheid en de vitale sector), of bij organisaties die software aan hen leveren. Het is daarom van belang om meer inzicht te krijgen in het gebruik van SAST-tools binnen deze groep.

DV3.1 Gebruiken Nederlandse organisaties de nieuwe tools?

Het is belangrijk dat elke organisatie een tool kan kiezen die bij ze past.

DV3.2 Welke nieuwe tools zijn geschikt voor welke organisaties?

Tot slot is het nuttig om te weten welke problemen er kunnen ontstaan als een organisatie daadwerkelijk een nieuwe tool wilt gaan implementeren.

DV3.3 Welke problemen kunnen ontstaan bij de implementatie van nieuwe tools?

Deelvragen 3.1 en 3.2 worden beantwoord door middel van een enquête onder Nederlandse softwareprofessionals. Deelvraag 3.3 wordt niet beantwoord in dit rapport, maar kan onderwerp zijn van een eventueel vervolgonderzoek.

DV1.1: Toolselectie

Om deelvraag 1.1 te beantwoorden is begonnen met een lange lijst van SAST-tools.[5] Via een proces van eliminatie zijn tools geselecteerd die:

Meer dan één populaire programmeertaal ondersteunen. Aangezien dit onderzoek ten goede moet komen aan Nederlandse organisaties, is het belangrijk dat de geselecteerde SAST-tools programmeertalen ondersteunen die die organisaties gebruiken. Dit rapport beschouwt een programmeertaal als populair wanneer die in de top 5 van de TIOBE-index[6] voorkomt. Door dit criterium vallen tools zoals *Brakeman* en *Clippy* af.

Meer ontdekken dan slechts problemen gerelateerd aan stijl en structuur of aan afhankelijkheden. Een probleem gerelateerd aan stijl en structuur is een probleem dat niet gaat om de betekenis van de code, maar slechts om de manier waarop die geformatteerd is. Een kwetsbare afhankelijkheid is een kwetsbaarheid die zich bevindt in code waar de gescande code afhankelijk van is, in plaats van in de gescande code zelf. Dit criterium sluit bijvoorbeeld de tools *Black* en *Dependency-Check* uit.

Nieuw zijn. Gezien dit onderzoek zich richt op nieuwe ontwikkelingen, worden alleen tools in beschouwing genomen die ontwikkeld (of grondig herontwikkeld) zijn in het de afgelopen 10 jaar. Hiermee worden tools als *Coverity Scan*, *PMD*, *SonarQube* en *Veracode* uitgesloten.

Prominent zijn. Om de studie behapbaar te houden beperkt dit onderzoek zich tot slechts de meest bekende tools, gemeten aan de hand van sterren op GitHub. De geselecteerde tools hebben allemaal minstens 5.000 sterren. Dit sluit meer obscure tools uit, zoals *MOPSA* en *TScanCode*, maar ook *FindSecBugs.*

Hiermee kan de eerste deelvraag worden beantwoord:

DV1.1 Wat zijn de meest prominente tools voor statische analyse?

De meest prominente nieuwe SAST-tools die minstens één populaire programmeertaal ondersteunen en scannen op meer dan slechts formatteerproblemen en kwetsbare afhankelijkheden zijn:

1. CodeQL[7]
2. Infer[8]
3. Semgrep OSS[9]
4. Snyk Code[10]

Voor de rest van dit rapport voegen we nog een extra criterium toe. Namelijk, dat de geselecteerde SAST-tool:

Transparante methoden heeft. Aangezien er ook een deelvraag is over de onderliggende techniek van deze tools, is het belangrijk dat daar iets over bekend is. Idealiter is de tool open source, maar als dat niet zo is dan moet op z'n minst moet ergens duidelijk beschreven staan hoe de tool werkt. Dit criterium sluit *Snyk Code* uit.

DV1.2: De geselecteerde tools

In deze paragraaf worden de drie geselecteerde tools kort beschreven.

CodeQL. Het kernidee van CodeQL is om broncode als data te beschouwen. Een geheel programma wordt dan een database, waar informatie uit kan worden opgehaalde door middel van *queries* (de QL staat dan ook voor *Query Language*). Deze techniek heeft een rijke geschiedenis,[11] en vormt de basis van CodeQuest,[12] een vroege voorganger van CodeQL. CodeQuest gebruikt een variant van de taal Datalog als *query language*, die een verbetering vormt ten opzichte van eerder werk door recursieve *queries* toe te laten zonder dat de complexiteit te hoog wordt. De *query language* van CodeQL heeft object-georiënteerde functionaliteit en is specifiek ontwikkeld om broncode mee te analyseren. Deze taal heette oorspronkelijk QL en is ontwikkeld door het bedrijfje Semmle,[13] een spin-off van de Universiteit van Oxford. In 2019 is Semmle overgenomen door het bedrijf GitHub, dat een jaar eerder onderdeel van Microsoft geworden was.

CodeQL onderscheidt zich van de meeste andere SAST-tools door nadruk te leggen op de mogelijkheid voor gebruikers om hun eigen *queries* te schrijven. Desondanks is CodeQL ook volledig geautomatiseerd te gebruiken, vanwege de grote hoeveelheid beschikbare queries voor klassen van kwetsbaarheden. Het feit dat gebruikers eigen queries kunnen schrijven maakt CodeQL verrassend veelzijdig. Naast gebruik als traditionele SAST-tool, wordt het bijvoorbeeld ook gebruikt als ondersteuning bij handmatige codereview, voor het vinden van zogeheten spectre gadgets,[14] en voor het identificeren van kwaadaardige softwarepakketten.[15]

Tegenwoordig is de meest populaire manier om CodeQL te draaien via GitHub's Code Scanning platform.[16] Deze functionaliteit is gratis voor open source projecten, en kan tegen betaling aangezet worden op de repositories van klanten van GitHub's Cloud omgeving.

[5] https://github.com/analysis-tools-dev/static-analysis
[6] https://www.tiobe.com/tiobe-index/
[7] https://codeql.github.com/
[8] https://fbinfer.com/
[9] https://semgrep.dev/
[10] https://snyk.io/product/snyk-code/

[11] Linton 1984.
[12] Hajiyev et al. 2006.
[13] https://www.cs.ox.ac.uk/innovation/research-impact/case-semmle.html
[14] https://github.com/google/security-research/tree/master/pocs/cpus/spectre-gadgets
[15] Gobbi & Kinder 2023.
[16] https://docs.github.com/en/code-security/code-scanning/introduction-to-code-scanning/about-code-scanning

CodeQL ondersteunt 10 programmeertalen, waaronder Python, C++, en Java.[17] Het heeft ook regels voor codekwaliteit, maar richt zich met name op kwetsbaarheden die een veiligheidsrisico vormen. De nieuwheid van CodeQL blijkt met name uit het feit dat het gebaseerd is op relatief recent academisch onderzoek. CodeQL heeft ongeveer 8.000 sterren op GitHub. De broncode van de *engine* van CodeQL is niet openbaar, maar die van alle *queries* wel.

