Security tooling in CI/CD pipelines

Afstudeerdossier

Computest

always on.

Ramin Töpfer

De Haagse Hogeschool
HBO-ICT NSE CST
3 juni 2022, Zoetermeer

Bedrijfsbegeleider: Matthijs Melissen
Stagebegeleider: Wilhelm Wieringa

Versiebeheer

Versie	Datum	Wijzigingen	Reviewer(s)
0.0	7 februari 2022	Opzet-versie	-
0.1	18 april 2022	Concept-bespreking	Matthijs Melissen
0.2	25 april 2022	Concept-bespreking	Wilhelm Wieringa
0.3	18 mei 2022	TTA	Matthijs Melissen
1.0	25 mei 2022	TTA	Wilhelm Wieringa & Examinator

Referaat

Ramin Töpfer, NSE Cyber Security Technology, ‘Security tooling in CI/CD pipelines’, Afstudeerdossier, Computest, Zoetermeer, 2022.

Descriptoren:

- Cyber security
- DevOps
- Pipelines
- Automation
- Proof of Concept
- Github
- GitLab
- Security tooling
- Open-source

Voorwoord

Voor u ligt het afstudeerdossier, het onderzoek voor dit dossier is uitgevoerd bij het bedrijf Computest. Dit dossier is geschreven in het kader van mijn afstuderen aan de opleiding HBO ICT in de afstudeerrichting Cyber Security aan de Haagse Hogeschool Delft. In de periode van 7 februari 2022 tot en met 3 juni 2022 heb ik gewerkt aan het onderzoek en het schrijven van het afstudeerdossier.

De onderzoeksvraag voor deze afstudeerstage heb ik samen met mijn bedrijfsbegeleider, Matthijs Melissen, opgesteld. Na uitvoerig onderzoek heb ik de onderzoeksvraag kunnen beantwoorden. Tijdens dit onderzoek stonden mijn bedrijfsbegeleider, Matthijs Melissen, en mijn stagebegeleider vanuit mijn opleiding, Fred Wieringa, altijd open voor vragen. Zij hebben mijn vragen beantwoord wanneer nodig waardoor ik verder kon met mijn onderzoek.

Bij deze wil ik graag mijn begeleiders bedanken voor de fijne begeleiding en ondersteuning tijdens de afstudeerstage.

Verder wil ik mijn collega's bij Computest graag bedanken voor de fijne samenwerking, ik heb regelmatig met hen kunnen discussiëren over mijn onderzoek.

Ik wens u veel leesplezier toe.

Ramin Töpfer

Zoetermeer, 18 mei 2022

Inhoudsopgave

1. Inleiding

Een groot gedeelte van de klanten van Computest maakt gebruik van CI/CD[1] pipelines. Deze pipelines worden gebruikt om de applicatie die ontwikkelt wordt uit te brengen. Om de security in deze applicatie te bevorderen wordt er regelmatig gewerkt met security tools die in deze pipelines geïntegreerd kunnen worden. Enkele voorbeelden hiervan zijn linters, static code analyzers en dependency checkers. Deze tools kunnen de ontwikkelaars van de klanten ondersteunen in het vinden en oplossen van security problemen in de applicatie.

Computest merkt op dat CI/CD tools in de praktijk vaak lastig in gebruik zijn. Ze geven bijvoorbeeld veel false-positives of het is lastig te bepalen welke meldingen wel en niet relevant zijn. Dit leidt er vaak toe dat de ontwikkeling regelmatig door de tools geblokkeerd wordt, of dat meldingen van de tools genegeerd worden. Computest heeft voorgesteld om een project te starten waarbij onderzoek wordt gedaan naar de implementatie van CI/CD security tools in andere projecten. Open-source projecten sluiten hier goed op aan, aangezien dit type projecten openbaar zijn en de bijhorende CI/CD pipelines publiek beschikbaar zijn.

Met dit onderzoek wordt een antwoord gezocht op de volgende onderzoeksvraag:

> *‘’Welke CI/CD security tools binnen open-source-gemeenschappen bewijzen het meest efficiënt/effectief te zijn in CI/CD pipelines?”*

Gedurende de periode van de afstudeerstage wordt er onderzoek gedaan naar de verschillende security tools die gebruikt kunnen worden in CI/CD pipelines. Dit onderzoek is relevant aangezien er binnen Computest veel vraag is naar functionele tools die gebruikt kunnen worden bij zijn klanten.

CI/CD security tooling is een erg breed onderwerp om te onderzoeken. De focus in het project zal liggen op tooling die al gebruikt wordt in andere open-source projecten. De onderzoeksdoelen zullen behaald worden door zowel kwalitatief als kwantitatief onderzoek uit te voeren. Deze onderzoeken zijn opgezet aan de hand van literatuurstudies, interviews en data-analyse met behulp van een proof of concept (PoC).

De opdrachtgever is Matthijs Melissen, security specialist bij Computest. Matthijs heeft gedurende de afstudeerstage de student begeleid met behulp van wekelijkse meetings en was beschikbaar voor vragen. Wanneer er resources of materiaal nodig waren voor de opdracht kon hier naar gevraagd worden.

Er is gekozen voor de Sashimi waterval methode, hierbij wordt er analyse gedaan in de vorm van onderzoek en vallen de fases ontwerp, implementatie en testen onder het opzetten van een PoC. Het uitvoeren van tests wordt aan de hand van een PoC gedaan.

Alle opgeleverde en relevante documentatie is te vinden in het meegeleverde bijlagenboek.

1. CI/CD = continuous integration/continuous delivery

2. Computest

2.1 Organisatieomschrijving

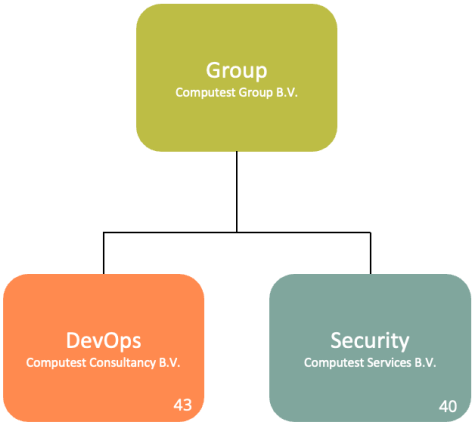
Computest is in 2005 opgericht door Hartger Ruijs. Het is een commercieel bedrijf dat zich specialiseert in DevOps (automatisering), Security en Performance. Enkele klanten en partners van Computest zijn: Staatsloterij, NS, 9292, Knab en Squla.

Waar Computest ooit is begonnen als een club die enkel specialisten detacheerde, werkt inmiddels de helft van de specialisten op projectbasis voor de klanten van Computest. Een belangrijke stap in de groei was de overname van Pine Digital Security in 2015. Hierdoor heeft Computest security aan zijn portfolio toegevoegd (Computest, z.d.).

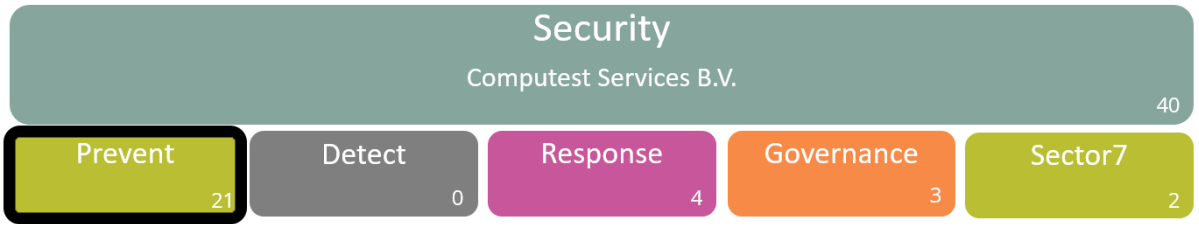
Organisatiestructuur

Computest is gevestigd in Zoetermeer en heeft ongeveer 100 werknemers. Het bedrijf is opgedeeld in 2 divisies: Computest Security en Computest DevOps.

Figuur 2.1 *Kantoorgebouw Zoetermeer*



Figuur 2.2 *Overzicht divisies*



Figuur 2.3 *Overzicht organisatiestructuur*

De opdracht is uitgevoerd onder het *Prevent* onderdeel van de security afdeling, zoals te zien in Figuur 2.3 (Computest, z.d.).

2.2 Missie en visie

In deze paragraaf wordt beschreven waar Computest voor staat en wat het bedrijf graag wilt bereiken. *"Plezier in ons werk vinden we bij Computest minstens zo belangrijk! We vinden het dan ook belangrijk dat alle teams regelmatig met elkaar op pad gaan en we organiseren borrels en pizzasessies. Enerzijds om vakinhoudelijk met elkaar van gedachten te wisselen en kennis te delen. Anderzijds om gewoon sociaal met elkaar te genieten van bijvoorbeeld*

een goed glas wijn. Het organiseren van onze jaarlijks studiereis naar de sneeuw is dan ook één van de tradities waar we elkaar beter leren kennen en onze kennis op peil houden."

"Wij zijn een gedreven groep sociale techneuten die vooral veel plezier willen beleven aan ons werk. Onze passie voel je aan ons onbegrensd enthousiasme: we zijn altijd nieuwsgierig naar nieuwe technologieën en willen alles tot op het bot doorgronden."

"Onze dagelijkse uitdaging bestaat erin om de IT van onze klanten beter en veiliger te maken. De digitale wereld heeft geen geheimen voor ons. Onze klanten krijgen topkwaliteit, zowel in onze adviezen als in de manier waarop we ze overbrengen en uitvoeren."
(Computest, z.d.)

2.3 Kernactiviteiten

Computest levert een groot assortiment aan diensten aan zijn klanten, deze zijn als volgt:

DevOps

- ***Testautomation:*** Computest helpt hun klanten met het opbouwen van automated tests, denk hierbij aan het automatisch laten draaien van opgestelde tests om zo hun software te testen. Hierbij wordt er gekeken of de software goed functioneert.

Security

- ***Prevent:*** Onder prevent valt onder andere pentesten, code reviews en vulnerability assessments. Deze type diensten helpen bedrijven in het versterken van hun Cyber Security om hacks te voorkomen.

- ***Detect:*** Detect is een dienst die Computest sinds kort aanbiedt, hierdoor kan een klant grip krijgen over actuele dreigingen in hun systemen (bijvoorbeeld binnen een Microsoft Azure omgeving).

- ***Response:*** Wanneer een klant van Computest zich in een noodsituatie bevindt op het gebied van security (gestolen data, ransomware attacks), kan deze 24 uur per dag contact opnemen voor support, met de noodlijn van Computest.

- ***Marvin_:*** *Marvin_* is een dienst die Computest al een tijd aanbiedt, het scant dagelijks het systeem van de klant. Wanneer er meldingen zijn worden deze gereviewd door security specialisten van Computest, als deze melding van belang is voor de klant wordt deze hiervan op de hoogte gesteld.

- ***Performance:*** Performance tests testen hoe goed de performance is van de software die getest wordt.

Voorbeelden van diensten die Computest levert aan klanten

- ***Staatsloterij*****:** het testen van performance in een cloud-omgeving (in verband met hoge benodigde capaciteit).

- ***NS*****:** trainen van IT-specialisten van NS in security-testen (in verband met toename aantal dreigingen op het gebied van cyber security).

- ***Aegon*****:** uitvoeren van security tests zodat de systemen en data veilig zijn. Dit valt onder de dienst Security Assurance (Computest, z.d.).

3. Opdrachtomschrijving

3.1 Probleemstelling

Computest merkt op dat bij zijn klanten CI/CD security tools in de praktijk vaak lastig in gebruik zijn. Ze geven bijvoorbeeld veel false-positives of het is lastig te bepalen welke meldingen wel en niet relevant zijn. Dit leidt er vaak toe dat de ontwikkeling regelmatig door de tools geblokkeerd wordt, of dat meldingen van de tools genegeerd worden.

Kennis op het gebied van CI/CD security tooling is binnen Computest slechts in beperkte mate aanwezig. Meer kennis op het gebied van CI/CD security tooling zou de opdrachtgever kunnen helpen om een beargumenteerde keuze te kunnen maken voor een CI/CD security tool die gebruikt kan worden bij zijn klanten.

3.2 Strategische relevantie

Het doel van het project is om bij te dragen aan de cyber security van de klanten van Computest, specifiek in de CI/CD pipelines.

Computest voert al security tests (penetration tests) uit bij hun klanten, maar het inzetten van CI/CD security tooling zal de security van de klant verder bevorderen.

Wanneer men gebruik blijft maken van niet goed functionerende CI/CD security tooling bij Computest en zijn klanten, zouden de relevante meldingen van tools over het hoofd gezien of overstemd kunnen worden door de grote hoeveelheid false-positives. Dit kan leiden tot onveilige situaties waarbij een kwetsbaarheid in een stuk software potentieel gemist wordt wat afgevangen had kunnen worden met een security tool.

3.3 Doelstelling

Zoals in de probleemstelling (Paragraaf 3.1) is genoemd is kennisverrijking op het gebied van CI/CD security tooling noodzakelijk zodat de opdrachtgever een keuze kan maken voor tooling die gebruikt kan worden bij zijn klanten.

Dit is de reden dat er wordt gekeken naar open-source projecten om kennis op te doen over hoe CI/CD security tooling wordt gebruikt in deze projecten. Het project dient als een kennisverbreding voor Computest.

Het uiteindelijke doel van deze opdracht is om een advies te geven aan Computest waarin de hoofdvraag (Paragraaf 3.4) wordt beantwoord. In dit advies worden een aantal CI/CD security tools aangeraden die het ‘beste’ blijken te scoren aan de hand van opgestelde test cases. Deze test cases worden opgesteld aan de hand van een analyse en worden vervolgens uitgevoerd met behulp van een opgesteld proof of concept (PoC).

3.4 Projectomschrijving

Het doel van het project is om te onderzoeken welke DevSecOps tools die gebruikt worden in open-source-gemeenschappen efficiënt/effectief zijn voor inzet bij klanten van Computest.

De focus in het project ligt specifiek op CI/CD security tooling die ook voorkomt in andere open source projecten.

De volgende hoofd en deelvragen zijn opgesteld:

Hoofdvraag

> *‘’Welke CI/CD security tools binnen open-source-gemeenschappen bewijzen het meest efficiënt/effectief te zijn in CI/CD pipelines?”*

Deelvragen

- *Wat houdt een CI/CD security tool in?*
- *Wat valt er onder CI/CD security tooling?*
- *Wat wordt er verstaan onder een ‘effectieve’ tool?*
- *Wat wordt er verstaan onder een ‘efficiënte’ tool?*
- *Welke security tools worden er gebruikt in de CI/CD pipelines van open-source-gemeenschappen?*
- *Welke voordelen hebben deze tools in de praktijk?*
- *Welke moeilijkheden ondervinden open-source-gemeenschappen bij het gebruik van deze tools?*

Scope

In deze subparagraaf zal worden aangegeven welke taken binnen of buiten het project liggen. De volgende taken liggen **wel** binnen de projectscope:

- Analyseren van hoofd/deelvragen aan de hand van kwalitatief/kwantitatief onderzoek
- Ontwerpen van een functioneel PoC
- Implementeren van een functioneel PoC
- Opzetten van test cases om CI/CD security tooling te testen/beoordelen
- Testen van verschillende CI/CD security tooling aan de hand van geïmplementeerde PoC en opgezette test cases
- Uitwerken van resultaten na testfase

De volgende taken liggen **niet** in de projectscope:

- Ontwerpen/ontwikkelen van een eigen CI/CD security tool
- Directe omgang met de CI/CD pipeline van klanten
- Implementeren resulterende CI/CD security tooling bij klanten

4. Aanpak

4.1 Methodiek

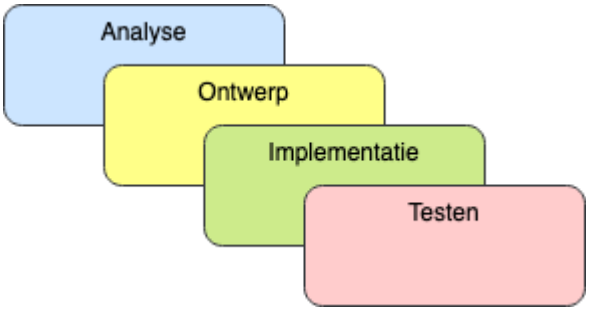
Er is onderzoek gedaan naar een passende methode bij een onderzoek-gerichte afstudeeropdracht, hieruit zijn een aantal methodes gekomen waaronder de SCRUM variatie SCORE (Scrum for Research) en de Sashimi watervalmethode.

Er heeft overleg plaatsgevonden met de stagebegeleider met betrekking tot de gebruikte methode, hieruit is geconcludeerd dat de Sashimi watervalmethode het beste zou passen bij het project. De reden dat de methode passend is bij een onderzoek-gerichte afstudeeropdracht heeft te maken met de manier waarop de verschillende fases overlappen met elkaar, dit maakt het mogelijk om terug te vallen op een vorige fase tijdens het project. Tijdens een onderzoek kan het namelijk voorkomen dat er informatie wordt opgedaan die betrekking heeft tot de vorige fase, of kan leiden tot nodige aanpassingen in een vorige fase.

De opzet van de verschillende fases van de Sashimi watervalmethode is te zien in Tabel 4.1, een overzicht van de fases is te zien in Figuur 4.2.

Fase		Beschrijving
1	Analyse	Onderzoek/deskresearch uitvoeren
2	Ontwerp	Ontwerpen van een PoC
3	Implementatie	Implementeren van het PoC
4	Testen	Testen van onderzoeksresultaten

Tabel 4.1 *Overzicht fasering*



Figuur 4.2 *Overzicht Sashimi watervalmethode*

Fase	Dagen	Onderzoeksvragen	(tussen) producten
1	10	*Wat houdt een CI/CD security tool in?* *Wat valt er onder CI/CD security tooling?* *Welke security tools worden er gebruikt in de CI/CD pipelines van open-source-gemeenschappen?*	Onderzoeksrapport
1	10	*Wat wordt er verstaan onder een 'effectieve' tool?* *Wat wordt er verstaan onder een 'efficiënte' tool?* *Welke moeilijkheden ondervinden open-source-gemeenschappen bij het gebruik van deze tools?'*	Onderzoeksrapport
2	15	-	Ontwerprapport
3	15	*Welke voordelen hebben deze tools in de praktijk?*	Implementatierapport
4	25	-	Testrapport
	10	Werken aan afstudeerdossier	Afstudeerdossier

Tabel 4.3 *Overzicht fasering & onderzoeksvragen*

In Tabel 4.3 staan een aantal fases en onderzoeksvragen onderverdeeld aan de hand van 75 dagen (+10 dagen voor het opbouwen van het afstudeerdossier). Deze fases worden afgesloten met een tussenproduct met betrekking tot de fase, dit zowel om voortgang te tonen aan de opdrachtgever als het notuleren van uitgevoerd onderzoek.

De fases zijn opgesteld aan de hand van de Sashimi-watervalmethode die aan het begin van deze paragraaf is behandeld.

Het was noodzakelijk om hands-on te gaan met desbetreffende CI/CD security tools. Dit om zo de moeilijkheden en voordelen verder te kunnen onderzoeken. Deze hands-on werkwijze heeft geleid tot een proof of concept, aan de hand van dit PoC is een advies opgesteld.

4.2 Onderzoeksmethoden

Er is gekozen om met verschillende onderzoeksmethoden te werken, per onderzoeksvraag/fase is een andere methode het best passend. Er is zowel gebruik gemaakt van kwalitatieve als kwantitatieve onderzoeksmethoden. In Tabel 4.4 wordt weergegeven welke methoden gebruikt zijn.

Onderzoeksvraag / fase	Benadering	Benadering	Onderzoek
Wat houdt een CI/CD security tool in? *Wat valt er onder CI/CD security tooling?*	Kwalitatief/ Kwantitatief	Secundair	Literatuurstudie
Welke moeilijkheden ondervinden open-source-gemeenschappen bij het gebruik van deze tools?'	Kwalitatief	Primair	Interview
Wat wordt er verstaan onder een 'effectieve' tool? *Wat wordt er verstaan onder een 'efficiënte' tool?*			
Welke security tools worden er gebruikt in de CI/CD pipelines van open-source-gemeenschappen?	Kwantitatief	Primair	Aselecte steekproef
Welke voordelen hebben deze tools in de praktijk?	Kwalitatief/ Kwantitatief	Primair	Observatie
Testfase	Parametrisch	Primair	Testen

Tabel 4.4 *Overzicht onderzoeksmethoden*

Literatuurstudie
Een belangrijk deel van het onderzoek was om secundaire data te verzamelen om zo de deelvragen te kunnen beantwoorden.

Interview
Om een beter begrip te krijgen van de moeilijkheden is er aan de hand van informele interviews primaire data verzameld.

Aselecte steekproef
Om aan te tonen wat voor tooling gebruikt wordt binnen open-source-gemeenschappen is er een aselecte steekproef uitgevoerd. Er is specifiek gekozen voor een **a**selecte steekproef aangezien er met kwantitatieve data wordt gewerkt (numerieke / statistische data).

Observatie
Om weer te geven wat voor voordelen tooling biedt in de praktijk is er aan de hand van het opgezette PoC een observatie gedaan.

Testen
In de testfase zijn er parametrische tests uitgevoerd.

4.3 Planning

In Tabel 4.5 op deze pagina is een planning weergeven voor de afstudeeropdracht.

Belangrijke data:

Bedrijfsbezoek (25% doorlooptijd):	ongeveer in werkweek 5.
Bespreking concept afstudeerdossier (60% doorlooptijd):	ongeveer in werkweek 11.
Tussentijds assessment (80% doorlooptijd):	ongeveer in werkweek 14.
Inleverdatum afstudeerdossier (100% doorlooptijd):	vrijdag 3 juni 2022, vóór 23:59u.

fase	kalenderweek	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	(tussen) producten
	werkweek	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
1. Analyse																			Onderzoeksrapport
1. Analyse																			Onderzoeksrapport
2. Ontwerp																			Ontwerprapport
3. Implementatie																			Implementatierapport
4. Testen																			Testrapport
Afstudeerdossier																			Afstudeerdossier

Tabel 4.5 *Werk planning*

4.4 Te leveren producten

De producten waar in de loop van het project aan is gewerkt is als volgt:

- Plan van Aanpak
- Onderzoeksrapport: Analyse fase, onderzoek naar CI/CD security
- Ontwerprapport: Ontwerp fase, ontwerpen PoC
- Implementatierapport: Implementatie fase, opzetten PoC
- Testrapport: Opzet test cases, testen CI/CD security tooling
- Advies: Opgenomen als hoofdstuk in afstudeerdossier

5. Analyse

5.1 Inleiding

In dit deel van het project is er onderzoek gedaan om de volgende deelvragen te kunnen beantwoorden:

- ***Wat houdt een CI/CD security tool in?***
- ***Wat valt er onder CI/CD security tooling?***
- ***Welke security tools worden er gebruikt in de CI/CD pipelines van open-source-gemeenschappen?***
- ***Wat wordt er verstaan onder een 'effectieve' tool?***
- ***Wat wordt er verstaan onder een 'efficiënte' tool?***

De deelvragen zijn opgedeeld in subvragen om deze zo beter te kunnen beantwoorden, deze zijn vermeld in het onderzoeksrapport.

Er is een literatuurstudie uitgevoerd om deze deelvragen te kunnen beantwoorden, verder hebben er nog (informele) interviews plaatsgevonden met medewerkers van Computest met betrekking tot de onderwerpen. Ook is er een aselecte steekproef uitgevoerd.

Verder is er vooronderzoek gedaan naar het opzetten van een proof of concept (PoC), waarbij onderzoek is gedaan naar een platform waar het beste een PoC opgezet kan worden. Ook is er nagedacht over wat voor repository gebruikt kan worden om tests op uit te voeren.

5.2 Huidige situatie

De huidige situatie kan inzicht geven over de relevantie van het project. Zoals eerder genoemd wordt er bij klanten van Computest regelmatig een security test (pentest) uitgevoerd om de cyber security te waarborgen (Paragraaf 2.3). Wat hiermee wordt bedoeld is dat er wordt gezocht naar mogelijke kwetsbaarheden in de applicatie die door de klant wordt ontwikkelt. Het uitsluiten van deze kwetsbaarheden is van groot belang.

Naast het testen van de applicaties van klanten wil Computest graag een breder advies kunnen geven over het opzetten van een veilige applicatie. CI/CD security tools kunnen hierbij helpen, het integreren van deze tools in de pipeline is een stap naar een veiliger proces.

Bij klanten van Computest wordt incidenteel gebruik gemaakt van een CI/CD security tool, maar zoals eerder genoemd is werken deze tools niet altijd naar behoren. De tools geven bijvoorbeeld veel false-positives en het is lastig te bepalen welke meldingen wel en niet relevant zijn. Dat leidt er vaak toe dat ontwikkeling geblokkeerd raakt door de tools, of dat meldingen genegeerd worden.

Collega's over DevSecOps tools

In de loop van de eerste twee weken van het project hebben er tijdens presentaties die zijn bijgewoond of tijdens de pauzes informele interviews plaatsgevonden. Bij deze interviews stond de ervaring met CI/CD security tooling centraal. Hieruit is waardevolle informatie naar boven gekomen die van pas kan zijn bij het onderzoek. De belangrijkste punten die resulteren uit desbetreffende interviews worden in deze paragraaf beschreven. Deze punten kunnen ondersteuning bieden om de huidige situatie van Computest op het gebied van CI/CD security tooling beter te begrijpen.

Collega's van de Security afdeling Prevent: Irritatiepunten over CI/CD security tooling
CI/CD security tools geven veel false-positives over onder andere XSS (cross-site scripting), HTTP headers en XSS in JSON files. Het komt er op neer dat de tools niet slim genoeg zijn, dit kan ertoe leiden dat meldingen of waarschuwingen van deze tools niet serieus genomen worden (en dus genegeerd worden).

Klanten gebruiken regelmatig tooling zoals SonarQube, nadelen hiervan zijn talloos veel false-positives en het kost erg veel tijd om goede policies in te stellen zodat het werkbaar wordt. Het is dan maar de vraag of het dan nog wel waard is om dit te automatiseren.

Tijdens de presentatie 'Functional Test Automation', door een Test Specialist
Er wordt wel eens gebruik gemaakt van de tool Netsparker binnen Computest.

Binnen Computest is geen voorkeur voor open source of commerciele tooling, zolang de tool maar prettig in gebruik is maken de kosten ervan niet erg veel uit.

Tijdens de presentatie 'Development', door een Senior Interaction Designer
Binnen de service *Marvin_* die Computest aanbiedt (Paragraaf 2.3) zijn een aantal security gerelateerde tools geïmplementeerd. Dit zijn de volgende tools:

- Nessus wordt gebruikt om scans uit te voeren van netwerken (timeline van issues van je servernetwerk)
- OpenVAS (Nessus open source)
- Sentinel Microsoft adapter (voor het aanmaken van issues binnen Marvin)

Een interessant punt wat naar boven was gekomen is dat er voorheen onderzoek is gedaan naar het implementeren van de CI/CD security tool *OWASP ZAP*. Het implementeren van deze tool (of een soortgelijke tool) zou de functionaliteit van Marvin_ sterk verbeteren.

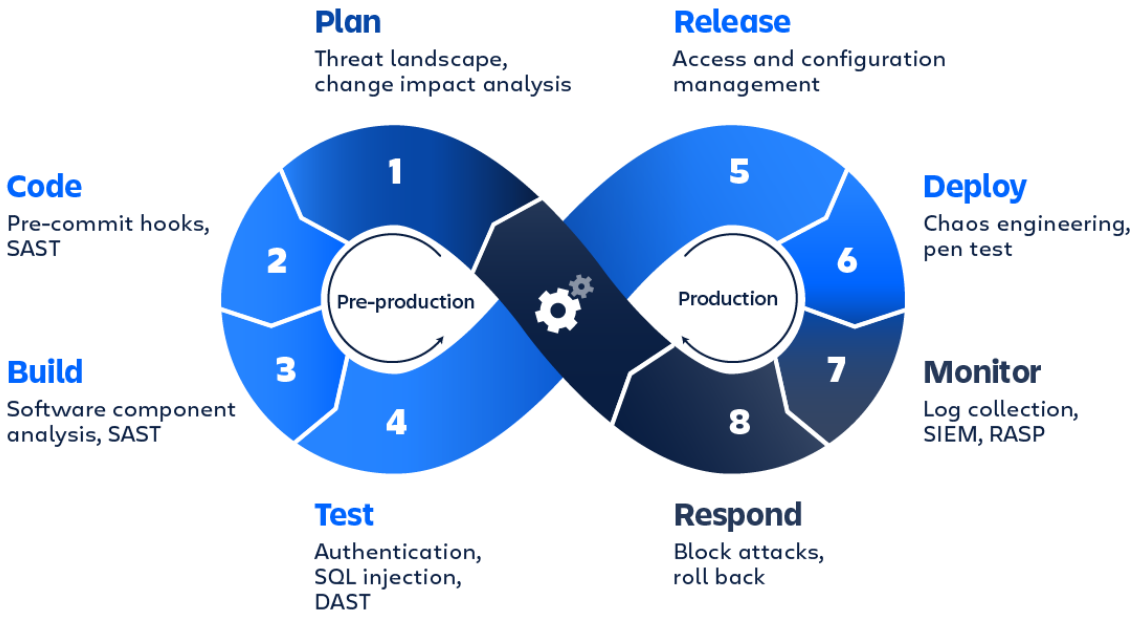
Conclusie
Computest medewerkers geven veel aan dat de tooling te veel false-positives melden en dat het opzetten van de tooling (voor goede functionaliteit) erg veel tijd in beslag neemt. Deze punten komen overeen met de probleemstelling die behandeld is in Paragraaf 3.1, ook is dit een bevestiging dat CI/CD security tools in de praktijk vaak lastig zijn in gebruik.

5.3 Wat houdt een CI/CD security tool in?

CI/CD security is het implementeren van security in een CI/CD (continuous integration, continuous delivery/deployment) pipeline. Door software security te automatiseren wordt het een actief, geintegreerd deel van het development proces (Atlassian, z.d.-b).

5.3.1 CI/CD pipeline

Een CI/CD pipeline is een serie aan stappen die uitgevoerd moeten worden om een nieuwe versie van een stuk software te kunnen leveren. Continuous integration/continuous delivery (CI/CD) pipelines zijn gefocust op het verbeteren van software levering door middel van DevOps (Red Hat, 2019). In Figuur 5.3.1 wordt een voorbeeld gegeven van een pipeline.



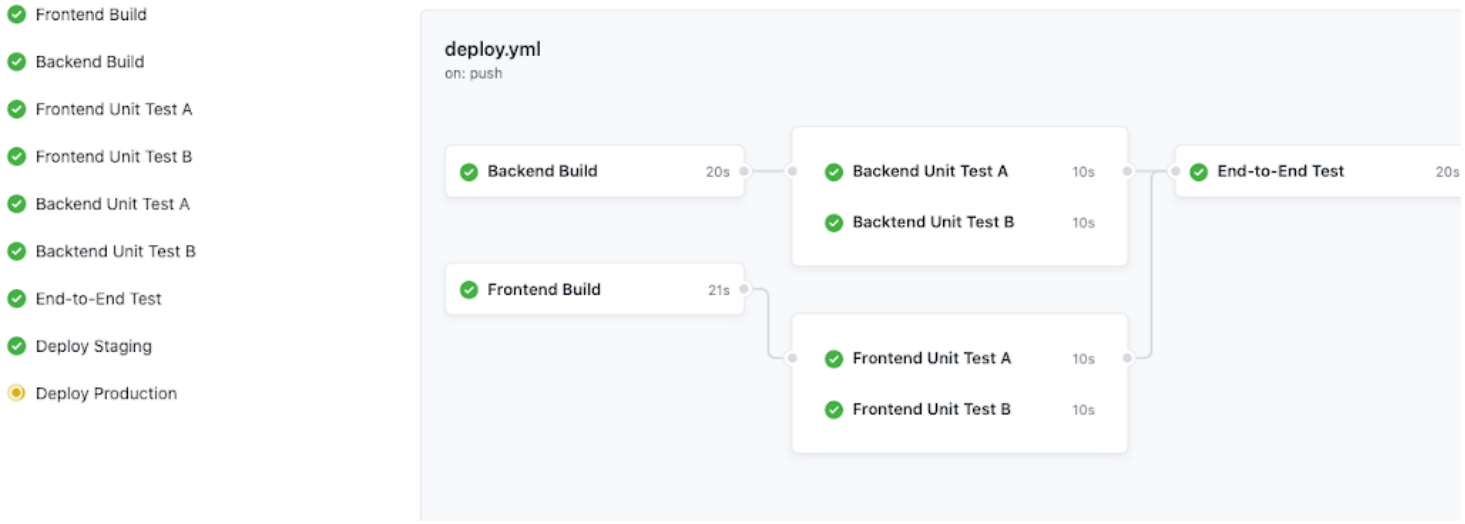
Figuur 5.3.1 *Voorbeeld (DevSecOps) pipeline (Atlassian, z.d.-a)*

Een CI/CD pipeline introduceert monitoring en automation om applicatie ontwikkeling processen te verbeteren, vooral in de fases *integration* en *testing*, ook tijdens *delivery en deployment*. Hoewel het mogelijk is om alle stappen uit een CI/CD pipeline handmatig uit te voeren, wordt de echte waarde van een CI/CD pipeline gerealiseerd door automation.

Elementen van een CI/CD pipeline
Hier volgt een voorbeeld van stappen die gevonden kunnen worden in een pipeline (deze zal bij elke organisatie verschillen):

Build - compilen (bouwen) van de applicatie
Test - testen van de code (op bijvoorbeeld performance)
Release - applicatie leveren aan de repository (bijvoorbeeld een Git repository)
Validation & compliance - security stappen die genomen kunnen worden (scannen)
Deploy - code leveren aan production (waar de klanten gebruik van maken)
(Red Hat, 2019)

Voorbeeld weergave CI/CD pipeline
Pipelines worden gebruikt bij (bijna) alle version-control software, denk hierbij aan Github, Gitlab of bijvoorbeeld Azure Devops. Deze bieden allemaal soortgelijke overviews van de pipeline, met verschillende ondersteuning en implementaties. In Figuur 5.3.2 is een voorbeeld te zien van hoe een pipeline er in de praktijk uit ziet bij Github.