Infer. De basis van Infer ligt in *separation logic*, een formele logica waarmee je kan redeneren over wat er in het geheugen van een computer zou gebeuren wanneer die een gegeven programma zou uitvoeren. Deze logica maakt het mogelijk om *los van elkaar* (vandaar *separation*) over verschillende gedeeltes van het geheugen te redeneren. Infer wordt ook wel een *verification-based* SAST-tool genoemd, omdat het probeert formeel te bewijzen dat een programma aan bepaalde voorwaarden voldoet (zoals de afwezigheid van *null pointer dereferences*). Infer doet dit door een soort samenvatting te maken van elke procedure van een programma. Deze samenvatting bestaat onder andere uit *preconditions*: voorwaarden die moeten gelden op het moment dat de procedure wordt aangeroepen. Waar *preconditions* bij formele verificatie normaal gesproken door de gebruiker moeten worden opgesteld, leidt Infer die automatisch af door middel van een techniek die bi-abductie heet.[18]

De ontwikkeling van Infer is terug te voeren tot de uitvinding van separation logic aan het begin van deze eeuw.[19] De effectiviteit van Infer's voorgangers Smallfoot[20] en SpaceInvader[21] leidde ertoe dat hun ontwikkelaars in 2009 een bedrijfje genaamd Moinodics oprichtten. Dit bedrijf werd in 2013 door Meta gekocht, waar Infer tot op de dag van vandaag in ontwikkeling is. De kracht van Infer is schaalbaarheid, mogelijk gemaakt door de wortels in separation logic. Omdat elke procedure afzonderlijk wordt geanalyseerd, is incrementele analyse met Infer mogelijk. Dat betekent dat Infer een nieuw stukje toegevoegde code kan scannen, zonder dat het hele programma opnieuw gescand moet worden.

Infer is gratis en wordt gebruikt door een groot aantal kleine en grote organisaties, waaronder het Britse NCSC.[22]

Infer ondersteunt C++, Java en Objective C. Daarnaast is er een variant, InferSharp, die C# ondersteunt. Infer richt zich voornamelijk op problemen op het gebied van *memory* en *concurrency*, die vaak een veiligheidsrisico vormen. Daarnaast kan Infer ook gebruikt worden om bijvoorbeeld te controleren of de gescande code aan bepaalde stijlregels voldoet. Ook Infer ontleent nieuwigheid aan een oorsprong in academisch onderzoek. Infer heeft ongeveer 15.000 sterren op GitHub en is transparant, want het is een open source.

Semgrep OSS. Net als CodeQL, analyseert Semgrep broncode ten opzichte van regels die gebruikers zelf kunnen opstellen. Ook bij Semgrep bestaat er een database van standaardregels waarmee Semgrep als een volledige geautomatiseerde SAST-tool gebruikt kan worden. Dit onderzoek beperkt zich tot de gratis open source variant van Semgrep, Semgrep OSS. Er bestaat ook een betaalde variant met extra functionaliteit.

De regels van Semgrep worden geschreven in een YAML-syntax en maken veel gebruik van patroonherkenning, vergelijkbaar met de bekende tool grep. De patroonherkenning wordt bij Semgrep echter verfijnd door middel van semantische informatie over het programma. Deze manier van werken is relatief gebruikersvriendelijk. Een ander voordeel is dat de code niet gecompileerd hoeft te worden voor gebruik.

Het bedrijf Semgrep is opgericht in 2017 onder de naam r2c. De tool Semgrep is een opvolger van de tool sgrep, ontwikkeld door Facebook in 2009.

Semgrep ondersteunt meer dan 35 programmeertalen, waaronder de gehele top 5 van de TIOBE-index. Ook Semgrep richt zich vooral op kwetsbaarheden met security impact, maar bevat daarnaast regels die meer algemene codekwaliteit bewaken. Semgrep is nieuw, omdat het in z'n huidige vorm nog geen 10 jaar bestaat. Het is daarnaast innovatief, omdat het (net als CodeQL) gebruik maakt van externe regels. Semgrep heeft ongeveer 11.000 sterren op GitHub. De *engine* en een groot deel van de regels zijn open source. Een aantal geavanceerde regels is afgeschermd en alleen te gebruiken tegen betaling.

Hiermee kan de tweede deelvraag worden beantwoord:

DV1.2 Op welke theoretische inzichten zijn de nieuwe tools gebaseerd?

CodeQL: het zien van code als een database en het gebruik van een domein-specifieke object-georiënteerde *query language* gebaseerd op Datalog.

Infer: separation logic en bi-abductie.

Semgrep: het verfijnen van grep-achtige patroonherkenning met semantische informatie.

[17] https://codeql.github.com/docs/codeql-overview/supported-languages-and-frameworks/
[18] Calcagno et al. 2009.
[19] Reynolds 2002.
[20] Berdine et al. 2006.
[21] Calcagno et al. 2009
[22] https://results2021.ref.ac.uk/impact/39476dc8-8346-4bcd-a932-e70060d472ca?Page=1

DV1.3: Vergelijking

Om de volgende onderzoeksvraag te beantwoorden worden de drie geselecteerde tools vergeleken met elkaar en met een meer gevestigde tool. De keuze voor de gevestigde tool is gevallen op SonarQube, aangezien uit de uitgevoerde enquête blijkt dat dit de meest gebruikte SAST-tool is in Nederland (zie figuur 3). De meeste artikelen in de literatuur vergelijken niet alle vier de tools, maar slechts een deel daarvan. Daarom volgen hieronder eerst paarsgewijze vergelijkingen.

Eerst volgen ter opfrissing de definities van een aantal termen uit de statistiek. Meldingen van SAST-tools zijn *echt positieven* als ze terecht zijn, en *foutpositieven* als ze onterecht zijn. De afwezigheid van een melding terwijl er wel degelijk een probleem is heet een *foutnegatieve*. Is er geen melding, en ook geen probleem? Dan spreek je van een *echt negatieve*. De *foutpositievenratio* is het aantal foutpositieven gedeeld door de som van de echt negatieven en de foutpositieven. In andere woorden: de foutpositievenratio is het percentage onschuldige code dat ten onrechte als probleem is gemarkeerd. De *echtpositievenratio*, *foutnegatievenratio*, en *echtnegatievenratio* hebben vergelijkbare definities. Tot slot maken sommige artikelen gebruik van de zogenaamde F1-score. Daarvoor definieert men eerst de *precisie* als het aantal echt positieven gedeeld door de som van het aantal echt positieven en het aantal foutpositieven. De F1-score is dan het harmonisch gemiddelde van de precisie en de echtpositievenratio.

SonarQube vs. CodeQL. Een studie uit 2023 van Li et al. vergelijkt zeven SAST-tools, waaronder SonarQube en CodeQL, door die toe te passen op Java-programma's. Op de synthetische OWASP-benchmark behaalt SonarQube een F1-score van 27%, waar die van CodeQL 49% is. De auteurs testen dezelfde tools ook op een door hen zelf samengestelde benchmark bestaande uit echte kwetsbaarheden in opensourceprogramma's. Op die benchmark zijn SonarQube en CodeQL ongeveer even accuraat. Dezelfde studie vergelijkt de tools ook op snelheid, en vindt dat SonarQube sneller is dan CodeQL, met name bij grote programma's (meer dan 50k regels code).

Een andere studie uit 2024 van Li et al. vergelijkt ook zeven SAST-tools, maar dan op C++-programma's. Ook daar presteert CodeQL significant beter dan SonarQube op een synthetische benchmark (de Juliet Test Suite). In deze studie wordt CodeQL ook op een echte dataset getest, maar SonarQube niet.

SonarQube vs. Infer. Deze tools worden in verschillende onderzoeken vergeleken, te beginnen met een studie uit 2023 van Liu et al. Die studie richt zich op Javaprogramma's en vergelijkt SonarQube en Infer (en nog drie andere tools) qua reikwijdte van regels, accuraatheid en snelheid. De studie concludeert dat SonarQube een grotere reikwijdte heeft dan Infer, en dus ook meer CWE's detecteert. Echter, binnen de klassen van kwetsbaarheden die Infer detecteert, is Infer accurater.