Figuur 5.3.2 *Voorbeeld CI/CD Github*

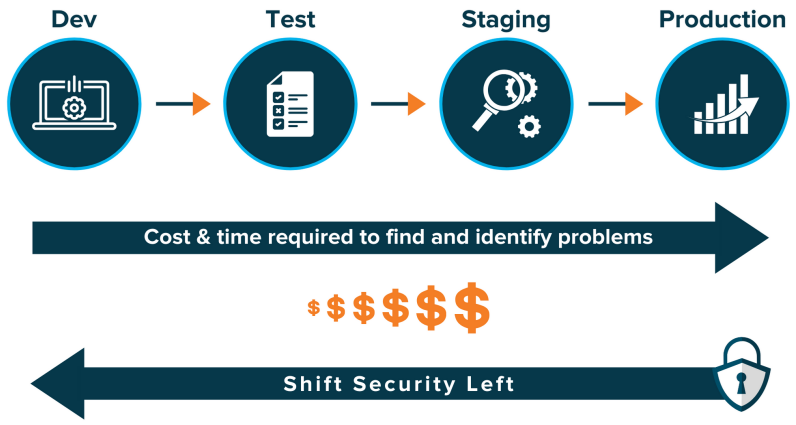
5.3.2 DevSecOps

Een andere benaming voor CI/CD security tooling is DevSecOps, deze benaming zal gedurende het document gebruikt worden. Bij DevSecOps is het van belang dat er in verschillende delen van de pipeline de juiste tools worden toegepast.

Een voorbeeld van hoe security tools gekoppeld kunnen worden aan de juiste stap in een CI/CD pipeline is te zien in Figuur 5.3.1. Voor elk deel van een CI/CD pipeline is een andere categorie security tool nodig, bij de stap *Code* is bijvoorbeeld een SAST (Static Application Security Tool) scan van belang (wordt later toegelicht) (Atlassian, z.d.-b).

Waarom zou je security in het DevOps proces (pipeline) willen voegen?
Dit heeft vooral te maken met kosten voor bedrijven, hoe verder je applicatie zich bevindt in de DevOps pipeline, hoe duurder (hoe meer tijd het kost) het is om nog aanpassingen te maken.

Stel dat er vlak voor de *Release* (aanleveren van de code en applicatie) een gevaarlijke kwetsbaarheid wordt gevonden, als security team of development team van een bedrijf zou je dit direct op moeten lossen. Het probleem is dat er nu teruggedraaid moet worden in het DevOps proces, de code moet worden aangepast en er zal opnieuw ge*build* en ge*test* moeten worden. Dit kost allemaal veel tijd en dus geld, als je eerder in het DevOps proces achter de kwetsbaarheid komt kan je dit eerder in het proces afvangen (Computest, 2022). Dit is te zien in Figuur 5.3.3.



Figuur 5.3.3 *Diagram Shift Left (Klogix, 2020)*

5.3.3 Shift Left

Een term die aan bod kwam in de literatuurstudie naar DevSecOps en in een interne presentatie bij Computest (over functional test automation) is *Shift Left*, deze term zal worden toegelicht.

Met shift left wordt er bedoeld dat security ingebouwd zit in het applicatie proces vanaf het begin van de development lifecycle. Wanneer dit eerder in het proces wordt gedaan (in vroege stappen) is het makkelijker problemen te vinden en op te lossen (Computest, 2022).

In traditioneel Security Testing worden er tests uitgevoerd na de *Release*:

Figuur 5.3.4 *Shift Left toelichting 1 (Pennington, 2019)*

Met DevSecOps worden er continue tools gedraaid door de gehele DevOps pipeline:

Figuur 5.3.5 *Shift Left toelichting 2 (Pennington, 2019)*

5.4 Wat valt er onder CI/CD security tooling?

DevSecOps tools zijn te groeperen in verschillende categorieën (GitLab, z.d.-c), in deze paragraaf zullen de belangrijkste categorieën worden behandeld. Met de ‘belangrijkste’ categorieën worden diegene bedoeld die het meest voorkomen in open-source projecten, dit zal later worden behandeld. De complete lijst aan gevonden categorieën DevSecOps tooling is te vinden in het onderzoeksrapport.

SAST & DAST

Static Application Security Testing (SAST)

SAST tools gaan te werk door de source code van een applicatie te scannen, een belangrijk kenmerk is dat dit statisch wordt gedaan (**S** = Static). Wat hier mee wordt bedoeld is dat de applicatie niet draait op het moment van de test, maar dat de regels aan code geanalyseerd worden (GitLab, z.d.-c).

Dynamic Application Security Testing (DAST)

Integendeel tot SAST is de applicatie wel draaiend tijdens de scan van een DAST tool, deze categorie tooling gaat dus dynamisch te werk (**D** = Dynamic). DAST tools voeren attacks uit op de draaiende applicatie vergelijkbaar met hoe een pentester dit zou doen, met voordeel dat de tool door automatisering vele attacks achter elkaar kan uitvoeren (GitLab, z.d.-c).
SAST en DAST tools verschillen dus als volgt van elkaar:

SAST	**DAST**
White box, op source code	Black box, op draaiende applicatie
Kan geen run-time/environment issues vinden	Kan run-time/environment issues vinden
Ondersteund alle soorten software	Ondersteund alleen web applicaties/services

Tabel 5.4.1 *SAST & DAST (Synopsys, 2016)*

Software Composition Analysis (SCA)

SCA tools voeren een Dependency Scan uit, een dependency van een applicatie is een gebruikte library of package die van een externe bron komt (bijvoorbeeld een open source package). Dit soort externe componenten kunnen kwetsbaarheden bevatten, een SCA tool scant de gebruikte dependencies op kwetsbaarheden aan de hand van een database. In deze database staan bekende kwetsbare libraries of packages (GitLab, z.d.-c).

Secret Detection

Secret detection tools scannen de repository (source code) van een applicatie en zoekt specifiek naar gevoelige tekst (‘high entropy strings’). Wat hiermee wordt bedoeld is dat er gezocht wordt naar stukken tekst die mogelijk een wachtwoord (of andere gevoelige tekst zoals een API key) kunnen zijn (GitLab, z.d.-c).

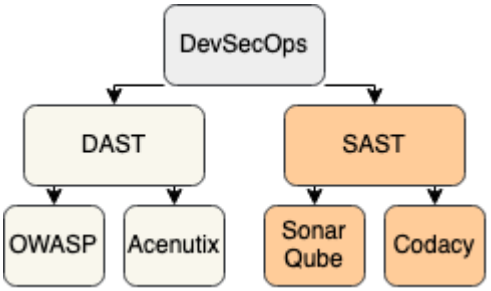
Container Scanning

Container scanning tools voeren een scan uit op containers, waaronder bijvoorbeeld Docker based containers. Deze containers kunnen kwetsbaarheden bevatten in bijvoorbeeld de componenten die hierin worden gebruikt. Vergelijkbaar met de manier waarop SCA tooling scant op dependencies, scannen container scanning tools ook de gebruikte dependencies (GitLab, z.d.-c).

5.4.1 DevSecOps tools

Er zijn een groot aantal DevSecOps tools beschikbaar, in deze subparagraaf worden uit de verschillende eerder genoemde categorieën enkele tools toegelicht (de genoemde tools dienen enkel als voorbeeld). De complete lijst aan tools die naar boven zijn gekomen in de uitgevoerde literatuurstudie is te vinden in het onderzoeksrapport. Figuur 5.4.2 bevat een visualisatie van de indeling van de tooling.

Codacy (SAST)
Codacy is een commerciele SAST tool die je kunt aansluiten op de repository van je applicatie. Deze tool scant de source code van de applicatie op bekende kwetsbaarheden, vervolgens is het mogelijk de gevonden problemen in te zien in de vorm van een rapport (Codacy, z.d.).



Figuur 5.4.2 *Indeling tooling (voorbeeld)*

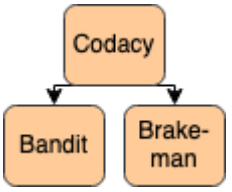
OWASP ZAP (DAST)
Zed Attack Proxy (ZAP) is een open source tool die is ontwikkelt door OWASP, deze tool voert een aantal attacks uit op een draaiende applicatie om zo kwetsbaarheden te vinden in de applicatie. Het is mogelijk om de tool te implementeren in een CI/CD pipeline om zo elke nieuwe versie van de applicatie (bij veranderingen aan de source code) automatisch te laten testen. De tool genereert na afloop van de test een rapport met gevonden problemen (OWASP, z.d.).

WhiteSource Bolt (SCA)
WhiteSource Bolt is een commerciele tool die de gebruikte dependencies binnen een project scant op kwetsbaarheden. Deze scan wordt gedaan aan de hand van een database met kwetsbare dependencies. De resultaten zijn vervolgens zichtbaar in een rapport (WhiteSource Software, z.d.).

Snyk (compleet pakket)
Vele bedrijven bieden ook complete oplossingen (gevarieerd compleet) voor security in CI/CD pipelines, een voorbeeld hiervan is Snyk. Snyk biedt zowel SAST, SCA en container scanning aan. Een voordeel aan dit type ‘pakketten’ is onder andere dat je de resultaten van de verschillende categorie tooling op één centrale plek kan inzien (Snyk, z.d.).

Plugins (engines)
Vele DevSecOps oplossingen maken onder water gebruik van dezelfde ‘scanning engines’, dit is vooral het geval bij SAST tooling. Neem bijvoorbeeld *Codacy*, Codacy maakt onder andere gebruik van de plugins Bandit en Brakeman. Dit zijn open-source scanning tools die ook individueel te implementeren zijn (Codacy, 2020).

GitLab en Github maken ook gebruik van verschillende plugins, hier is een literatuurstudie over uitgevoerd. De resultaten hiervan zijn te vinden in het onderzoeksrapport.



Figuur 5.4.3 *Indeling plugins (voorbeeld)*

5.5 Welke security tools worden er gebruikt in de CI/CD pipelines van open-source-gemeenschappen?

In deze paragraaf staat de deelvraag: '*Welke security tools worden er gebruikt in de CI/CD pipelines van open-source-gemeenschappen?'* centraal, deze vraag zal deels worden beantwoord. Verschillende CI/CD security tools die in open-source-gemeenschappen worden gebruikt zullen worden behandeld. Deze informatie is resulterend uit verschillende onderzoeksmethoden die benoemd zullen worden.

5.5.1 Gebruikte software voor hosting van open-source projecten

Voordat er gezocht kan worden naar verschillende open-source projecten en de bijhorende CI/CD pipeline kan worden onderzocht. Moet er eerst een selectie worden gemaakt aan hosting platforms.

Er is een literatuurstudie uitgevoerd om te achterhalen welke platforms de meeste tractie hebben wat betreft open-source projecten. Er zijn onder andere artikelen geraadpleegd waarin de hoeveelheid gebruikers van verschillende hosting services wordt behandeld. Een voorbeeld van een desbetreffend artikel is een blogpost van Xopero software (Xopero Software S.A., 2021). Hieruit is gebleken dat de volgende platforms het meest worden gebruikt om open-source projecten op te hosten:

Github
Github is een van de meest bekendste version control systemen en maakt gebruik van Git. De meeste bekende en recente open-source projecten zijn te vinden op Github, onder de *Explore* pagina van Github zijn dan ook talloos populaire nieuwe repositories te vinden.

GitLab
GitLab is naast Github ook een bekend platform om open-source projecten op te hosten, vergeleken met Github staan hier echter minder projecten op.

Bitbucket & Azure DevOps
Bitbucket en Azure DevOps worden minder gebruikt om open-source projecten op te delen, dit komt mede doordat deze platforms niet bekend staan om hun sociale aspecten. Deze platforms beschikken bijvoorbeeld niet over een *Explore* of *Trending* pagina waarin open-source projecten worden weergegeven.

Self-hosted oplossing
Het komt voor dat open-source projecten gebruik maken van een self-hosted platform.
Voorbeelden van projecten die gebruik maken van een self-hosted platform zijn onder andere Mozilla Firefox en Blender (een van de grootste open-source projecten), deze projecten worden ge-selfhost op Phabricator.

5.5.2 DevSecOps tools in open-source projecten

In deze paragraaf zijn de resultaten van een aselecte steekproef weergegeven, het doel van de steekproef is om grote open-source projecten te vinden waarin DevSecOps tools worden gebruikt. Er is onder andere op Github en GitLab gezocht onder *Explore* en *Trending* naar projecten met bijvoorbeeld de meeste sterren of views. Verder is er secundaire data verzameld aan de hand van gepubliceerde artikels waarin onderzoek is gedaan naar de meest populaire en grootste open source projecten, waaronder Github, GitLab, BitBucket en Azure DevOps.

Vervolgens is er onderzoek gedaan naar de pipelines van desbetreffende open-source projecten, de gebruikte DevSecOps tools die hierin werden gebruikt zijn vervolgens genoteerd. Enkele voorbeelden worden toegelicht, alle resultaten zijn te vinden in het onderzoeksrapport.

Open-source projecten op Github

VScode
Visual Studio Code is een code editor developed door Microsoft, het project wordt gehost op Github en maakt gebruik van een aantal tools om de security te behouden. Waaronder:

- CodeQL (SAST)
- eslint (linter)

Medusa
Medusa is een open-source alternatief van de website Shopify, binnen de repository wordt er gebruik gemaakt van een Secret Detection tool. Door gebruik te maken van de tool GitGuardian wordt er voorkomen dat er gevoelige data (bijvoorbeeld wachtwoorden) op de repository worden geplaatst.

Ant Design
Ant Design is een React UI library die gebruikt kan worden om enterprise-class user interfaces te ontwerpen voor webapplicaties. Binnen de repository wordt er gebruik gemaakt van de volgende tools:

- Codacy (SAST)
- Gitleaks (Secret Detection)

SerenityOS
SerenityOS is een open-source besturingsysteem die ontwikkelt wordt door een groep hobbyisten, binnen dit project worden de volgende tools gebruikt om de security van de applicatie te behouden:

- PVS-Studio Static Analysis (SAST)
- Sonar Cloud Static Analysis(SAST)

Open-source projecten op GitLab

Meltano
Meltano is een DataOps besturingssysteem wat gehost wordt op GitLab, binnen de repository wordt gebruik gemaakt van de SAST tools Semgrep, Brakeman, Bandit en eslint.

Wireshark
WireShark is een open-source packet analyzer die gebruikt kan worden voor het analyseren van netwerkverkeer, de volgende tools worden gebruikt binnen het project:

- Coverity GCC Scan (SAST)
- Clang Static Analyzer (SAST)
- ASAN Fuzzing
- Cpp check (SAST)

Inkscape
Inkscape is een open-source vector image editor, er wordt gebruik gemaakt van de linter Clang-tidy:linux en de SAST tool Code Quality + Protect.

Open-source projecten op Bitbucket & Azure DevOps

Op Bitbucket en Azure DevOps zijn geen grote open-source projecten gevonden waar gebruik gemaakt wordt van DevSecOps tooling.

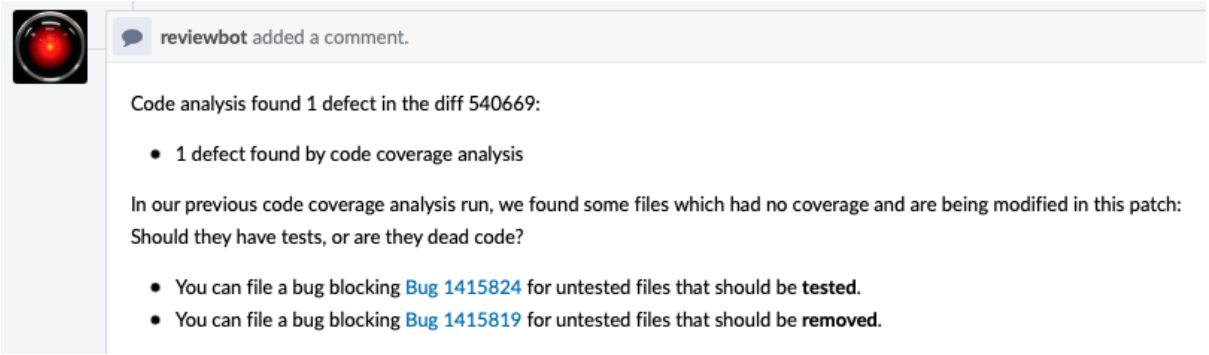
Open-source projecten op self hosted oplossing

Firefox
Firefox is een van de meest gebruikte browsers en valt onder een van de grootste open source projecten die er bestaan. In deze subparagraaf is er een kort onderzoek uitgevoerd naar de CI/CD pipeline van Firefox en de DevSecOps tools die hierin worden gebruikt (Mozilla, z.d.-a).

CI/CD pipeline
Mozilla maakt gebruik van Mercurial, een versiebeheertool soortgelijk aan Git.

Mozilla heeft een eigen dashboard systeem *Treeherder* waarin gebruikers de resultaten van automatische builds en resultaten van tests kunnen inzien. Verder wordt er gebruik gemaakt van *Phabricator* (in combinatie met Mercurial), een collaboration tool zoals Github waarin gebruikers onder andere pull requests of commits kunnen aanmaken.

DevSecOps tools
Binnen Phabricator wordt er gebruik gemaakt van een bot (*reviewbot, Code Review Bot)*, deze bot voert automatisch tests en checks uit op nieuwe commits en pull requests en geeft weer wat voor problemen er gevonden zijn.
Een voorbeeld van een gevonden probleem is te zien in Figuur 5.5.1.



Figuur 5.5.1 *Screenshot reactie reviewbot (Mozilla, z.d.-c)*

In Figuur 5.5.1 is er in een pull request van een gebruiker een probleem gevonden aan de hand van code coverage analysis.
Er is in de documentatie van Mozilla code-review meer info te vinden over welke specifieke tools er gebruikt worden in de back-end van deze bot (Mozilla, z.d.-b). Deze zijn als volgt:

- clang-format + clang-tidy (style analyzers)
- Mozlint (bevat meerdere linters, waaronder: flake8, eslint en rustfmt)
- Coverity (SAST)

Blender
Blender is een gratis 3D-modelling software developed door een nederlands bedrijf, het is net zoals Firefox open source en is ook een van de grootste open-source projecten. In deze subparagraaf wordt er een kort onderzoek uitgevoerd naar de versioning software die gebruikt wordt en welke DevSecOps tools gebruikt worden.

Versioning software en CI/CD pipeline
Blender maakt gebruik van een self hosted *Phabricator* omgeving waarin code reviews en code audits plaatsvinden.

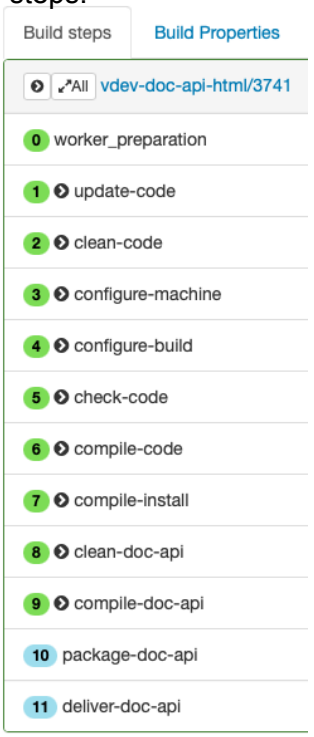
Blender gebruikt *Buildbot* voor het automatisch builden en testen van nieuwe commits en pull requests. Deze tool is vergelijkbaar met bijvoorbeeld een CI/CD pipeline die je in GitLab kunt vinden, in Figuur 5.5.2 is een voorbeeld te zien van een aantal build steps.

DevSecOps tools
Zoals eerder genoemd zitten er teststappen in de pipeline die met BuildBot is opgesteld, er worden hier verschillende unit tests uitgevoerd. Er zijn geen DevSecOps tools gevonden in de pipeline (Blender, z.d.-b).

Conclusie

De resultaten laten zien dat er een grote variatie in de tooling is die voorkomt in open-source projecten. Wat opvalt is dat in de meeste projecten op Github gebruik gemaakt wordt van CodeQL, dit is de geïntegreerde tooling die Github zelf aanbiedt. Bij projecten op GitLab valt het op dat er vooral tooling wordt gebruikt die GitLab zelf aanbiedt (en dus ook uitgebreide integratie heeft met het platform).

Verder is het interessant dat er enkele projecten gevonden zijn met erg uitgebreide CI/CD pipelines en vele verschillende security tools. Terwijl er in sommige andere projecten enkel gebruik gemaakt wordt van een linter.



Figuur 5.5.2 *Screenshot Buildbot (Blender, z.d.-a)*

Dat er vooral gebruik gemaakt wordt van CodeQL en GitLab's eigen tooling is mogelijk als volgt te redeneren:
Beheerders van de repositories geven de uitgebreide integratiemogelijkheden veel waarde.
Beheerders maken simpelweg gebruik van deze tools aangezien deze makkelijk op te stellen zijn en (grotendeels) gratis aangeboden worden.

5.6 Steekproef categorie DevSecOps tooling

In Paragraaf 5.5 is er onderzoek gedaan naar welke DevSecOps tools er gebruikt worden in open source projecten op (onder andere) Github en GitLab. In deze paragraaf zal er een steekproef worden uitgevoerd aan de hand van de top 50 grootste projecten op zowel Github en GitLab. De resultaten van deze steekproef zullen worden uitgewerkt in een grafiek. Deze paragraaf is een vervolg op 5.5 en zal ook dienen om antwoord te kunnen geven op de deelvraag: '*Welke security tools worden er gebruikt in de CI/CD pipelines van open-source-gemeenschappen?'.*

Het doel van deze steekproef is om in een oogopslag te kunnen zien welk **categorie** DevSecOps tool het meest wordt gebruikt in open-source projecten. Het gaat hier om de **categorie** tool, bijvoorbeeld SCA of SAST (Paragraaf 5.4).

5.6.1 Opzet steekproef

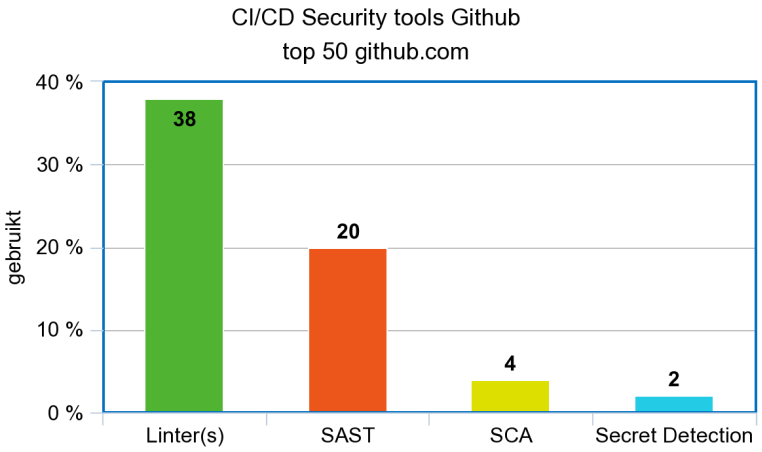
Er is op zowel Github als GitLab gezocht op de grootste open-source projecten, de steekproef is genomen op *22/02/2022*.

Github
Onder 'grootste' wordt het volgende verstaan: top 50 projecten onder de zoekterm: *stars:>5000* gesorteerd op *Most Stars*. Op zowel Github en GitLab is *Starring* een methode om een repository op te slaan, verder dient de hoeveelheid Stars die een repository heeft als een manier om projecten te rangschikken. De *Explore* pagina laat bijvoorbeeld populaire repositories zien gebaseerd op de hoeveelheid Stars die deze repositories hebben ontvangen (Github, z.d.-b).

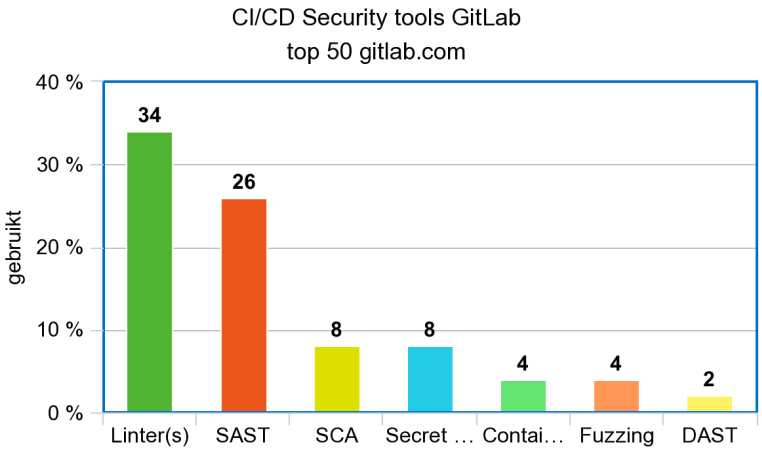
GitLab
Onder 'grootste' wordt het volgende verstaan: top 50 projecten onder *Most Stars* gesorteerd op *Most Stars*.

5.6.2 Resultaten

Aan de resultaten is te zien wat de meest gebruikte categorie aan DevSecOps tools is in de **top 50** open-source projecten. *Gebruikt* = hoe vaak de tool voorkwam in de top 50 projecten.



Grafiek 5.6.1 *DevSecOps tools Github*



Grafiek 5.6.2 *DevSecOps tools GitLab*

Aan de hand van de resultaten is vast te stellen welke categorie DevSecOps tooling het meest gebruikt wordt in de top 50 open-source projecten op Github en GitLab.

5.6.3 Conclusie

We zien dat vooral SAST en Linter tools het meest gebruikt worden in de CI/CD pipelines van open-source-gemeenschappen. Met Grafiek 5.6.1 en 5.6.2 kan de volgende vraag worden beantwoord: '*hoeveel procent van de top 50 projecten op Github en GitLab gebruikt een <> tool?*', verder is te concluderen dat op zowel Github als GitLab de meest gebruikte categorie DevSecOps tool een *Linter* is. Na Linter en SAST tooling zijn SCA tools het meest voorkomend.

Wat verder erg opvalt is dat er in een groot gedeelte van de projecten, gemiddeld 62% (te zien in de volledige resultaten in het bijlagenboek), geen enkele categorie DevSecOps tooling wordt gebruikt. Dit is erg opmerkelijk aangezien veel van deze applicaties erg populair zijn en erg actief worden gebruikt. Dan zou de vraag gesteld kunnen worden of deze applicaties wel veilig zijn en of de gebruikers van desbetreffende applicaties mogelijk kwetsbaar zijn voor potentiële attackers? Ook zou het mogelijk kunnen zijn dat de ontwikkelaars van de open-source projecten de applicatie (source code of bijvoorbeeld gebruikte libraries) lokaal op hun eigen machine scannen op kwetsbaarheden.

Verder zijn de resultaten te redeneren aan de hoeveelheid tijd en moeite dat het kost om de categorie aan tooling op te zetten. Theoretisch zijn Linter en SAST tools gemakkelijk op te stellen aangezien het statische tooling is, hier wordt in Hoofdstuk 5.11.1 en 7 verder op toegelicht.

5.7 Afhandeling gevonden issues in open-source projecten

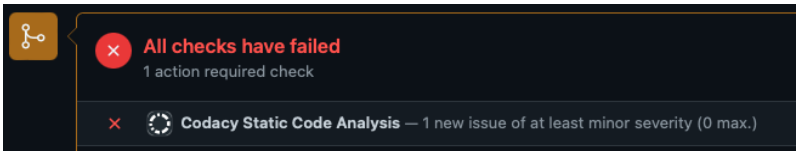
In deze paragraaf worden er aan de hand van een observatie voorbeelden gegeven van issues die gevonden zijn in open-source projecten door DevSecOps tooling. Dit kan een beter zicht geven op wat voor problemen deze tools in de praktijk aan het licht kunnen brengen. Er zal een open-source project worden genoemd en er zal vervolgens een gevonden probleem worden toegelicht met potentiële gevolgen wanneer dit probleem niet spoedig wordt opgelost. Er is dieper ingegaan op projecten die resulteren uit de steekproef in Paragraaf 5.5.

5.7.1 Ontbrekende afhandeling van gevonden issues

In deze subparagraaf zal worden gekeken naar gevonden issues in open-source projecten waarbij de afhandeling ontbreekt.

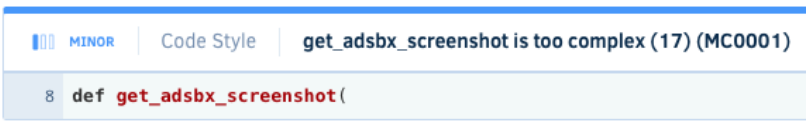
Voorbeeld van repository: **plane-notify**

In een pull request heeft de SAST tool *Codacy* (Subparagraaf 5.4.1) een minor issue gevonden, er is in de pull request te zien dat de automated test gefaald heeft:



Figuur 5.7.1 *Screenshot pull request plane-notify*

Op de frontend van de SAST tool wordt verdere toelichting gegeven over de melding, in dit geval is het een simpele *Code Styling* issue:

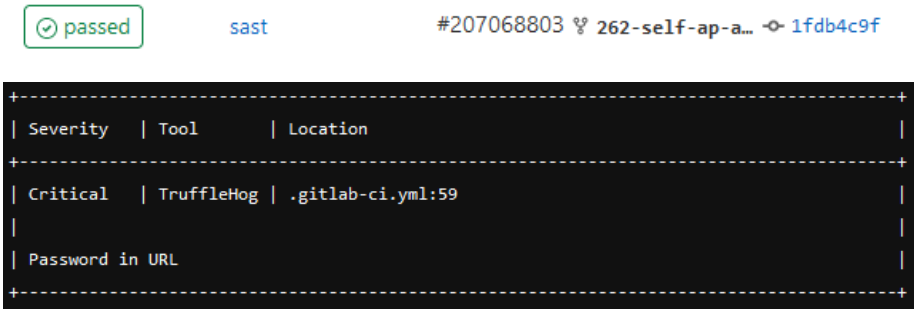


Figuur 5.7.2 *Screenshot Codacy plane-notify*

Dit type issue brengt weinig tot geen security problemen met zich mee, het is een probleem wat alleen de leesbaarheid (quality) van de code naar beneden brengt. Vandaar dat het wordt aangegeven als *minor* issue.

Voorbeeld van repository: **prismo**

In een check van de tool *TruffleHog* (Secret Detection, Paragraaf 5.4) is er een Secret gevonden in de code, dit is als volgt als *Critical* severity te zien:



Figuur 5.7.3 *Screenshot SAST prismo*

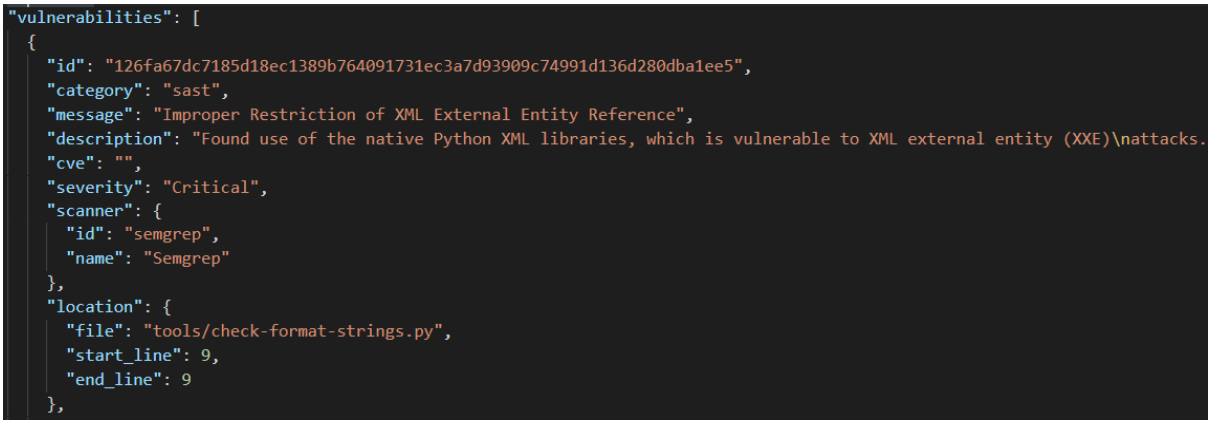
Wanneer er in de code wordt gekeken staat er inderdaad een password in de url:

```
59      DATABASE_URL: "postgresql://postgres:postgres@postgres:5432/$POSTGRES_DB"
```

Het is zorgwekkend dat de tool zo is ingesteld dat de pipeline wel als *passed* wordt aangegeven bij het vinden van een *Critical severity*. Er zou worden verwacht dat bij het vinden van een secret in de code, de pipeline niet succesvol zou moeten afronden. In dit geval is het een default password van een (waarschijnlijk) nog niet geconfigureerd PostgreSQL database.

Voorbeeld van repository: **fdroid-client**

In een check van de tool *Semgrep* (SAST, Paragraaf 5.4) is er een kwetsbaarheid gevonden in de code, deze kwetsbaarheid is gemarkeerd als *Critical* severity.



Figuur 5.7.4 *Screenshot SAST fdroid-client*

Dit is een vulnerability die zou kunnen leiden tot kwetsbaarheden voor de gebruiker.

XML eXternal Entity injection (XXE), is een type attack tegen een applicatie die XML input parsed (verwerkt). Desbetreffende attack kan voorkomen wanneer onvertrouwde XML input een reference bevat naar een *external entity* (buitenstaande url bijvoorbeeld) en vervolgens wordt verwerkt door een slecht geconfigureerde XML parser. Deze attack zou kunnen leiden tot het uitlekken van gevoelige data of een denial of service (beschikbaarheid en vertrouwelijkheid) (OWASP Foundation, z.d.-e).

Onderzoek XXE kwetsbaarheid
Er is een onderzoek uitgevoerd waarin de kwetsbaarheid in de applicatie getest is, dit onderzoek is uitgewerkt in het onderzoeksrapport, de resultaten hiervan zullen worden benoemd. Na dit onderzoek is te concluderen dat de module *Element Tree* niet kwetsbaar was voor de XXE attack, maar dat er wel degelijk kwetsbaarheden in de module zitten (waaronder de billions laugh attack). De melding van de SAST tool die gebruikt wordt is dus een false positive, aangezien deze aangeeft dat er een XXE vulnerability in de applicatie zit. Mogelijk is de Semgrep regel die in deze situatie getriggerd is verouderd en was een oudere versie van Element Tree wel kwetsbaar voor een XXE attack.