Een andere studie, van Alqaradaghi & Kozsik uit 2022, vergelijkt SonarQube en Infer specifiek wat betreft de detectie van *null pointer dereferences* in Javaprogramma's. Het resultaat is dat de echtpositievenratio van Infer marginaal hoger is dan die van SonarQube, en de foutpositievenratio van SonarQube substantieel hoger dan die van Infer.

Tot slot is er een masterscriptie van Ablasser uit 2019. Deze neemt de echtpositievenratio en foutpositievenratio in beschouwing en concludeert dat Infer accurater is dan SonarQube op zowel programma's geschreven in C++ als op programma's geschreven in Java.

SonarQube vs. Semgrep OSS. Bij de eerdergenoemde studie van Li et al. uit 2023 wordt ook Semgrep meegenomen. Op de synthetische OWASP-benchmark is de F1-score van Semgrep 80% (en die van SonarQube 27%). Op de zelf samengestelde benchmark met echte programma's is Semgrep ongeveer even accuraat als SonarQube. Semgrep is iets minder snel dan SonarQube. Interessant genoeg wordt de snelheid van Semgrep nauwelijks beïnvloed door de grootte van het gescande project; dat komt omdat Semgrep verschillende delen van het project parallel scant.

De eerdergenoemde studie van Liu et al. uit 2023 bekijkt ook Semgrep. De reikwijdte van Semgrep is minder groot dan die van SonarQube (18 CWE-categorieën vs. 28 CWE-categorieën). De accuraatheid van Semgrep blijkt in deze studie een stuk minder groot dan die van SonarQube. Dat is in ieder geval gedeeltelijke te verklaren door het feit dat Semgrep sommige klassen van kwetsbaarheden niet ondersteunt. Deze studie bevestigt dat Semgrep minder snel is dan SonarQube en dat de snelheid van Semgrep nauwelijks beïnvloed wordt door de grootte van het gescande project. Een derde studie die zowel SonarQube als Semgrep heeft getest is de eerdergenoemde studie van Li et al. uit 2024. Op de synthetische Juliet Test Suite is Semgrep significant accurater dan SonarQube.

CodeQL vs. Infer. De enige gevonden studie die zowel CodeQL als Infer test is die van Mantovani et al. uit 2022. Op een beperkte dataset van echte programma's, slaagt CodeQL erin om 9 van de 10 kwetsbaarheden te detecteren, waar Infer er slechts 2 van de 10 detecteert. Het moet hierbij wel worden benoemd dat de auteurs eigen CodeQL *queries* hebben toegevoegd die op maat zijn gemaakt voor de gevonden kwetsbaarheden.

CodeQL vs. Semgrep OSS. De twee studies die zowel CodeQL als SonarQube bekijken, bekijken ook Semgrep.

Bij de studie van Li et al. uit 2023 is op de synthetische benchmark de F1-score van CodeQL 49% en die van Semgrep 80%. Op de benchmark met echte programma's zijn SonarQube en Semgrep ongeveer even accuraat. CodeQL is sneller op relatief kleine programma's (< 50k coderegels), en Semgrep is sneller op relatief grote programma's.

Uit studie van Li et al. uit 2024 blijkt dat CodeQL accurater is dan Semgrep op de synthetische Juliet Test Suite.

Er is nog een niet eerdergenoemde studie die zowel CodeQL en Semgrep vergelijkt, namelijk die van Bennett et al. uit 2024. Deze studie test 8 tools op de SAP-database, die bestaat uit handmatig geselecteerde open source Java-programma's met bekende kwetsbaarheden. CodeQL en Semgrep detecteren ongeveer evenveel van deze kwetsbaarheden (beide zo'n 15%). Door op maat gemaakte regels toe te voegen aan Semgrep, slaagden de auteurs erin de detectiegraad van Semgrep op deze dataset te verhogen tot 44,7%. Het is daarbij natuurlijk wel de vraag of die regels ook breder toepasbaar zijn dan alleen op deze dataset.

Infer vs. Semgrep OSS. Bij de enige gevonden studie die beide tools bekijkt, die van Liu et al. uit 2023, detecteren zowel Infer als Semgrep zeer weinig kwetsbaarheden. Dit komt waarschijnlijk doordat de kwetsbaarheden in de benchmarks die bij deze studie gebruikt worden vooral kwaliteit gerelateerd zijn in plaats van specifiek security gerelateerd. Qua reikwijdte scoren Infer en Semgrep ongeveer even hoog. Deze studie observeert dat Infer sneller is dan Semgrep, behalve bij zeer grote projecten (> 100k coderegels).

Hiermee kan de derde deelvraag worden beantwoord:

DV1.3 Hoe verhouden deze tools zich tot elkaar en tot meer gevestigde tools?

- Infer is accurater dan SonarQube, maar op een nauwere selectie aan CWE's.
- Wat betreft kwetsbaarheden die een veiligheidsrisico vormen, zijn Semgrep OSS en CodeQL accurater dan SonarQube.
- SonarQube is accurater dan de nieuwe tools als het gaat om algemene problemen met codekwaliteit.
- CodeQL is over het algemeen iets accurater dan Semgrep OSS.
- SonarQube is sneller dan zowel CodeQL als Semgrep OSS.
- Op relatief kleine projecten zijn CodeQL en Infer sneller dan Semgrep OSS, maar op grotere projecten geldt het omgekeerde.

DV2.1: Ervaringen ontwikkelaars

Er is behoorlijk veel onderzoek gedaan naar de ervaringen van ontwikkelaars bij het gebruik van SAST-tools. In de afgelopen jaren zijn er twee artikelen verschenen die door middel van systematisch literatuuronderzoek een overzicht geven van deze resultaten.

Nachtigall et al. (2022) destilleert uit de literatuur de volgende zes bruikbaarheidscriteria. In het artikel wordt elk criterium nog verder onderverdeeld in verschillende subcriteria.

1. Meldingen: een SAST-tool moet informatieve meldingen geven op basis waarvan een gebruiker actie kan ondernemen.
2. Oplossingshulp: het blijkt uit de literatuur dat ontwikkelaars graag hulp van een SAST-tool krijgen bij het oplossen van een gesignaleerd probleem.
3. Foutpositieven: één van de meest belangrijke obstakels bij het gebruik van SAST-tools is het hoge aantal gerapporteerde foutpositieven. Een SAST-tool moet dit op een bepaalde manier mitigeren. Bijvoorbeeld door een bepaalde betrouwbaarheid score toe te kennen aan een melding, of door gebruikers foutpositieven handmatig te laten onderdrukken.
4. Gebruikersfeedback: SAST-tools moeten gebruikers de mogelijkheid bieden om feedback te geven aan de tool. Dit kan bijvoorbeeld door de regels aanpasbaar te maken, of door gebruikers meldingen te laten filteren.
5. Workflow integratie: SAST-tools moeten makkelijk te integreren zijn in de workflow van de gebruiker. Afhankelijk van de specifieke gebruiker kan dit op meerdere manieren: in de IDE, in de CI/CD-pijplijn, of als standalone tool. Idealiter ondersteunt een tool meerdere workflows.
6. Interface: een SAST-tool moet een interface hebben die goed gebruik stimuleert. Bijvoorbeeld door coderegels waarvoor een melding geldt op te laten lichten in de IDE. Of door een overzicht te geven van welke meldingen al zijn opgelost en welke nog open staan.

De studie van Nachtigall et al. evalueert 46 tools aan de hand van deze criteria, waaronder Semgrep OSS en InferSharp (een variant van Infer). CodeQL en Infer worden niet meegenomen in de vergelijking; tot CodeQL konden de auteurs geen toegang krijgen, en bij Infer hadden de auteurs problemen met installeren. Om een eerlijke vergelijking te maken worden in dit rapport voor de beantwoording van DV2.2 alle drie de tools opnieuw tegen het licht gehouden.

De andere recente studie die een overzicht geeft van gebruikerservaringen met SAST-tools is Wadhams et al. (2024). Via een systematisch literatuuronderzoek, waarin 89 relevante artikelen zijn bestudeerd, identificeert deze studie vijf obstakels bij het gebruik van SAST-tools. Deze vijf obstakels zijn, op volgorde van meest tot minst voorkomend in de literatuur:

1. Veel foutpositieven
2. Slechte presentatie van output
3. Tijdrovend om op te zetten
4. Te weinig hulp bij het oplossen van meldingen
5. Slechte workflow integratie

Daarnaast noemt de studie ook het gebrek aan aanpasbaarheid van SAST-tools als een belangrijk obstakel.

Door de bevindingen van de twee genoemde overzichtsartikelen met elkaar te combineren vormt zich het volgende antwoord op DV2.1:

DV2.1 Wat zijn de belangrijkste obstakels die ontwikkelaars ervaren bij het gebruik van ASAT's?

1. Te veel foutpositieven
2. Onvoldoende informatieve meldingen
3. Te weinig hulp bij het oplossen van meldingen
4. Onvoldoende workflow integratie
5. Gebrek aan incorporatie van gebruikersfeedback

DV2.2: Adressering obstakels door nieuwe tools

De vijf geïdentificeerde obstakels worden één voor één besproken. Elk obstakel wordt opgedeeld in verschillende subobstakels, en bij elk subobstakel wordt per SAST-tool handmatig een inschatting gemaakt of die het subobstakel niet, gedeeltelijk, of helemaal adresseert.

Foutpositieven

Percentage foutpositieven

Uit verschillende van de eerder besproken onderzoeken blijkt dat de nieuwe tools accurater zijn dan het gevestigde SonarQube. Dat betekent echter nog niet noodzakelijk dat die tools ook minder foutpositieven opleveren.

In Ami et. al (2024) wordt door middel van interviews met 20 gebruikers onderzocht hoe zij SAST-tools ervaren. Daarbij komt ook de vraag aan bod welk percentage van foutpositieven deze gebruikers kunnen tolereren. De studie concludeert dat, waar de in de literatuur vaak een grens van 20% wordt genoemd, de door hun geïnterviewde gebruikers ook een hoger aantal zouden tolereren, soms tot wel 80%.

De studie Shen et al. (2023) past CodeQL toe op verschillende open source embedded softwareprojecten. Uit een door de auteurs uitgevoerde handmatige analyse blijkt dat 23% foutpositieven oplevert. De eerdergenoemde Li et al. (2023) en Li et al. (2024) vinden een foutpositievenratio van respectievelijk 60% (op de OWASP-benchmark) en 23% (op de Juliet Test Suite). Met een gemiddeld percentage foutpositieven van ongeveer 35%, kan worden gezegd dat CodeQL dit obstakel **gedeeltelijk** adresseert.

In Kharkar (2022) wordt door middel van machine learning het aantal foutpositieven van Infer gereduceerd. Het meest succesvolle model slaagt erin om dit terug te brengen tot 14,6%. Zonder enige toevoegingen is het percentage foutpositieven van Infer volgens deze studie al 27,3%. Dit is voldoende om te concluderen dat Infer dit obstakel **geheel** adresseert.

Bij Li et al. (2023) en Li et al. (2024) is het percentage foutpositieven van Semgrep respectievelijk 30% en 68%, met een gemiddeld percentage van 49% kan worden gezegd dat Semgrep dit obstakel **gedeeltelijk** addreseert.

Betrouwbaarheidsscore

Uit eerder onderzoek blijkt dat gebruikers beter met foutpositieven om kunnen gaan als een SAST-tool een betrouwbaarheidsscore toekent aan waarschuwingen. Dit doet CodeQL **niet,** Infer ook **niet,** en Semgrep **geheel**.

Informatieve meldingen

Transparante redeneringen

Een melding is beter te begrijpen als het mogelijk is om te achterhalen op basis van welke redenering een SAST-tool die melding geeft. Bij Infer is dit bij de meeste meldingen **niet** het geval. Voor Semgrep en CodeQL geldt dat elke melding gemaakt wordt op basis van een publiek beschikbare regel. Aangezien die regels bij CodeQL soms moeilijk te begrijpen zijn, is de inschatting dat CodeQL dit obstakel **gedeeltelijk** adresseert. De regels van Semgrep zijn beter te begrijpen, dus die tool adresseert dit obstakel **geheel.**

Pad omschrijving

Een significant deel van de kwetsbaarheden ontstaat doordat onbetrouwbare input (ook wel de *source* genoemd) een weg maakt door het programma naar een kwetsbare functie (die noemt men de *sink*). Een melding van dit soort kwetsbaarheden is informatiever op het moment dat had pad van de source naar de sink expliciet wordt gemaakt. Alle drie de beschouwde tools doen dit **geheel**.

Extra informatie (bijvoorbeeld via links)

Een melding wordt ook informatiever als de gebruiker extra informatie kan krijgen, bijvoorbeeld via een link. Bij alle drie de beschouwde tools is er online uitgebreide informatie te vinden over alle specifieke meldingen. Ze adresseren dit obstakel daarmee **geheel**.

Ernst

Een andere manier waarop een melding informatief kan zijn, is door de ernst te classificeren. Hiermee kunnen gebruikers meldingen prioriteren. Zowel CodeQL als Semgrep doen dit **geheel**, terwijl Infer het **niet** doet.

Hulp bij het oplossen van meldingen

Autofix
Idealiter bieden tools de mogelijkheid om meldingen automatisch te verhelpen. Dit vergroot namelijk de kans dat ontwikkelaars ook daadwerkelijk iets met de meldingen doen. CodeQL maakt het sinds kort mogelijk om meldingen automatisch te verhelpen door middel van de AI-assistent Copilot. Daarmee adresseert CodeQL dit obstakel **geheel**. Bij Semgrep is het mogelijk om een *fix* toe te voegen aan een regel. Aangezien lang niet alle regels van deze mogelijkheid gebruik maken, adresseert Semgrep dit obstakel **gedeeltelijk**. Infer, tot slot, biedt geen *autofix* voorziening en adresseert dit obstakel daarom **niet.**

Workflow integratie

Prioritering
Het helpt als de meldingen op een geprioriteerde manier aan de gebruiker worden getoond. Bij CodeQL is dit het geval in de webinterface. De CLI-applicatie kan de resultaten in verschillende formaten uitdraaien, waaronder het SARIF-formaat, dat prioritering faciliteert. De IDE-versie van CodeQL is meer bedoeld om individuele queries te draaien en te testen, waarbij prioritering minder relevant is. Al met al is het oordeel dat CodeQL dit obstakel **geheel** adresseert. Semgrep OSS en Infer adresseren dit obstakel **niet**.

IDE-integratie
Vrijwel alle ontwikkelaars schrijven hun code in een IDE. Het is daarom heel nuttig als een SAST-tool in de IDE geïntegreerd kan worden. Dit geldt voor alle drie de tools **geheel**.

CLI
Naast gebruik in de IDE, is het nuttig als een SAST-tool de mogelijkheid biedt om vanuit de command line uitgevoerd te worden. Hiermee worden dingen als integratie met andere tools en gebruik op afstand makkelijker om te implementeren. Alle drie de tools ondersteunen gebruik in de command line **geheel**.