Wanneer iemand met kwade bedoelingen een aanpassing zou maken aan een XML bestand (een localization bestand) kan dit leiden tot een denial of service bij de machine van een developer van het project. De machine zou een korte tijd niet goed functioneren.

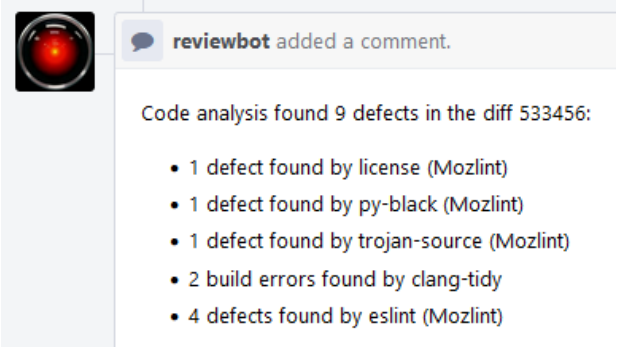
Oplossing
Deze kwetsbaarheid zou opgelost kunnen worden door in plaats van gebruik te maken van *Element Tree* gebruik te maken van *defusedxml*. Dit is een python module met fixes voor desbetreffende vulnerabilities (Python Software Foundation, z.d.).

5.7.2 Succesvolle afhandeling van gevonden issues

In deze subparagraaf zal worden gekeken naar de manier waarop gevonden issues in open-source projecten succesvol worden afgehandeld.

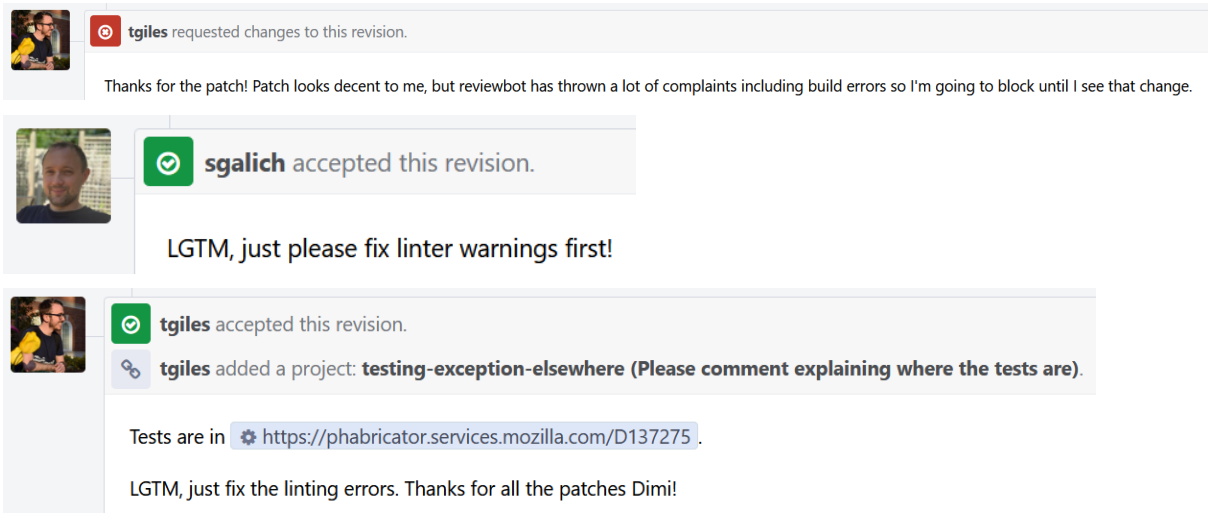
Voorbeeld **Firefox**

Zoals eerder genoemd (Subparagraaf 5.5.2) is gebruikt Mozilla een eigen bot (die gebruik maakt van DevSecOps tools) om hun pull requests en source code te controleren. In Figuur 5.7.5 is een voorbeeld te zien waarbij de bot een bericht achterlaat voor de developer. In dit bericht is te zien welke problemen gevonden zijn.



Figuur 5.7.5 *Screenshot reviewbot (Mozilla, z.d.-c)*

In Figuur 5.7.6 is te zien hoe er op dit bericht wordt gereageerd.



Figuur 5.7.6 *Screenshot differential (Mozilla, 2022)*

Vervolgens werden er nieuwe commits gepushed die de problemen die door de DevSecOps tools (in dit geval linters) worden aangegeven, oplossen. Hierna wordt er opnieuw gescand door de tools en is het probleem verholpen.

5.7.3 Conclusie

De impact van een gevonden kwetsbaarheid varieert en het is van belang op wat voor manier hier mee wordt omgegaan (Subparagraaf 5.7.1). De configuratie van de tooling moet niet achterwege gelaten worden (zoals te zien bij de Secret Detection melding bij prismo). In sommige open-source projecten lijken critical issues genegeerd te worden. Zo is er in de projecten prismo en fdroid-client bij sommige pull request een critical issue gevonden door een DevSecOps tool. Het is erg opmerkelijk dat hier verder niet op wordt gereageerd in desbetreffend pull request. Dit zou kunnen leiden tot kwetsbaarheden in de software. Wanneer de applicatie in gebruik zou worden genomen bij een gebruiker is deze mogelijk kwetsbaar voor attackers.

Bij een goed opgestelde CI/CD pipeline, waarbij het oplossen van gevonden issues vereist is, kunnen de problemen snel verholpen worden (Subparagraaf 5.7.2). Dit leidt tot een veiligere applicatie, in dit geval leidde het oplossen van de linter warnings tot een betere code quality (leesbaarheid en overzichtelijkheid van geschreven code).

Het blijkt dat het verschilt per open-source project op wat voor manier de gevonden issues worden afgehandeld. Bij sommige projecten wordt hier onmiddellijk op in gegaan en kan de pull request niet succesvol worden afgerond bij openstaande problemen, terwijl er bij sommige projecten niet op wordt gereageerd.

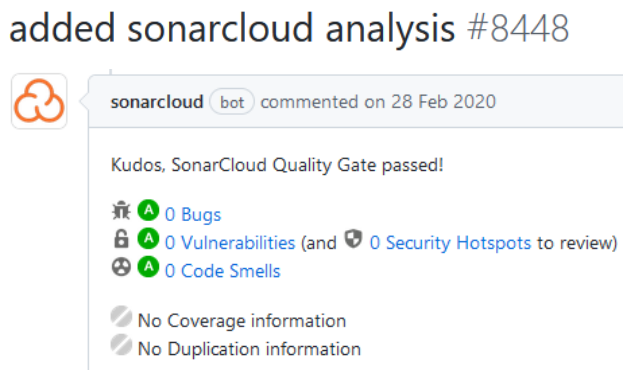
5.8 Transitie naar het in gebruik nemen van DevSecOps tools

Het in gebruik nemen van DevSecOps tools kan veel problemen met zich meebrengen, wanneer een organisatie over een grote code-repository beschikt. Dit heeft te maken met de manier waarop DevSecOps tooling werkt, wanneer er een SAST of Code Quality tool gerund zou worden geeft deze hoogstwaarschijnlijk een overweldigende lijst aan meldingen en

waarschuwingen. In deze paragraaf zijn de resultaten weergegeven van een kort onderzoek hoe hier bij open-source projecten mee wordt omgegaan. Er is gezocht naar projecten waarbij een tool in gebruik is genomen, vervolgens is er gekeken hoe de resulterende foutmeldingen van desbetreffende tool zijn opgelost.

Brave Browser

In sommige gevallen verloopt de transitie naar het gebruiken van een DevSecOps tool erg goed, dit was het geval bij *Brave Browser* die *SonarCloud* (SAST tool) in gebruik nam. Dit is te zien in desbetreffende pull request waar de tool in gebruik wordt genomen, zie Figuur 5.8.1.

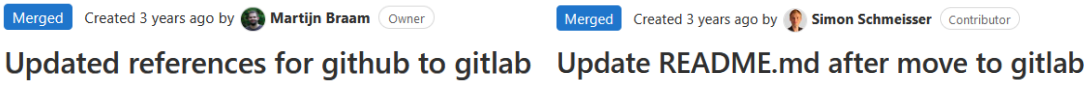


Figuur 5.8.1 *Screenshot Brave Browser pull request*

De tool heeft geen kwetsbaarheden gevonden in de repository.

pmbootstrap

Het komt voor dat wanneer open-source projecten wisselen van (bijvoorbeeld) Github naar GitLab er gebruik gemaakt wordt van de toolset die GitLab aanbiedt (SAST tooling bijvoorbeeld). Dit is te zien in het pull requests in Figuur 5.8.2.



Figuur 5.8.2 *Screenshot pmbootstrap pull request*

Wanneer deze tools in gebruik worden genomen kost het tijd om de initiële meldingen die de tooling aangeeft op te lossen. In dit geval gaf de linter *Flake8* veel meldingen die opgelost moeten worden, deze meldingen worden opgelost aan de hand van pull requests (5.8.3).

Figuur 5.8.3 *Screenshot pmbootstrap pull request*

Wanneer er wordt gekeken naar de tijd tussen pull requests is te concluderen dat een groep van 3 á 4 gebruikers ongeveer een halve week tijd heeft besteed aan het oplossen van de foutmeldingen (ruwe schatting).

Conclusie

Het verschilt per project hoe de overgang naar het in gebruik nemen van DevSecOps tooling gaat. Dit kan vlekkeloos gaan (in het geval van Brave Browser) of met de nodige aanpassingen en fixes aan de code om de meldingen van de desbetreffende tooling te verwerken (pmbootstrap).

5.9 Wat wordt er verstaan onder een ‘effectieve’ tool?

In deze paragraaf worden de resultaten weergegeven van een literatuurstudie naar wanneer een DevSecOps tool effectief is.
De term *effectief* houdt het volgende in: *‘waardoor **het beoogde doel bereikt wordt**; = doeltreffend’* (Van Dale, 2022a). Om deze vraag goed te kunnen beantwoorden is het van belang dat er duidelijk wordt gemaakt wat precies het **doel** van een DevSecOps tool is.

5.9.1 Doel van een DevSecOps tool

Zoals in Subparagraaf 5.3.2 en 5.3.3 is genoemd is het doel van *DevSecOps* om security problemen sneller te kunnen vinden (*shift left*) zodat bedrijven goedkoper uit zijn met het oplossen van desbetreffende problemen. Het doel van de DevSecOps tool **zelf** verschilt per *categorie* tooling, er zullen enkele voorbeelden worden geven, de complete lijst is te vinden in het onderzoeksrapport.

Software Composition Analysis (SCA)
Het doel van een SCA tool is om (externe) dependencies (libraries) die gebruikt worden in een applicatie te controleren op kwetsbaarheden. Het doel is bereikt wanneer de gevonden kwetsbare dependencies die voorkomen in de applicatie worden weergegeven aan de gebruiker (OWASP Foundation, z.d.-c).

Dynamic Application Security Testing (DAST)
Het doel van een DAST tool is om kwetsbaarheden te vinden in draaiende webapplicaties. Het doel is bereikt wanneer de gevonden kwetsbaarheden zijn weergegeven aan de gebruiker (OWASP Foundation, z.d.-c).

Static Application Security Testing (SAST)
Het doel van een SAST tool is om de source code van een applicatie te scannen op bekende kwetsbaarheden. Het doel is bereikt wanneer de resultaten van de uitgevoerde test zijn weergegeven aan de gebruiker (OWASP Foundation, z.d.-c).

Conclusie

Om het doel van DevSecOps tools te bereiken zijn de volgende twee stappen nodig:

1. Uitvoeren van scan of test
2. Resultaten presenteren aan de gebruiker

Wanneer dit succesvol is uitgevoerd is het **doel** van de tool bereikt.

Het is van belang deze twee stappen te beoordelen bij DevSecOps tooling, hieruit zijn de volgende twee deelvragen op te stellen:

Wanneer heeft een DevSecOps tool zijn scan of test succesvol uitgevoerd?
Wanneer presenteert een DevSecOps tool de resultaten goed aan de gebruiker?

Deze deelvragen zijn verder onderzocht, de resultaten hiervan zullen worden behandeld in de hieropvolgende subparagrafen.

5.9.2 Wanneer heeft een DevSecOps tool zijn scan of test succesvol uitgevoerd?

Een DevSecOps tool heeft onder andere als doel alle kwetsbaarheden (in het geval van Secret Detection alle secrets) te vinden in een project of applicatie. De meeste tools adverteren en beweren dat ze alle kwetsbaarheden afvangen, dit zou getest moeten worden met een PoC. In dit PoC zouden kwetsbaarheden gezet moeten worden, vervolgens moet er gecontroleerd worden of de tooling alles afvangt.

5.9.3 Wanneer presenteert een DevSecOps tool de resultaten goed aan de gebruiker?

In dit subparagraaf worden een aantal belangrijke elementen behandeld die van belang zijn bij het presenteren van test of scan resultaten aan de gebruiker. Verder wordt er ingegaan op de false positive rate en false negative rate. Dit zijn resultaten van een literatuurstudie.

De gebruiker

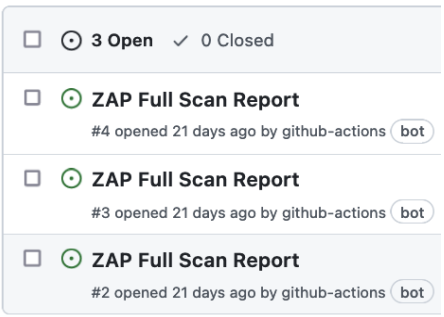
De gebruiker van de DevSecOps tool zou een klant van Computest kunnen zijn, bijvoorbeeld een groep developers die aan een applicatie werkt.

User interface

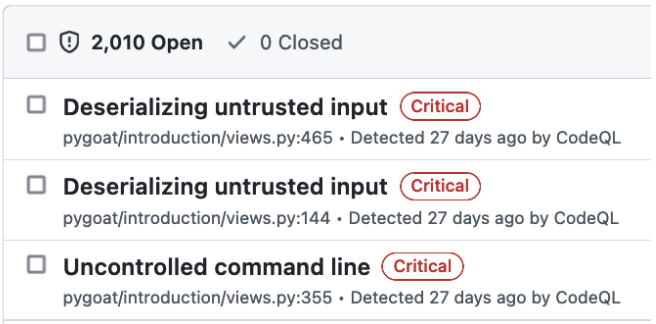
De user interface, ofwel gebruikersomgeving, is erg van belang wanneer er wordt gewerkt met security tools. Aangezien de eindgebruiker van de tool (developers of security specialisten) de gevonden problemen of kwetsbaarheden zullen moeten oplossen of zullen moeten doorgeven aan een andere partij die desbetreffend probleem oplost. De user interface valt onder **efficiëntie**, aangezien het tijd kan besparen wanneer een probleem op een overzichtelijke manier wordt weergegeven aan de gebruiker. Er zijn een aantal verschillende onderdelen van user interface, deze zullen worden beschreven (uitgebreidere resultaten met screenshots van voorbeelden zijn te vinden in het onderzoeksrapport).

Integratie versiebeheer tooling (Github/GitLab)

Het is mogelijk om DevSecOps tooling te integreren met de gebruikte versiebeheer tool (Github/GitLab), dit leidt tot een duidelijk overzicht van de gevonden kwetsbaarheden. Het is aan de developer van de DevSecOps tooling om deze integratie op te zetten, een voorbeeld van het resultaat van desbetreffende integratie (in dit geval *Github Issues* & *Security tabs) :*



Figuur 5.9.1 *Screenshot Github issues tab*



Figuur 5.9.2 *Screenshot Github Security tab*

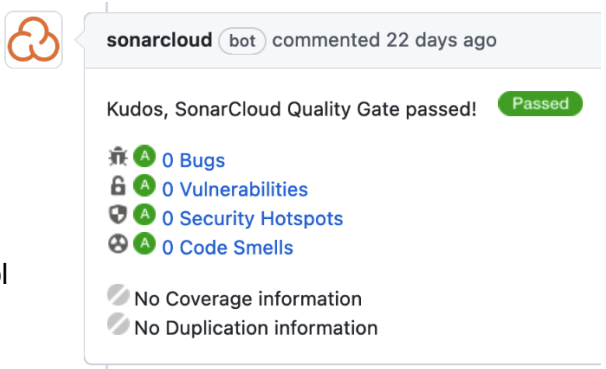
Dit type integratie brengt veel voordelen met zich mee, het biedt niet alleen een overzichtelijke weergave van de gevonden kwetsbaarheden, maar biedt verdere opname in

het Github platform. Hiermee wordt bedoeld dat er van een gevonden kwetsbaarheid, bijvoorbeeld *Deserializing untrusted input*, een *Issue* aangemaakt kan worden. Vervolgens kan er op dit issue gediscussieerd worden door leden van de repository en kunnen er onder andere leden aan toegevoegd worden. Deze integratie ondersteund dus een workflow in Github om zo op een efficiënte manier een gevonden kwetsbaarheid op te lossen.

Reacties/bot die berichten plaatst op pull requests

Enkele tools bieden de mogelijkheid om een bot te implementeren in de repository als lid, deze bot zal na afloop van een scan van een pull request een reactie achterlaten. In deze reactie wordt kortbondig weergegeven of er problemen gevonden zijn in de aanpassingen die gemaakt zijn door de developer. Vervolgens wordt er doorverwezen naar het dashboard van de tool om de gevonden problemen in detail te kunnen inzien.

Dit levert een efficiënte manier om aanpassingen aan de code te controleren in een oogopzicht. Zie Figuur 5.9.3 voor een voorbeeld.



Figuur 5.9.3 *Screenshot SonarCloud*

'Leesbaarheid' meldingen/issues

De manier waarop kwetsbaarheden worden voorgelegd aan de gebruiker is erg van belang, het verschilt per tool op wat voor manier dit wordt gedaan. Een aantal kenmerken zijn hierbij van belang

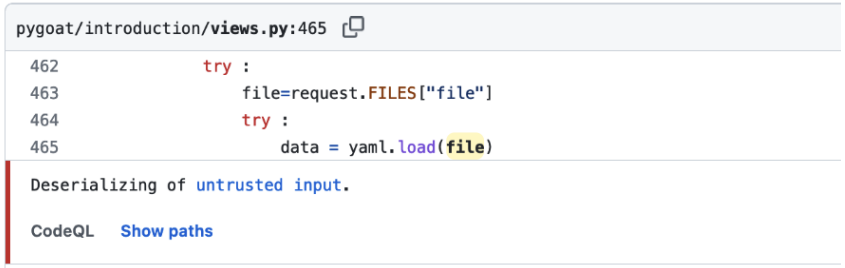
- CWE nummer
- Locatie van kwetsbaarheid in de code
- Het is in een oogopzicht duidelijk wat de kwetsbaarheid is en waar in het project deze zich plaatsvindt

De lengte van de tekst in de titel van de aangemaakte issues is van belang, wanneer deze zó lang is dat het niet direct duidelijk is wat het probleem is kan dit leiden tot lagere efficiëntie. Een voorbeeld van goede leesbaarheid is te zien in Figuur 5.9.4.

Github Code Scanning (SAST)

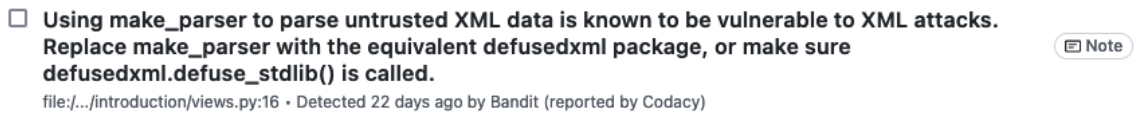
Figuur 5.9.4 *Screenshot CodeQL melding*

Er is direct duidelijk om wat voor type kwetsbaarheid het gaat, in dit geval gaat het om het deserialiseren van onvertrouwde input. Verder is te zien dat het een *Critical* kwetsbaarheid is en op welke plek in het project deze zich bevind. Verder staat er een stuk uitleg (inclusief CWE code) en wordt gevisualiseerd waar de kwetsbaarheid precies zit (Figuur 5.9.5).



Figuur 5.9.5 *Screenshot CodeQL melding*

Een voorbeeld van een minder duidelijke melding (waarbij het niet in een oogopzicht duidelijk is wat de kwetsbaarheid is) is te zien in Figuur 5.9.6.
Bandit (Codacy SAST)



Figuur 5.9.6 *Screenshot Bandit melding*

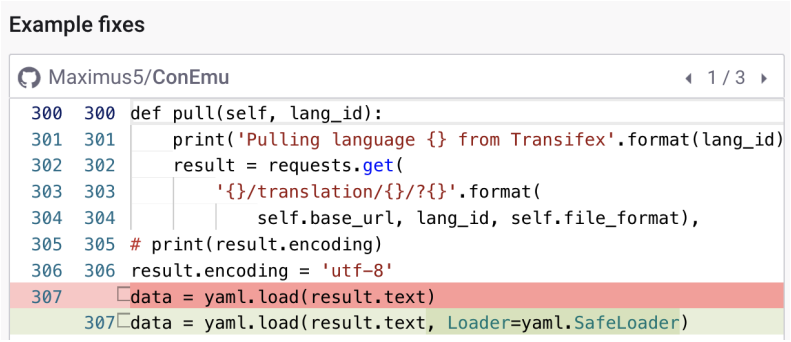
De titel bevat een grote hoeveelheid tekst, in een lijst met meerdere soortgelijke meldingen wordt dit snel onoverzichtelijk. Ook worden de kwetsbaarheden niet getagged met bijvoorbeeld een *'Critical'* tag, het is niet meteen duidelijk welke kwetsbaarheden van groot belang zijn. Ook ontbreekt de CWE code en is de locatie van de kwetsbaarheid in de gedetailleerde weergave niet beschikbaar.

Dashboard van tool zelf

Het komt bij veel DevSecOps tools voor dat er een eigen dashboard beschikbaar is om de gevonden kwetsbaarheden te kunnen inzien. Dit zorgt voor een duidelijk overzicht, alle kwetsbaarheden kunnen op één lokale plek bekeken worden. Verder wordt er in sommige gevallen de mogelijkheid geboden deze kwetsbaarheden te koppelen aan leden van de repository. Een voorbeeld van een tool die een eigen dashboard aanbiedt is *Snyk (SAST, SCA en Container Scanning)*. Een dashboard zoals deze van Snyk brengt veel voordelen met zich mee, alle kwetsbaarheden zijn op één plek overzichtelijk te zien. Verder wordt er in detail beschreven waar in het project de kwetsbaarheid zich bevindt en hoe deze potentieel op te lossen is. Dit leidt tot een hogere efficiëntie.

Suggesties voor het oplossen van kwetsbaarheden

Enkele tooling biedt de mogelijk om suggesties te geven voor het oplossen van een kwetsbaarheid, sommige tooling beschikt over de mogelijkheid om hier zelf een pull request voor aan te maken. Een voorbeeld hiervan is *Snyk*, zie Figuur 5.9.7.



Figuur 5.9.7 *Screenshot Snyk suggestion*

In dit voorbeeld worden verschillende oplossingen aangeboden voor de kwetsbaarheid, dit kan de gebruiker op weg helpen met het vinden van een mogelijke oplossing.

Figuur 5.9.8 *Screenshot Snyk pull request*

In Figuur 5.9.8 is er automatisch een pull request aangemaakt voor het oplossen van een kwetsbaarheid in een gebruikte package (SCA). In de file changes is dan ook te zien dat er naar een nieuwere (veilige) package wordt ge-upgrade. Dit kan de gebruiker tijd besparen, dit type suggesties en automatiseringen zijn erg efficiënt.

False positive rate en False negative rate

Een belangrijk element van een DevSecOps tool is de **false positive rate (FPR)**.
De false positive rate is de proportie van elk moment wanneer een test een **positief** resultaat levert wanneer deze **negatief** zou moeten zijn (Google, 2020).

In de context van cyber security is de FPR wanneer de testing tool aangeeft dat er een kwetsbaarheid is in de code, wanneer deze er eigenlijk **niet** is. Dit kan vervelend zijn voor de gebruiker en kan bij een té hoge hoeveelheid potentieel tot de volgende dingen leiden:

- De tool wordt genegeerd in verband met de grote hoeveelheid false positives
- De gebruiker wordt overladen met meldingen en is té veel tijd kwijt aan het doornemen van de resultaten

Een ander belangrijk element van een DevSecOps tool is de **false negative rate (FNR).**
De false negative rate is de proportie van elk moment wanneer een test een **negatief** resultaat levert wanneer deze **positief** zou moeten zijn. In de context van cyber security is dit wanneer een testing tool aangeeft dat er geen kwetsbaarheid is in de code, wanneer deze er eigenlijk **wel** is. In principe houdt dit in dat de tool dus gefaald heeft een kwetsbaarheid te detecteren. Dit kan potentieel erg gevaarlijk zijn, afhankelijk van de severity van de kwetsbaarheid (severity is te bepalen aan de hand van de CVE score). In verhouding tot de FPR is de FNR in principe ‘gevaarlijker’, waar de FPR leidt tot irritatie bij de gebruiker kan een false negative leiden tot een gevaarlijke situatie. Een situatie waarbij de klant een applicatie geleverd krijgt met een kwetsbaarheid.

Er is een onderzoek gedaan naar FPR en FNR en hoe deze gebruikt kan worden om DevSecOps tooling te beoordelen. Parameters die van pas komen bij het beoordelen van DevSecOps tooling (waar de FPR en FNR in is verwerkt) zijn Precision, Recall en de F-Score. Deze parameters zullen worden behandeld in Paragraaf 5.10.

5.10 Onderzoek FPR en FNR

5.10.1 Onderzoeksmethode

Er is onderzocht hoe de parameters FPR en FNR gebruikt kunnen worden om DevSecOps tooling te kunnen beoordelen. Er is besloten om verder onderzoek uit te voeren naar het scoren van tooling aan de hand van het aantal false positives en false negatives. In deze paragraaf zullen een aantal parameters worden toegelicht en behandeld die waarde leveren aan het kunnen scoren van DevSecOps tooling. Dit zal worden gedaan aan de hand van een literatuurstudie.

Literatuur
In dit onderzoek is documentatie van NIST geraadpleegd, The National Institute of Standards and Technology (NIST) is opgericht in 1901 en is onderdeel van de U.S. Department of Commerce. Relevante projecten waar NIST zich mee bezig houdt is een Cybersecurity Framework en verschillende security standaarden.

In 2011 heeft NIST een methodologie opgesteld voor het analyseren van Statische Analyse tools (SAST tooling). Echter kan deze methodologie ook gebruikt worden voor het

analyseren van andere DevSecOps tooling, aangezien andere tooling ook false positives en false negatives hebben in zijn resultaten. Hoewel de documentatie dateert uit 2011 is de opgezette methode nog steeds relevant, de tooling is inmiddels gemoderniseerd maar de metrics die kunnen worden afgeleid van de resultaten zijn onveranderd (bijvoorbeeld false positives of false negatives) (NIST, 2022).

Verder is een artikel van Snyk behandeld, Snyk is een bedrijf wat werkt aan verschillende DevSecOps tooling die eerder in het project aan bod is geweest.

5.10.2 Parameters

In deze subparagraaf zullen een aantal parameters worden behandeld die te maken hebben met de waardes uit Tabel 5.10.1.

Waarde	Definitie
True Positive (TP)	Tool vermeldt **correct** een kwetsbaarheid die zich bevindt in de code
False Positive (FP)	Tool vermeldt **incorrect** een kwetsbaarheid, de kwetsbaarheid bevindt zich **niet** in de code
False Negative (FN)	Tool heeft geen vermelding. Een kwetsbaarheid die zich bevindt in de code is **niet** gevonden door de tool

Tabel 5.10.1 *Parameters*

Precision

Met precision (*positief voorspellende waarde*) wordt het volgende bedoeld: de verhouding van kwetsbaarheden vermeld door een tool ten opzichte van de werkelijke kwetsbaarheden in de applicatie. Dit wordt gedefinieerd als de aantal kwetsbaarheden wat correct is vermeld door de applicatie (True Positives) gedeeld door het totaal aantal kwetsbaarheden wat vermeld is door de tool (True Positives + False Positives) (NSA, 2011).

$$Precision = \frac{\#\,TP}{\#\,TP + \#\,FP}$$

Precision is synoniem met de True Positive Rate (TPR) en een aanvulling op de False Positive Rate (FPR). In deze methodologie geeft Precision weer hoe goed een tool kwetsbaarheden kan identificeren. Stel dat een tool 40 meldingen heeft (False Positives en True Positives), waarvan 10 werkelijke kwetsbaarheden (True Positives), dan is de Precision 10 uit de 40, dus 0.25 (NSA, 2011).

Precision helpt om weer te geven hoeveel ‘vertrouwen’ een tool gegeven kan worden in het identificeren van kwetsbaarheden. Een tool met een Precision van 1 vermeldt alleen werkelijke kwetsbaarheden (True Positives). Een tool met een Precision van 0 vermeldt alleen incorrect kwetsbaarheden (False Positives) (NSA, 2011).

Verder betekent een hogere Precision minder *‘noise’* (irrelevante meldingen). Noise is ook een factor wat gebruikers (klanten van Computest) zou weerhouden gebruik te maken van de tooling (Snyk, 2021).

Recall (Sensitivity)
De Recall parameter (Sensitivity) representeert het gedeelte van werkelijke kwetsbaarheden vermeld door de tool. Recall is gedefinieerd als het aantal werkelijke kwetsbaarheden vermeld (True Positives) gedeeld door het totaal aantal werkelijke kwetsbaarheden, vermeld of niet vermeld, die zich bevinden in de code (True Positives + False Negatives) (NSA, 2011).

$$Recall = \frac{\#\,TP}{\#\,TP + \#\,FN}$$

Recall is altijd een waarde groter of gelijk aan 0 en kleiner dan of gelijk aan 1. Bijvoorbeeld een tool die 10 werkelijke kwetsbaarheden vermeld terwijl de code 20 kwetsbaarheden bevat heeft een Recall van 10 van de 20, 0.5 (NSA, 2011).

Een hoge Recall betekent dat de tool een hoog aantal werkelijke kwetsbaarheden identificeert. Bijvoorbeeld een tool met een Recall van 1 vermeld elke werkelijke kwetsbaarheid in de code. De tool heeft geen False Negatives. Tegenovergesteld vermeld een tool met een Recall van 0 geen werkelijke kwetsbaarheden (NSA, 2011)

Des te hogere Recall een tool heeft, des te beter de visibility en bescherming de tool biedt. Echter leidt een hogere Recall ook tot meer meldingen, maar wanneer deze gekoppeld is met een hoge Precision, zullen de meldingen relevant zijn (NSA, 2011).

Recall kan alleen worden berekend wanneer het bekend is hoeveel en welke kwetsbaarheden zich bevinden in de code (Snyk, 2021).

F-Score (Accuracy)
Als toevoeging op de Precision en Recall parameters is de F-Score berekend door het harmonisch gemiddelde te berekenen van de Precision en Recall waardes. Sinds het een type gemiddelde is zal de waarde van de F-Score altijd tussen Precision en Recall vallen (tenzij Precision en Recall gelijk zijn, dan zal de F-Score hier gelijk aan zijn) (NSA, 2011).

De F-Score biedt een richting in het identificeren van een goede DevSecOps tool door te berekenen hoeveel kwetsbaarheden gevonden zijn (True Positives) en hoeveel noise (False Positives) is geproduceerd (NSA, 2011). De F-Score wordt als volgt berekend:

$$F\,Score = 2 * \left(\frac{Precision * Recall}{Precision + Recall}\right)$$

Een harmonisch gemiddelde is wenselijk aangezien het verzekert dat een tool goed moet presteren met zowel een goede Precision als Recall parameter. Wat hiermee wordt bedoeld wordt is dat een tool geen hoge F-Score zal krijgen wanneer deze hoog scoort in één parameter maar laag in een andere parameter. Een tool die bijvoorbeeld slecht scoort op één parameter en hoog op de andere zal geen hogere F-Score krijgen die een tool die gemiddeld scoort op beide parameters (NSA, 2011).

De F-Score is uiteindelijk een manier om de Accuracy van een test te meten.

5.11 Wat wordt er verstaan onder een ‘efficiënte’ tool?

De term *‘efficiënt’* houdt het volgende in: *‘Zó dat het de minste middelen, inspanning enz. kost; = doelmatig* (Van Dale, 2022b)
Om deze vraag goed te kunnen beantwoorden is het van belang dat er duidelijk wordt hoeveel **inspanning, tijd en middelen** het kost om de categorie DevSecOps tool in te stellen en in gebruik te nemen.

5.11.1 Inspanning en tijd

Het verschilt per categorie DevSecOps tool hoeveel inspanning en tijd er nodig is om deze op te stellen. Uit de steekproef die genomen is in Paragraaf 5.6 blijkt dat de statische testing tools het meest gebruikt worden, hieruit zou ook opgenomen kunnen worden dat deze het makkelijkst in gebruik te nemen zijn.

Linter, SAST, SCA, Secret Detection, Container Scanning
Deze categorieën aan tools zijn relatief makkelijk op te zetten, dit komt mede doordat deze onder **‘Static’** testing vallen, de source code, container images en libraries/componenten worden gescand.
De moeilijkheid bij deze tooling ligt het bij het afstellen van de parameters zodat de FPR (Subparagraaf 5.9.3) niet te hoog ligt. Voordat hier meer over gezegd kan worden zou dit meegenomen moeten worden in een PoC.

DAST
Deze categorie tooling is relatief moeilijk op te zetten, dit komt mede doordat deze onder **‘Dynamic’** testing valt, er wordt getest op een draaiende applicatie. Dit kan lastig zijn om op te zetten.
Echter zou er voordat er een conclusie over getrokken kan worden een PoC opgezet moeten worden, hierin zouden de verschillende categorieën tools vervolgens ingesteld en getest moeten worden. Uit het PoC zou blijken of de theorie (op papier) en de praktijk overeenkomen en dus tot dezelfde conclusie zullen leiden of niet.

Met inspanning wordt de ‘moeilijkheidsgraad’ van het opzetten van de tool bedoeld, hier zal in Hoofdstuk 7 en 8 verder op worden toegelicht.