Snelheid
Om de workflow van een ontwikkelaar zo min mogelijk te storen, is het van belang dat een tool niet te langzaam is. Op basis van persoonlijke ervaring van de auteur en de literatuur is de bevinding dat CodeQL dit obstakel **niet** adresseert, Infer **geheel** en Semgrep **gedeeltelijk**.

Gebruikersfeedback

Aanpasbare regels
Aanpasbare regels bieden verschillende voordelen. Het stelt gebruikers bijvoorbeeld in staat om een tool te kalibreren naar hun codebase en regels te gebruiken uit de gemeenschap. CodeQL en Semgrep ondersteunen dit **geheel**. Bij Infer is het ook mogelijk om zogeheten checkers te schrijven, maar dat is minder makkelijk en ook minder gebruikelijk. Daarom ondersteunt Infer dit **gedeeltelijk**.

Gebruikersfeedback echt positieven
SAST-tools kunnen hun accuraatheid verbeteren door te leren van gebruikersfeedback over echt positieven. De drie bekeken tools ondersteunen dit allen **niet**.

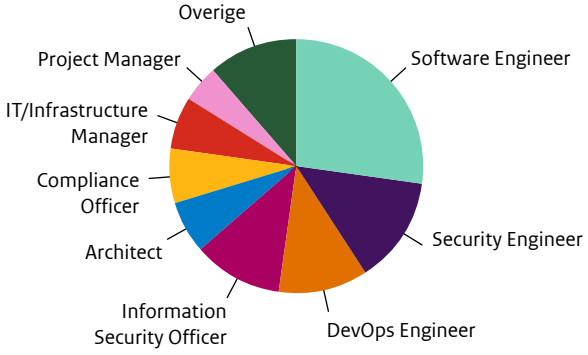
Onderdrukking
Het is belangrijk dat gebruikers bepaalde meldingen kunnen onderdrukken, zodat ze zich kunnen richten op meldingen die belangrijker zijn. Alle drie de tools ondersteunen dit **geheel**.

Filtering
Het kan nuttig zijn om meldingen te kunnen filteren op bijvoorbeeld het type kwetsbaarheid, of de ernst van een melding. CodeQL en Infer ondersteunen dit **geheel** en Semgrep ondersteunt dit **niet**.

DV2.2 In hoeverre worden deze obstakels geadresseerd door de nieuwe tools?

Zie tabel 1 voor een volledig overzicht van de handmatige inschatting.

Figuur 1 *functies respondenten*



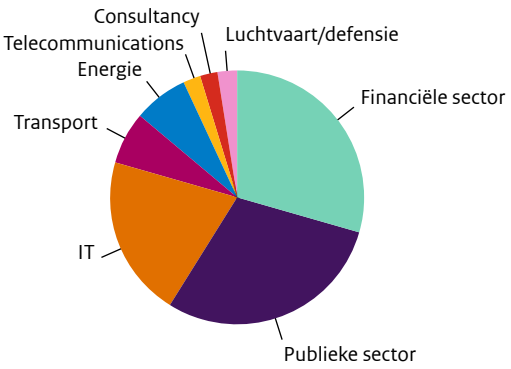
Tabel 1 *overzicht van de (deel)obstakels en de inschatting van de mate waarin de geselecteerde tools die adresseren: niet (-), gedeeltelijk (±) of geheel (+). C staat voor CodeQL, I staat for Infer, en S staat voor Semgrep OSS.*

	C	I	S
Vals-positieven			
Percentage vals-positieven	±	+	±
Betrouwbaarheidsscore	-	-	+
Informatieve meldingen			
Transparante redeneringen	+	±	+
Pad omschrijving	+	+	+
Extra informatie (bijvoorbeeld via links)	+	+	+
Ernst	+	-	+
Hulp bij het oplossen van meldingen			
Autofix	+	-	±
Workflow integratie			
Prioritering	+	-	-
IDE-integratie	+	+	+
CLI	+	+	+
Snelheid	-	+	±
Gebruikersfeedback			
Aanpasbare regels	+	±	+
Gebruikersfeedback echtpositieven	-	-	-
Onderdrukking	+	+	+
Filtering	+	+	-

DV3.1 & DV3.2: Enquêteresultaten

Om de derde deelvraag te beantwoorden is er via verschillende kanalen een enquête uitgezet bij de doelgroepen van het NCSC. Er zijn 44 respondenten. In Figuur 1 en Figuur 2 is te zien hoe de respondenten verdeeld zijn over functies en sectoren. Ongeveer de helft van de respondenten werkt als softwareontwikkelaar. De rest heeft een rol met verantwoordelijkheid over het proces, zoals Information Security Officer. De best gepresenteerde sectoren zijn de publieke sector, de financiële sector, en de IT-sector. Iets meer dan driekwart van de respondenten werkt in één van die sectoren.

Figuur 2 *sectoren respondenten*



De respondenten is gevraagd met welke SAST-tools ze ervaring hebben, en welke ze tegenkomen in hun huidige werk. Zie figuren 3 en 4 voor een volledig overzicht van de antwoorden.

Wat betreft gebruik van de nieuwe tools, blijkt dat van de 44 respondenten er 0 ervaring hebben met Infer, 4 met CodeQL (waarvan 1 in de huidige functie) en 5 met Semgrep (waarvan 4 in de huidige functie). Ter vergelijking: 32 respondenten hebben ervaring met SonarQube (waarvan 25 in de huidige functie).

Het is ook interessant om deze resultaten te vergelijken met een recent onderzoek van Bennet et al. uit 2024. Het aandeel van hun respondenten dat SonarQube gebruikt (59%) is ongeveer even groot als dat van ons (57%), terwijl hun aandeel CodeQL-gebruikers (25%) en Semgrep-gebruikers (17%) een stuk groter zijn dan dat van ons (resp. 2% en 9%). Een ander interessant verschil is dat er onder onze respondenten relatief veel gebruikers van Checkmarx en Fortify zijn, terwijl die tools bij Bennet et al. buiten de top 7 vallen.

Op basis hiervan kan de eerste deelvraag van vraag 3 worden beantwoord:

DV3.1 Gebruiken Nederlandse organisaties de nieuwe tools?

Op basis van de enquête kan geconcludeerd worden dat de nieuwe tools nog **zeer weinig** worden gebruikt bij Nederlandse organisaties. Dat staat in contrast met de bevindingen van Benett et al. (2024), die onder wereldwijde respondenten een groter aantal CodeQL- en Semgrep-gebruikers aantreffen.

De volgende deelvraag gaat over welke tools geschikt zijn voor welke organisaties. Er valt een aantal observaties te maken.

Voor de meeste organisaties is de belangrijkste reden om SAST-tools te gebruiken het detecteren van security bugs.

Zie figuur 5 voor een overzicht van waar de respondenten de SAST-tools voor gebruiken. Daaruit blijkt dat het vinden van security bugs de belangrijkste reden is om SAST-tools te gebruiken. Aan de ene kant is dit opvallend, omdat SAST-tools vaak ook meer functionaliteit bieden, zoals stimuleren van een uniforme codestijl. Aan de andere kant is het begrijpelijk dat ontwikkelaars kwetsbaarheden in hun code nóg liever willen voorkomen dan andere kwaliteitsproblemen. In figuur 6 is te zien dat ditzelfde beeld ook binnen specifieke sectoren geldt.