5.11.2 Middelen

Wanneer er over middelen gesproken wordt bij DevSecOps tooling kan men nadenken over de hoeveelheid geld die een desbetreffende tool zou kosten. Een groot gedeelte van de gevonden tools in Subparagraaf 5.5 zijn commercieel en bieden verschillende pakketten aan. Echter zijn er ook een aantal open-source tools die volledig gratis in gebruik te nemen zijn.
Hier zou de volgende vraag over opgesteld kunnen worden:

- *‘Hoe wegen open-source (gratis) tools op tegen commerciele tools?’*

5.12 Conclusie

Wat houdt een CI/CD security tool in? & Wat valt er onder CI/CD security tooling?
Na de vragen op te delen in subvragen en deze te beantwoorden aan de hand van een literatuurstudie is het duidelijk wat een CI/CD security tool inhoudt en wat hieronder valt (de resultaten van deze onderzoeken zijn te vinden in Paragraaf 5.3 en 5.4).

Welke security tools worden er gebruikt in de CI/CD pipelines van open-source-gemeenschappen?
Na het uitvoeren van literatuurstudies en steekproeven is er een opsomming gemaakt waarin de gebruikte DevSecOps tools van de grootste open-source projecten te zien zijn.

Er blijken weinig open-source projecten te zijn met uitgebreide CI/CD pipelines waarbij veel security tools worden gebruikt. De meeste open-source projecten gebruiken enkel de basis tools (basis checks van Github bv. *CodeQL*).

De meeste open-source projecten op Github maken gebruikt van de built-in Github security tools, waaronder CodeQL + Dependabot. Hiernaast wordt er in de meeste gevallen gebruik gemaakt van een linter die past bij de gebruikte programming language, in enkele projecten wordt er gebruik gemaakt van een SAST tool (blijkt uit de steekproeven uit Paragraaf 5.5 en 5.6).

De meeste open-source projecten gehost op GitLab maken gebruik van de (uitgebreide) selectie aan built-in security tools van GitLab. In enkele projecten wordt er gebruik gemaakt van andere third party tools.

De informatie die is opgedaan in Paragraaf 5.9 en 5.11 bij het beantwoorden van de onderzoeksvragen: ***'Wat wordt er verstaan onder een 'effectieve' tool?'*** en ***'Wat wordt er verstaan onder een 'efficiënte' tool?'*** is erg van belang in fase 4. Uit deze paragraaf kunnen test parameters worden opgedaan die van pas komen voor het vergelijken van verschillende tooling.

Test parameters die meegenomen zullen worden zijn als volgt:

- User interface (aanwezigheid hiervan, gebruiksvriendelijkheid)
- Precision, Recall & F-Score (Accuracy)
- Tijd/inspanning opzetten tool
- Middelen (open source, commercieel)

Deze test parameters zullen in Hoofdstuk 8 verder worden uitgewerkt, de scoring zal onder andere worden toegelicht.

De deelvraag ***'Wat wordt er verstaan onder een 'efficiënte' tool?'*** zal ook worden meegenomen naar de volgende fase (Hoofdstuk 6 en 7). Hier zal de deelvraag '*Hoeveel inspanning/tijd kost het om de categorie tooling in/op te stellen?'* verder worden uitgewerkt.

6. Ontwerp

In dit hoofdstuk zal het ontwerp van het proof of concept worden toegelicht. Een PoC wordt gebruikt om te verifiëren dat een concept praktische mogelijkheden heeft (WebFinance Inc., 2016), in dit geval bestaat het PoC uit een opzet om de praktische mogelijkheden van verschillende DevSecOps tools aan te tonen. Dit hoofdstuk is onderverdeeld in de delen **Doel** en **Opzet.**

6.1 Doel PoC

Het doel van het PoC is om de theoretische onderbouwing op de deelvragen uit Paragraaf 5.11 te onderbouwen. De volgende deelvragen worden bedoeld:

Wat wordt er verstaan onder een 'efficiënte' tool?

- *Hoeveel inspanning/tijd kost het om de categorie tooling in/op te stellen?*
- *Wat voor middelen kost het om de categorie tooling in/op te stellen?*

Verder wordt het PoC gebruikt om de verschillende DevSecOps tools te kunnen beoordelen, de manier waarop deze tools worden beoordeeld zal worden toegelicht in Hoofdstuk 8. Met tools worden hier de specifieke tools bedoeld zoals bijvoorbeeld SAST tool *Codacy* (Subparagraaf 5.4.1). Het zal noodzakelijk zijn specifieke test cases op te stellen per categorie tooling, deze test cases zullen worden opgesteld in Hoofdstuk 8. De parameters uit Paragraaf 5.12 zullen worden gebruikt in deze test cases.

Uiteindelijk zal aan de hand van het PoC de volgende stappen genomen kunnen worden

1. Opzetten DevSecOps tool
2. Uitvoeren scan
3. Scoren van tool aan de hand van scan-resultaten en test cases

6.2 Opzet PoC

Om een PoC op te zetten zijn er 3 componenten nodig:

1. CI/CD pipeline software
2. Test repository
3. DevSecOps tools

Deze componenten zullen in deze paragraaf behandeld worden.

6.2.1 CI/CD pipeline software

Er is gekozen om een PoC op te zetten aan de hand van Github, aangezien het een veel gebruikt versiebeheer systeem is wat gratis CI/CD mogelijkheden aanbiedt voor open source repositories. Verder is er gekozen een PoC op te zetten met behulp van GitLab, ook GitLab wordt veel gebruikt en biedt veel gratis CI/CD mogelijkheden aan.

Github Actions

Op repositories op Github wordt gebruik van *Github Actions* gratis aangeboden, wanneer de repository openbaar is. Github Actions is het CI/CD pipeline systeem van Github.

6.2.2 Test repository

Naast een CI/CD pipeline is er een repository nodig waar DevSecOps tools hun scans en tests op uit kunnen voeren. Het is handig om een repository te gebruiken die verschillende kwetsbaarheden bevat.

Voor het vinden van kwetsbare repositories is de Vulnerable Web Applications Directory van OWASP geraadpleegd, dit project is een register met bekende kwetsbare web en mobiele applicaties wat onderhouden wordt door OWASP. Deze applicaties zijn onder andere bestemd voor penetration testers die bijvoorbeeld hacking tools willen uitproberen. Deze applicaties zijn ideaal voor het testen van DevSecOps tools aangezien ze bekende kwetsbaarheden bevatten (OWASP Foundation, z.d.-d).

PyGoat
PyGoat is een opzettelijke kwetsbare web applicatie opgezet met django, de kwetsbaarheden in de repository zijn gebaseerd op de OWASP top 10 (OWASP Foundation, z.d.-a).

WrongSecrets
WrongSecrets is een applicatie die gebruikt wordt om te leren hoe er **niet** met secrets omgegaan moet worden in een repository of applicatie (OWASP Foundation, z.d.-b).

Door gebruik te maken van dit soort repositories ben je ervan verzekerd dat de DevSecOps tools voldoende kwetsbaarheden hebben om te vinden. Op beide repositories zal in Hoofdstuk 8 verder in detail worden ingegaan.

6.2.3 DevSecOps tools

Om de volgende deelvragen te beantwoorden is het van belang om elke **categorie** DevSecOps tooling op te zetten:

- *Hoeveel inspanning/tijd kost het om de* **categorie** *tooling in/op te stellen?*
- *Wat voor middelen kost het om de* **categorie** *tooling in/op te stellen?*

De volgende categorieën aan tools zullen dus opgezet moeten worden:

- Linter
- SAST
- SCA
- Secret Detection
- Container scanning
- DAST

Van elke categorie DevSecOps tooling zal er een selectie aan tools worden uitgekozen die vervolgens opgezet zullen worden in het PoC. Zoals eerder genoemd is (Paragraaf 3.4) ligt de focus in het project op tooling die al gebruikt wordt in andere open-source projecten. Er is gekozen voor de categorieën aan tools die zijn voorgekomen bij open-source projecten (Paragraaf 5.6).

7. Implementatie

In dit hoofdstuk worden de resultaten van het implementeren van het PoC behandeld, de gedetailleerde stappen (inclusief screenshots) zijn te vinden in het implementatierapport. Zoals benoemd is in Subparagraaf 6.2.3 zijn er van elke categorie tooling een aantal tools opgezet. Er is geen specifieke keuze gemaakt voor tooling, de tooling die genoemd wordt in Paragraaf 7.1 dient enkel als voorbeeld/test om het PoC op te zetten.

7.1 Opgezette tooling

De volgende tools zijn geïmplementeerd in het PoC:

	PoC - Github	PoC - GitLab
Linter	Super-Linter	Pylint, flake8
SAST	CodeQL, Codacy, SonarCloud, Snyk	GitLab SAST
SCA	Dependabot, Snyk	OWASP Dependency Check
Secret Detection	Github Secret Scanning, Gitleaks, GitGuardian	GitLab Secret Detection
Container scanning	Snyk, Trivy	Snyk
DAST	OWASP Zap, StackHawk	StackHawk

Tabel 7.1.1 *PoC tooling*

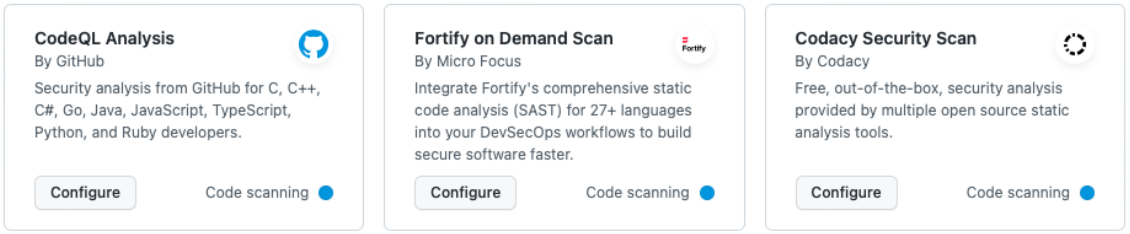
7.2 PoC Github

Github Actions

Voor het eerste PoC is er een mirror (kopie) gemaakt van de kwetsbare repository genaamd *PyGoat* (onderhouden door OWASP). Vervolgens zijn Github Actions voor deze repository aangezet en is er gekeken naar de beschikbare cyber-security gerelateerde actions.

Onder Github actions is een lijst met kant en klare actions beschikbaar, hieronder stonden onder andere de volgende tools:

- CodeQL
- Codacy Security Scan
- Semgrep/Brakeman

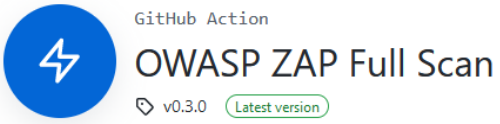


Figuur 7.2.1 *Screenshot Github actions*

Er zal een voorbeeld gegeven worden van de implementatie van een tool, de overige implementaties zijn te vinden in het implementatierapport.

OWASP ZAP

Het opzetten van OWASP ZAP is gedaan aan de hand van een beschikbare Github Action.



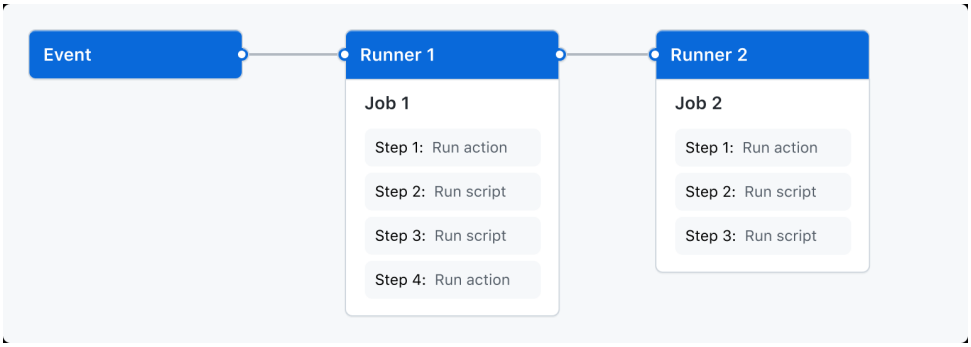
Figuur 7.2.2 *Screenshot OWASP action*

Deze action maakt gebruik van de *Full Scan* van OWASP ZAP, dit type scan voert een actieve scan uit op een draaiende applicatie. Het opzetten hiervan heeft best veel tijd gekost aangezien de volgende stappen genomen moeten worden in de Github Action:

1. Bouwen en draaien van de applicatie die getest moet worden (in dit geval *PyGoat*)
2. Uitvoeren van de OWASP ZAP scan op de draaiende applicatie
3. Weergeven en opslaan van de resultaten van scan

De OWASP ZAP Full Scan action die beschikbaar is gesteld regelt alleen stap 2 en 3, stap 1 moet zelf opgezet worden.

Runners van Github werken als volgt:



Figuur 7.2.3 *Screenshot Github Runners (Github, z.d.-a)*

Er wordt een nieuwe runner (virtuele machine) aangemaakt voor elke aparte *Job*, het is van belang dat stappen 1-3 uitgevoerd worden op dezelfde *runner*. Dit aangezien de OWASP ZAP Scan toegang moet hebben tot de draaiende applicatie (Github, z.d.-c).

Er is een workflow file opgezet genaamd *zapscan.yml* waarin de volgende stappen genomen worden: (zie implementatierapport voor meer details)

1. *De Github Action wordt gecalled op push en pull requests.*
2. *De Job draait op ubuntu, verder wordt de test ingesteld op python versie 3.6, 3.7 en 3.8.*
3. *Python wordt opgezet op de runner.*
4. *De applicatie PyGoat wordt opgezet (installeren van nodige dependencies).*
5. *De server wordt gestart als achtergrond proces.*
6. *Uiteindelijk wordt de ZAP Scan uitgevoerd op de target van de applicatie (localhost).*

Bij nieuwe commits en pull requests wordt de action (workflow) nu uitgevoerd en deze werkt naar behoren. De resultaten van de test worden door OWASP ZAP aangemaakt onder de tab *Issues* op Github, verder wordt er een erg gedetailleerd rapport gegenereerd wat gedownload kan worden (Figuur 7.3.1).

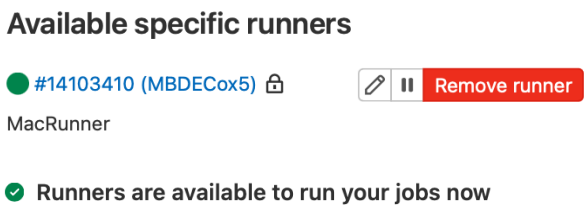
7.3 PoC GitLab

7.3.1 GitLab CI/CD

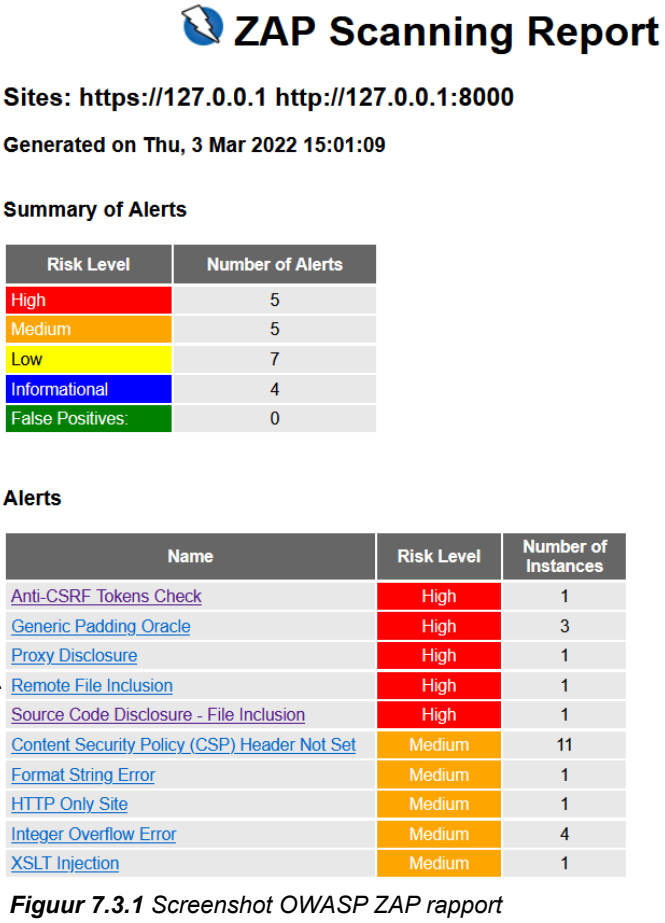
Opzetten GitLab Runner

Voor het opzetten van het PoC in GitLab was het noodzakelijk om een *GitLab Runner* op te zetten. Dit komt doordat er misbruik wordt gemaakt van de gratis CI/CD runners (machines) die GitLab aanbiedt, deze kunnen alleen gratis gebruikt worden bij verificatie met credit card. Een alternatief is het handmatig opzetten van een GitLab runner op een eigen machine, deze stap hoeft een klant van Computest niet te maken.

Met behulp van de documentatie van GitLab is er een GitLab Runner opgezet om de CI/CD stappen op uit te voeren, in plaats van deze op de *'Shared Runners'* die GitLab zelf levert uit te voeren. Er is nu te zien dat de runner actief is (Figuur 7.3.2).