Bij de meeste sectoren, maar niet bij de publieke sector, zijn SAST-tools een belangrijker middel om security bugs te detecteren dan handmatige codereview en pentests.

Zie figuur 6. Van bijvoorbeeld de IT-sector, vindt 100% het gebruik van SAST-tools belangrijk of zeer belangrijk voor het detecteren van security bugs, terwijl dat respectievelijk 89% en 67% is voor handmatige codereview en pentests. Dit bevestigt dat SAST een zeer belangrijk middel is voor organisaties om kwetsbaarheden te detecteren.

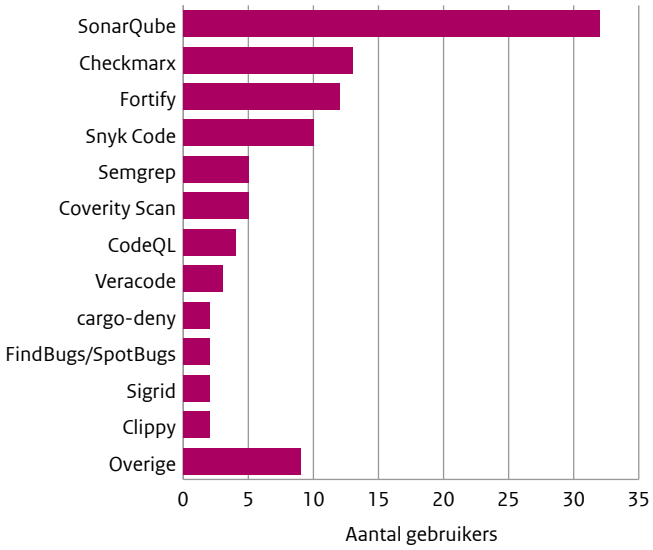
De respondenten uit de IT-sector en de financiële sector ervaren minder obstakels bij het gebruik van SAST-tools dan die uit de publieke sector en de overige sectoren.

Zie figuur 8. Met name bij workflow integratie is het percentage respondenten dat dat een aanzienlijk obstakel vindt, of vindt dat dat SAST-tools onbruikbaar maakt, in de financiële sector (38%) en de IT-sector (44%) veel kleiner dan in de publieke sector (77%) en de overige sectoren (78%). Dit zou kunnen betekenen dat de publieke sector van de andere sectoren kan leren op het gebied van SAST-tools.

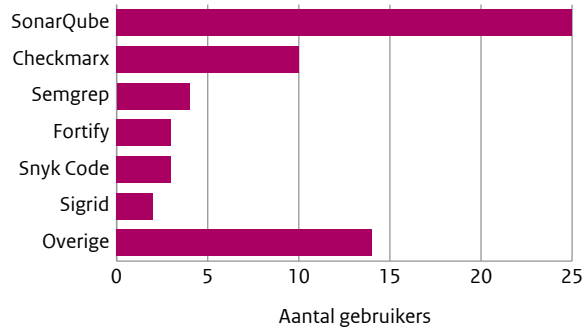
DV3.2: Welke tools zijn geschikt voor welke organisaties?

Aangezien Infer een kleinere klasse van kwetsbaarheden detecteert, valt het hoe dan ook aan te raden om naast Infer ook een andere SAST-tool te gebruiken. Met het oog op workflow integratie is het wellicht te overwegen om CodeQL te gebruiken wanneer je code op GitHub is gehost, en Semgrep te gebruiken wanner je code op GitLab is gehost.

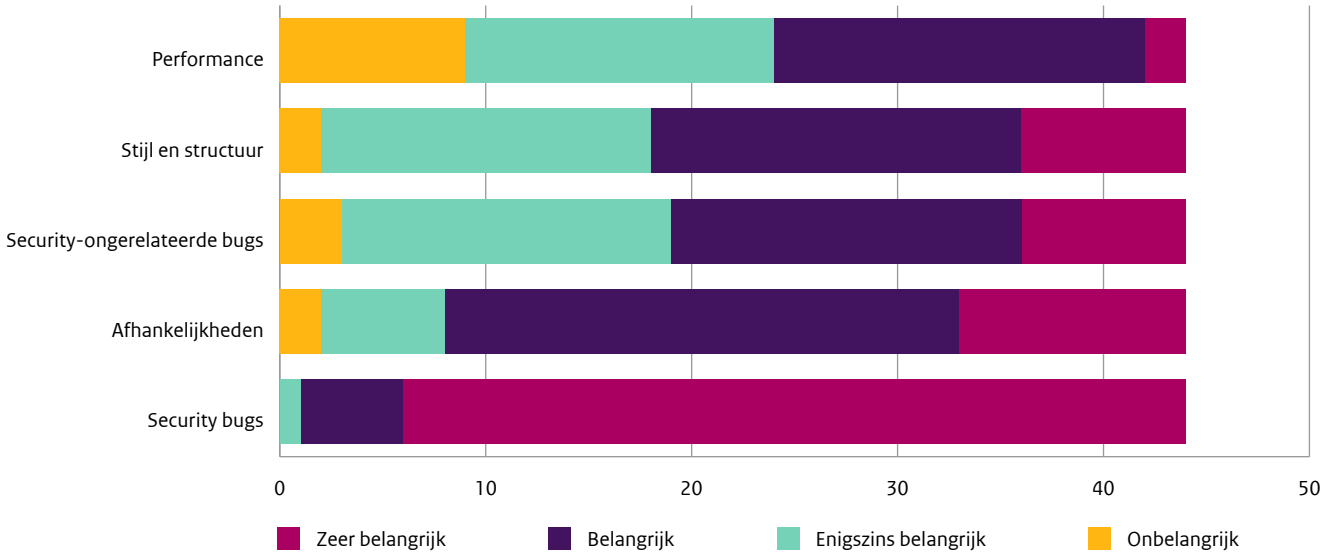
Figuur 3 *ervaringen respondenten met verscheidene SAST-tools*



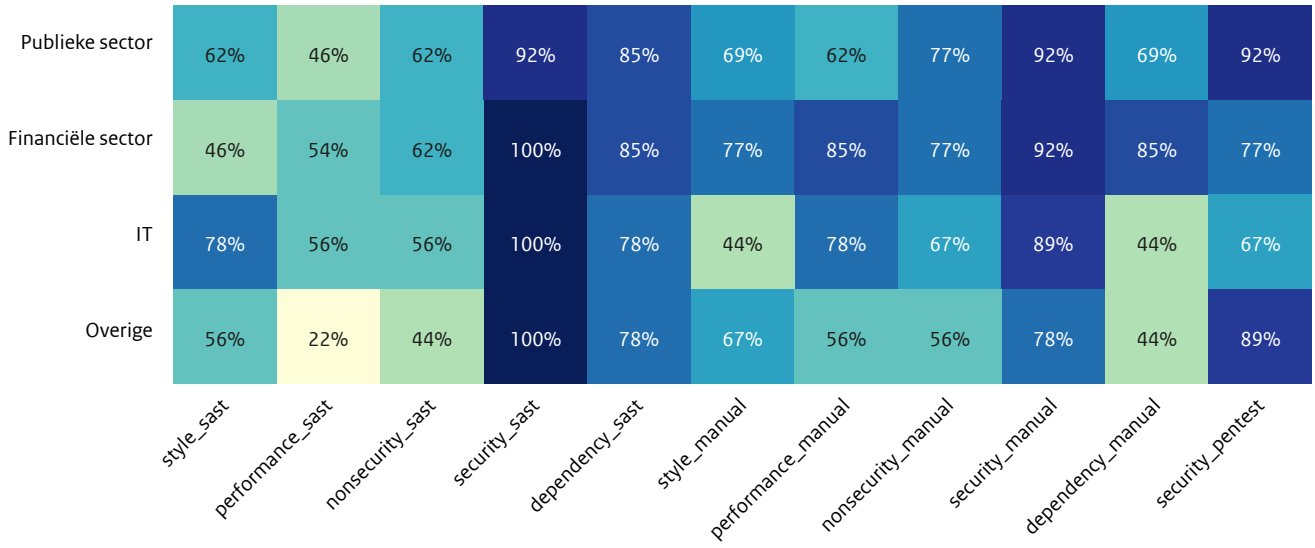
Figuur 4 *ervaringen respondenten met verscheidene SAST-tools in huidige functie*



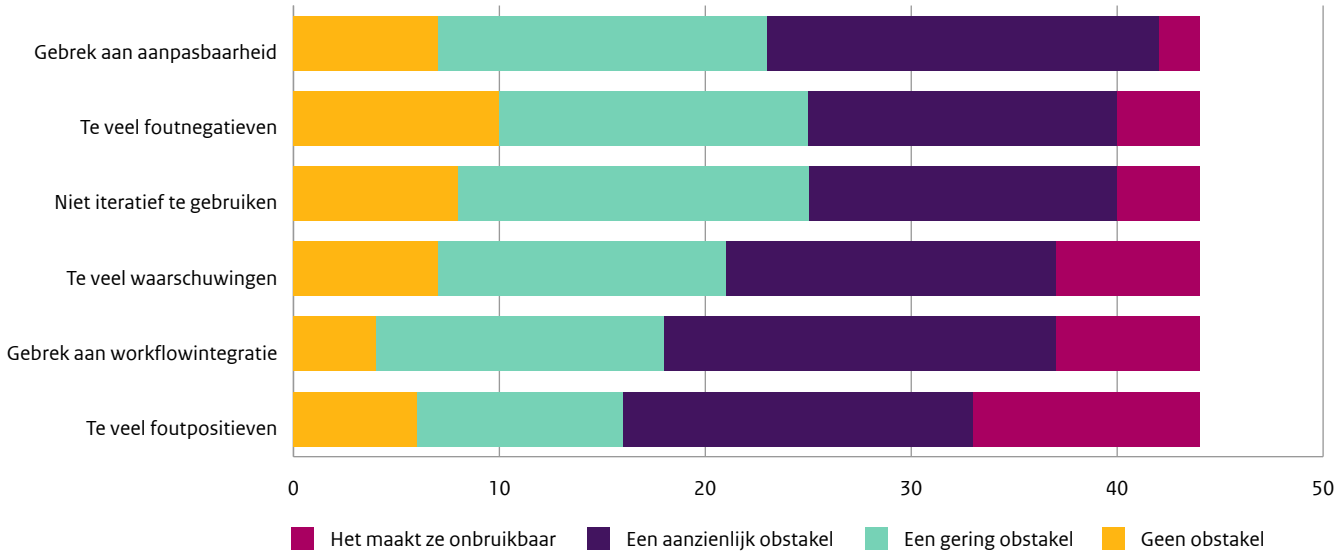
Figuur 5 *belang gebruik van SAST-tools door respondenten voor verschillende doeleinden*



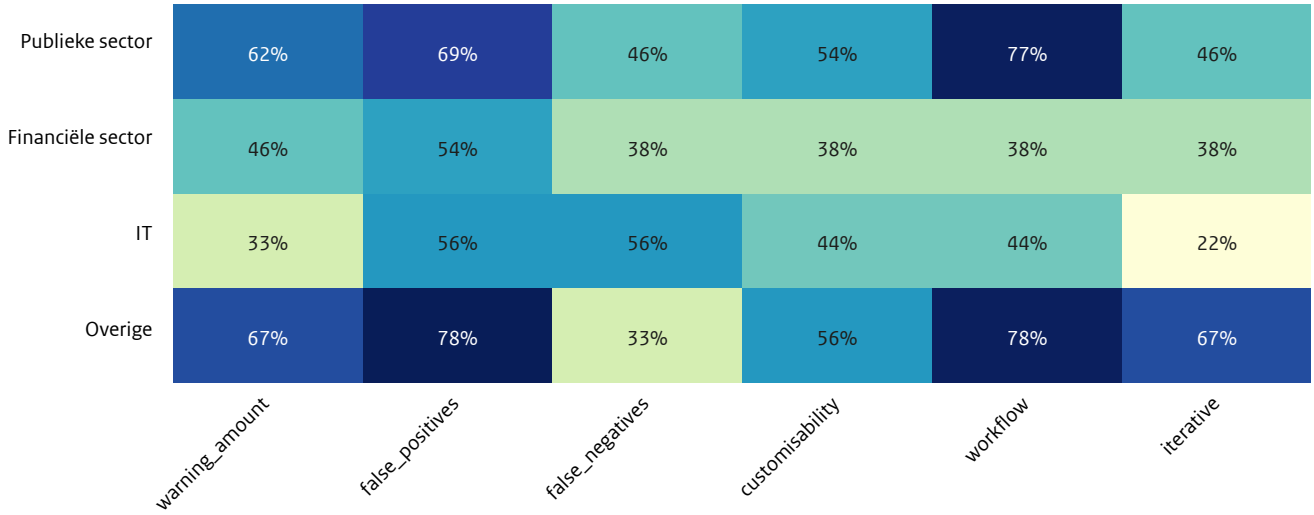
Figuur 6 *voor zowel SAST als handmatige codereviews hebben de respondenten aangegeven of ze dat niet belangrijk, enigszins belangrijk, belangrijk of zeer belangrijk vinden voor het detecteren van problemen op het gebied van performance, security-ongerelateerde bugs, security bugs, afhankelijkheden, en stijl en structuur. Naar het belang van penetration tests is alleen gevraagd op het gebied van security bugs. Deze heatmap geeft per sector het percentage van respondenten dat een onderdeel óf belangrijk, óf zeer belangrijk vindt.*



Figuur 7 *obstakels bij het gebruik van SAST-tools*



Figuur 8 *heatmap van percentage minstens aanzienlijk obstakel per sector en onderdeel*



Conclusies en adviezen

In dit hoofdstuk worden een aantal concrete adviezen gegeven voor hoe Nederlandse organisaties nieuwe SAST-tools tot hun voordeel kunnen aanwenden.

Gebruik SAST-tools
SAST-tools zijn volgens de respondenten van de enquête een zeer belangrijk middel om kwetsbaarheden te detecteren. Het is dan ook alle organisaties aan te raden om SAST-tools te gebruiken.

Experimenteer met nieuwe tools
Nieuwe tools lijken op verschillende vlakken beter te presteren dan gevestigde tools. Voor elke van de drie in dit rapport besproken nieuwe tools geldt dat ze gratis uit te proberen zijn. Het is voor Nederlandse organisaties de moeite waard om met nieuwe tools te experimenteren om te kijken of ze ook in hun specifieke situatie een verbetering vormen ten opzichte van de status quo. Dat hoeft natuurlijk niet beperkt te blijven tot de tools in dit rapport. De ontwikkelingen gaan snel, en voor sommige organisaties zijn gespecialiseerde tools geschikter.

Gebruik meerdere tools
Uit verschillende onderzoeken, waaronder Bennet et al. (2024), blijft dat SAST-tools elkaar aanvullen: de één detecteert fouten die de andere niet ziet en vice versa. Voor optimale dekking valt het dus aan te raden om meerdere tools tegelijkertijd te gebruiken.

Schrijf eigen regels, of configureer SAST-tools naar de eigen organisatie
Twee nieuwe SAST-tools, CodeQL en Semgrep, onderscheiden zich door aanpasbare regels te gebruiken. Uit verschillende onderzoeken blijkt dat deze tools nog effectiever zijn als de regels aangepast zijn aan de organisatie waar ze worden toegepast. Het is de moeite waard om binnen een organisatie kennis op te bouwen over hoe deze regels worden geschreven. Ook bij SAST-tools die geen aanpasbare regels ondersteunen is het verstandig om te kijken hoe die optimaal geconfigureerd kunnen worden naar de eigen organisatie.

Kennisuitwisseling tussen organisaties uit verschillende Nederlandse sectoren kan het gebruik van SAST-tools verbeteren
Uit de enquêteresultaten blijkt dat respondenten uit de IT-sector en de financiële sector minder obstakels ervaren bij het gebruik van SAST-tools dan die uit de publieke sector. Het zou goed zijn als er meer kennisuitwisseling zou zijn, zodat bijvoorbeeld de publieke sector op dit gebied van de andere sectoren kan leren. Het zou ook interessant zijn om beter te begrijpen waarom de publieke sector de enige sector is waarin pentests en handmatige codereview even belangrijk zijn voor het detecteren van kwetsbaarheden als SAST-tools.

Discussie en vervolg

Een belangrijk gebrek van dit onderzoek is dat de enquête waarschijnlijk slechts is ingevuld door mensen die met SAST-tools werken. Dit zou een vertekend beeld kunnen geven. In een vervolgstudie zou het daarom interessant zijn om ook expliciet softwareontwikkelaars te bevragen die geen SAST-tools gebruiken.

Het is ook belangrijk om te vermelden dat de kosten van de vergeleken SAST-tools niet in beschouwing genomen zijn. In de praktijk spelen die natuurlijk wel een rol bij de beslissing van een organisatie om een SAST-tool te gebruiken. Het Amerikaanse agentschap CISA heeft recentelijk gesignaleerd dat er onvoldoende empirische data is over de kosteneffectiviteit van de best practices op het gebied van veilige softwareontwikkeling.[23]

Een andere omissie van dit onderzoek is dat het zich specifiek op statische analyse richt, en weinig vergelijkingen maakt met andere manieren om software te testen. Bij de enquête is respondenten wel gevraagd naar hun ervaringen met handmatig testen en pentesten (zie figuur 6), maar specifiek niet naar dynamische analyse. Bij vervolgonderzoek zou het nuttig zijn om ook het gebruik van dynamische analyse onder deze doelgroep in kaart te brengen.

Van de 44 respondenten hebben er 20 hun mailadres achtergelaten, omdat ze open staan om aan een vervolgstudie deel te nemen. Deze vervolgstudie zou zich kunnen richten op de vooralsnog onbeantwoorde DV 3.3: welke problemen kunnen ontstaan bij de implementatie van deze tools? Deze studie zou de vorm kunnen hebben van interviews, maar ook bijvoorbeeld een *case study* waarin nieuwe tools bij een organisatie worden geïmplementeerd.

[23] https://www.cisa.gov/sites/default/files/2024-10/CSAC_October-Quarterly-Meeting_SBD-Recommendations_20241011_508.pdf

Literatuur

Ablasser, M. (2019). *Effectiveness of Verification Tools* [Master's thesis, TU Graz].

Alqaradaghi, M., & Kozsik, T. (2022). Inferring the Best Static Analysis Tool for Null Pointer Dereference in Java Source Code. *Proceedings http://ceur-ws.org ISSN*, *1613*, 0073.

Ami, A. S., Moran, K., Poshyvanyk, D., & Nadkarni, A. (2024). "False negative-that one is going to kill you": Understanding Industry Perspectives of Static Analysis based Security Testing. In *2024 IEEE Symposium on Security and Privacy (SP)* (pp. 3979-3997). IEEE.

Bennett, G., Hall, T., Winter, E., & Counsell, S. (2024). Semgrep*: Improving the limited performance of static application security testing (SAST) tools. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering* (pp. 614-623).

Berdine, J., Calcagno, C., & O'Hearn, P. W. (2006). Smallfoot: Modular automatic assertion checking with separation logic. In *Formal Methods for Components and Objects: 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, November 1-4, 2005, Revised Lectures 4* (pp. 115-137). Springer Berlin Heidelberg.

Calcagno, C., Distefano, D., O'Hearn, P., & Yang, H. (2008). Space invading systems code. In *International Symposium on Logic-Based Program Synthesis and Transformation* (pp. 1-3). Berlin, Heidelberg: Springer Berlin Heidelberg.

Calcagno, C., Distefano, D., O'Hearn, P., & Yang, H. (2009). Compositional shape analysis by means of bi-abduction. In *Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (pp. 289-300).

Gobbi, M. F., & Kinder, J. (2023). Poster: Using CodeQL to Detect Malware in npm. In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (pp. 3519-3521).

Hajiyev, E., Verbaere, M., & De Moor, O. (2006). Codequest: Scalable source code queries with datalog. In *ECOOP 2006– Object-Oriented Programming: 20th European Conference, Nantes, France, July 3-7, 2006. Proceedings 20* (pp. 2-27). Springer Berlin Heidelberg.

Kharkar, A., Moghaddam, R. Z., Jin, M., Liu, X., Shi, X., Clement, C., & Sundaresan, N. (2022). Learning to reduce false positives in analytic bug detectors. In *Proceedings of the 44th International Conference on Software Engineering* (pp. 1307-1316).

Li, K., Chen, S., Fan, L., Feng, R., Liu, H., Liu, C., ... & Chen, Y. (2023). Comparison and Evaluation on Static Application Security Testing (SAST) Tools for Java. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 921-933).

Li, Z., Liu, Z., Wong, W. K., Ma, P., & Wang, S. (2024). Evaluating C/C++ Vulnerability Detectability of Query-Based Static Application Security Testing Tools. *IEEE Transactions on Dependable and Secure Computing*.

Linton, M. A. (1984). Implementing relational views of programs. ACM SIGSOFT Software Engineering Notes, 9(3), 132-140.

Liu, H., Chen, S., Feng, R., Liu, C., Li, K., Xu, Z., ... & Chen, Y. (2023). A comprehensive study on quality assurance tools for java. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 285-297).

Mantovani, A., Compagna, L., Shoshitaishvili, Y., & Balzarotti, D. (2022). The Convergence of Source Code and Binary Vulnerability Discovery--A Case Study. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security* (pp. 602-615).

Nachtigall, M., Schlichtig, M., & Bodden, E. (2022). A large-scale study of usability criteria addressed by static analysis tools. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 532-543).

Reynolds, J. C. (2002). Separation logic: A logic for shared mutable data structures. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science* (pp. 55-74). IEEE.

Shen, M., Pillai, A., Yuan, B. A., Davis, J. C., & Machiry, A. (2023). An Empirical Study on the Use of Static Analysis Tools in

Open Source Embedded Software. *arXiv preprint arXiv:2310.00205*.

Wadhams, Z. D., Izurieta, C., & Reinhold, A. M. (2024). Barriers to Using Static Application Security Testing (SAST) Tools: A Literature Review. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering Workshops* (pp. 161-166).

Uitgave
Nationaal Cyber Security Centrum (NCSC)
Postbus 117, 2501 CC Den Haag
Turfmarkt 147, 2511 DP Den Haag
070 751 5555

Meer informatie
www.ncsc.nl
info@ncsc.nl
@ncsc_nl

Augustus 2025