Figuur 7.3.2 *Screenshot GitLab runner*



ZAP Scanning Report

Sites: https://127.0.0.1 http://127.0.0.1:8000

Generated on Thu, 3 Mar 2022 15:01:09

Summary of Alerts

Risk Level	Number of Alerts
High	5
Medium	5
Low	7
Informational	4
False Positives:	0

Alerts

Name	Risk Level	Number of Instances
Anti-CSRF Tokens Check	High	1
Generic Padding Oracle	High	3
Proxy Disclosure	High	1
Remote File Inclusion	High	1
Source Code Disclosure - File Inclusion	High	1
Content Security Policy (CSP) Header Not Set	Medium	11
Format String Error	Medium	1
HTTP Only Site	Medium	1
Integer Overflow Error	Medium	4
XSLT Injection	Medium	1

Figuur 7.3.1 *Screenshot OWASP ZAP rapport*

Er is een runner opgezet met als executor-type *Docker*, dit type runner maakt gebruik van Docker containers om de CI/CD pipeline op te draaien.

Repository

De repository die eerder is aangemaakt en gecloned naar Github is geïmporteerd naar GitLab, dit is de PyGoat vulnerable repository beheerd door OWASP. Deze zal ook gebruikt worden voor de PoC opzet in GitLab.

7.3.2 GitLab Auto Devops en Workflow

Auto DevOps

Om gebruik te maken van de security tools die worden aangeboden door GitLab kan er gebruik gemaakt worden van Auto Devops. Auto Devops is een collectie aan voor geconfigureerde functies en integraties, security tooling (die GitLab zelf aanbiedt) valt ook onder deze functies.

Het is mogelijk om de default stappen in de pipeline aan te passen, dit kan door de workflow file *.gitlab-ci.yml* aan te passen. Ook is het mogelijk de security tooling van GitLab te gebruiken in een eigen (custom) CI/CD pipeline, dit zou de voorkeursoptie zijn voor bedrijven die al een CI/CD pipeline gebruiken en hier security tools aan toe willen voegen. Wanneer je bijvoorbeeld de SAST scanning tool zou willen gebruiken kan je deze opnemen in de *.gitlab-ci.yml* workflow file.

Er is voor het PoC gebruik gemaakt van een eigen workflow file in plaats van de Auto DevOps functionaliteit, dit aangezien dit de meest logische optie zou zijn voor een bedrijf dat al gebruik maakt van GitLab, omdat er dan al gebruik gemaakt wordt van een eigen workflow file. De target user base van Auto DevOps is voor bedrijven en mensen die geen eigen workflow hebben en deze default template als basis kunnen gebruiken (GitLab, z.d.-a).

Workflow file
De manier waarop bij GitLab CI/CD ingesteld kan worden is vergelijkbaar met Github, er wordt gebruik gemaakt van een *yaml*-file genaamd *.gitlab-ci.yml*. In deze file staan de stappen die genomen worden in de CI/CD pipeline. Er is een standaard workflow file aangemaakt als test. Deze CI/CD stappen draaien op de eerder opgezette runner (lokaal op een Mac machine). Het is nu mogelijk andere security tooling in de workflow file op te nemen (GitLab, z.d.-c).

7.4 Resultaten

In deze paragraaf zullen de resultaten van het opzetten van het PoC worden behandeld, vervolgens zal er worden geconcludeerd door terug te blikken op het theoretisch onderzoek uit Paragraaf 5.11.

7.4.1 Statisch testen

Onder statisch testen vallen de volgende categorieën aan tooling: Linter, SAST, SCA, Secret Detection en Container Scanning.
Deze categorieën aan tooling blijken relatief makkelijk op te zetten, de volgende stappen moeten genomen worden voor het opzetten van deze categorieën aan tooling:

Github
1. Aanmaken Github Action (workflow file)
2. Enkele parameters invullen
    a. Wanneer scan getriggerd moet worden
    b. Eventueel de programmeertaal

GitLab
1. Aanmaken workflow file (*.gitlab-ci.yml*) (voor gebruik GitLab's eigen toolset kunnen template workflow files gebruikt worden)
2. Vereiste parameters invullen (en eventuele optionele parameters)

Bij tooling met eigen website/dashboard (bijvoorbeeld SonarCloud)
1. Koppelen Github account (autoriseren) aan repository
2. Enkele parameters invullen

Vervolgens worden de scans uitgevoerd op het aangegeven moment en worden de resultaten weergegeven via de interface van Github/GitLab of een dashboard.

7.4.2 Dynamisch testen

De volgende categorie tooling valt onder dynamisch testen: DAST.
Deze categorie aan tooling blijkt relatief moeilijk op te zetten, bij deze categorie aan tooling wordt een extra stap vereist ten opzichte van statisch testen. Dit is het bouwen en draaien van de applicatie die getest moet worden, dit kan complicaties teweeg brengen. Dit bleek ook uit het opzetten van het PoC. Er was ongeveer een dag besteed aan het werkend krijgen van bijvoorbeeld de DAST tool OWASP ZAP, uiteindelijk werkte deze tool wel naar behoren. Het beoordelen van een tool op de hoeveelheid tijd/moeite die het kost om deze op te zetten zal later terugkomen (8.6.3).

7.4.3 Overige bevindingen

Github Actions
Het integreren van nieuwe tools aan de hand van Github Actions (workflow files) is prettig in gebruik. Voor de meeste tools zonder native ondersteuning voor Github (ondersteuning van de developer van de tool) bestaat wel een Github action die de tool heeft geïmplementeerd.

GitHub dashboard integraties (issues / Security)
De dashboards die geïntegreerd zitten in de Github omgeving (*issues* en *Security*) zijn erg overzichtelijk en sluiten functioneel erg goed aan op het ecosysteem van Github. De mogelijkheden om hier bijvoorbeeld repository-members aan te koppelen is erg handig in gebruik.

7.5 Conclusie

Wat wordt er verstaan onder een 'efficiënte' tool?

- *Hoeveel inspanning/tijd kost het om de categorie tooling in/op te stellen?*

Over de bovenstaande deelvraag is in Subparagraaf 5.11.1 theoretisch onderzoek uitgevoerd. Hieruit volgde de verwachting dat statische DevSecOps tools relatief makkelijk op te zetten zouden zijn, ten opzichte van dynamische DevSecOps tools die moeilijker op te zetten zouden zijn.

De resultaten die zijn opgedaan tijdens het opzetten van het PoC komen overeen met de theoretische bevindingen van Subparagraaf 5.11.1.

Bruikbaarheid PoC
Het PoC wat in deze fase is een opgesteld is bruikbaar voor het uitvoeren van testen in fase 4, *testen*. Ook is er in deze fase veel kennis opgedaan over het opzetten van een PoC, mochten er aanpassingen moeten komen aan de repository of tools die getest moeten worden kan dit snel worden geïmplementeerd.

In fase 4 (Hoofdstuk 8 en 9) zullen verschillende specifieke DevSecOps tools met elkaar worden vergeleken op basis van parameters die in Hoofdstuk 5 zijn opgesteld.

8. Opzetten testen

8.1 Inleiding en onderzoeksmethode

In dit deel van het project is er onderzoek gedaan om de volgende deelvraag te kunnen beantwoorden: ***Welke voordelen hebben deze tools in de praktijk?***

In dit deel van het project staat testen centraal (fase 4), verschillende specifieke tools van verschillende categorieën zullen met elkaar worden vergeleken aan de hand van parameters. Dit zal verder worden toegelicht in de hierop volgende paragrafen.

Er wordt in dit deel van het project gebruikt gemaakt van de volgende onderzoeksresultaten uit voorafgaande tussenrapportage:

Onderzoeksrapport
De resultaten van de volgende onderzoeksvragen van het onderzoeksrapport zullen gebruikt worden of aan bod komen in deze fase:

- *Wat valt er onder CI/CD security tooling? (Paragraaf 5.4)*
- *Welke security tools worden er gebruikt in de CI/CD pipelines van open-source-gemeenschappen? (Paragraaf 5.5)*

De parameters die zijn opgesteld resulterend uit de onderzochte onderzoeksvragen zullen gebruikt worden in deze fase.

Ontwerprapport en implementatierapport
De PoC's die zijn opgesteld in fase 2 en 3 zullen worden gebruikt, waaronder het PoC aan de hand van Github en een PoC aan de hand van GitLab. Het belangrijkste wat wordt meegenomen is de kennis die is opgedaan voor het opspinnen van DevSecOps tooling, aangezien er veel gewisseld moet worden tijdens het testen en vergelijken van deze tools.

De volledige versie met uitgebreidere toelichting op de testcases en methodes is te vinden in het testrapport.

8.2 Waarom er getest wordt

De testfase (fase 4) is erg van belang, de resultaten uit voorgaande fases worden op de proef gesteld, aan de hand van de resultaten kunnen de deelvragen (en hierdoor ook de hoofdvraag) beantwoord worden.

Welke CI/CD security tools binnen open-source-gemeenschappen bewijzen het meest efficiënt/effectief te zijn in CI/CD pipelines?

- *Welke voordelen hebben deze tools in de praktijk?*
- *Welke moeilijkheden ondervinden open-source-gemeenschappen bij het gebruik van deze tools?*

Het resultaat van dit testrapport is een testmethode met templates die gebruikt kan worden voor het uitvoeren van de tests en vergelijking van de verschillende DevSecOps tooling.

8.3 Wat er getest wordt

8.3.1 Categorieën DevSecOps tools

In deze testfase is er van elke *categorie* DevSecOps tooling een selectie gemaakt die verder getest is. Met deze categorieën worden de volgende categorieën aan tools bedoeld:

- Linter
- SAST
- SCA
- Secret Detection
- Container scanning
- DAST

Met een selectie aan tooling wordt de specifieke tooling bedoeld, bijvoorbeeld *Snyk* of *Github Code Quality*.

Er is gekozen voor de categorieën die ook zijn voorgekomen in open-source projecten en naar boven zijn gekomen in de steekproef die is genomen (zie Paragraaf 5.6). De reden dat hiervoor gekozen is, is om de scope van de testfase te beperken, er is niet voldoende tijd beschikbaar om alle mogelijk categorieën te testen. Zoals in Paragraaf 3.4 is genoemd ligt de focus in het project specifiek op CI/CD security tooling die ook voorkomt in andere open source projecten.

8.3.2 Geselecteerde DevSecOps tools

De tools die geselecteerd zijn om te testen zijn de tools die naar boven zijn gekomen bij de literatuurstudie in de analysefase (Paragraaf 5.5) en tooling die behandeld is in de implementatie fase. De specifieke tools zullen in Paragraaf 8.5 behandeld worden.

8.3.3 Ondersteunde programmeertalen

Het verschilt per tool welke talen er ondersteund worden, dit is normaal gesproken op de website of repository van de tool (in geval van open source tooling) te vinden. De ondersteunde talen zullen vermeld worden bij de test case, maar zullen niet meetellen voor de beoordeling van de tool. Het ligt aan de gebruiker voor welke taal of talen ondersteuning nodig is in zijn use case.

8.4 Hoe er getest wordt

8.4.1 Keuze test methode

Er is een literatuurstudie uitgevoerd naar verschillende test en scoring methodes, uiteindelijk is er gekozen voor een *weighted scoring model*. Dit aangezien er een methode nodig was waarbij er een bepaalde waarde gegeven kan worden aan een parameter die belangrijker is dan een andere parameter.

8.4.2 Weighted scoring model

Een weighted scoring model (oftewel weighted scorecard) is een project management techniek die gebruikt kan worden voor het afwegen van keuzes. Waaronder bijvoorbeeld product kenmerken en features, in het geval van DevSecOps tooling is dit ideaal. Er kan een zwaarder *gewicht* gegeven worden aan een feature die meer waarde heeft dan een andere feature. Naar aanleiding van verschillende keuzes en features ontstaat er een scorekaart (Morpus, 2021a).

Stap 1: Opsommen van opties

De eerste stap in het proces is het opsommen van de opties die meegenomen worden. In het geval van dit project zijn dit de verschillende DevSecOps tools, alle eventuele kandidaten voor de tools worden meegenomen.

Stap 2: Parameters afstellen

Nu er een lijst aan opties is opgesteld worden de parameters bepaald. Deze parameters zijn in fase 1 opgesteld, deze zijn als volgt:

- User interface (aanwezigheid hiervan, gebruiksvriendelijkheid)
- Precision, Recall & F-Score (Accuracy)
- Tijd/inspanning opzetten tool
- Middelen (open source, commercieel)

Stap 3: *Weight* toekennen aan de parameters

Het verschilt hoeveel belang elke parameter heeft, dit is waar het weighted scoring model van pas komt. Er zal aan verschillende parameters een ander gewicht worden toegekend in de vorm van een percentage. Een voorbeeld van het toekennen van *weight* is als volgt:

Figuur 8.4.1 *Voorbeeld weight toekenning (Morpus, 2021b)*

In dit geval lijkt het alsof het gemiddelde is gepakt van de individuele scores, echter zijn deze als volgt *geweight* (voorbeeld):

- Ease of use (40%)
- Support (20%)
- Pricing (20%)
- Features (20%)

Door deze *weighting* heeft *Ease of use* in dit geval meer invloed op het eindresultaat.
In dit voorbeeld wordt er dus meer ‘waarde gehecht’ aan het hebben van een goede user experience ten opzichte van bijvoorbeeld de prijs.

Er wordt gestart met 100% en dit wordt onderverdeeld onder de parameters gebaseerd op de prioriteiten van datgene dat wordt gescoord.

Stap 4: Opzetten van weighted scoring grafiek

Nu de opties, parameters and weights zijn opgesteld kan er een weighted matrix worden opgezet, deze zou er als volgt uit kunnen zien:

	ROI	COST	EFFORT	SUSTAINABILITY	RISK	TIME	QUALITY
Weight	.20	.20	.15	.15	.10	.10	.10
Option 1	3	2	5	2	5	1	3
Option 2	2	2	4	2	5	2	1
Option 3	3	4	1	4	5	1	3
Option 4	1	1	5	2	3	4	5
Option 5	3	2	5	5	3	2	3

Tabel 8.4.2 *Voorbeeld weighted scoring model (Morpus, 2021c)*

(in dit geval is de *range* aan waardes 1-5)

Vervolgens kan er aan de hand van de opgesteld weights een totale score uitgerekend worden, in dit voorbeeld hebben *costs* een waarde van 20% (0.20). Wanneer dit is uitgerekend ziet dit er uiteindelijk als volgt uit:

	ROI	COST	EFFORT	SUSTAIN-ABILITY	RISK	TIME	QUALITY	TOTAL SCORE
Weight	.20	.20	.15	.15	.10	.10	.10	1.00
Option 1	3 * .20 = 0.6	2 * .20 = 0.4	5 * .15 = 0.75	2 * .15 = 0.3	5 * .10 = 0.5	1 * .10 = 0.1	3 * .10 = 0.3	0.42 * 100 = 42

Tabel 8.4.3 *Voorbeeld weighted scoring model (Morpus, 2021c)*

In Tabel 8.4.3 is te zien dat bijvoorbeeld de waarde 2 bij *cost* wordt vermenigvuldigd met de weight (0.20). De totale score is de som van de individuele parameters, in dit geval 42 / 100.

Het is in het geval van DevSecOps tooling echter lastiger om alleen aan de hand van een getal aan te tonen op wat voor manier deze tool goed is. Er zal bij elke test case een toelichting worden gegeven over unieke kenmerken (zowel positief als negatief) hiervan. De weighted scoring methode kan gebruikt worden als leidraad om toch in een oogopzicht te kunnen zien welke tooling in ieder geval afvalt aan de hand van de parameters.

De stappen voor het gebruiken van een *weighted scoring model* die zijn genoemd zullen worden gevolgd in de komende paragrafen (Morpus, 2021a).

8.5 Opsommen van opties

Enkele voorbeelden van tools die getest zijn, zijn als volgt:

SAST	SCA	Secret Detection	Container Scanning	DAST	Linter
Codacy	Dependabot	GitGuardian	Trivy	OWASP Zap	PyLint
CodeQL	WhiteSource Bolt	Gitleaks	Snyk Container Scan	StackHawk	Flake8
SonarCloud	Debricked	TruffleHog	Clair	Arachni	
Semgrep	Snyk Open Source				
Bandit					

Tabel 8.5.1 *Opsommen van opties tools*

De volledige lijst aan tools is te vinden in het test plan.

8.6 Parameters afstellen

Zoals eerder genoemd zijn de parameters opgesteld in fase 1:

- User interface (aanwezigheid hiervan, gebruiksvriendelijkheid)
- Precision, Recall & F-Score (Accuracy)
- Tijd/inspanning opzetten tool
- Middelen (open source, commercieel)

Deze parameters zullen toegelicht worden en nauwkeurig afgebakend worden.
Er is gekozen om een *range* aan waardes/scoring van 1-10 te gebruiken.

8.6.1 User interface

Het belang van een user interface en wat er precies mee wordt bedoeld in het geval van DevSecOps tooling is toegelicht in Subparagraaf 5.9.3.

Voor het scoren/beoordelen van tooling wordt er gekeken naar de volgende kenmerken van een user interface (zie Subparagraaf 5.9.3 voor toelichting):

- *Integratie versiebeheer tooling (Github/GitLab)*
- *Reacties/bot die berichten plaatst op pull requests*
- *‘Leesbaarheid’ meldingen/issues*
- *Dashboard van tool zelf*
- *Suggesties voor oplossen kwetsbaarheden*

Het belangrijkste doel van de user interface is dat het de gebruiker zo makkelijk mogelijk wordt gemaakt om de gevonden kwetsbaarheden in te zien en op te lossen. Op een zo’n efficiënt mogelijke manier. Bovenstaande kenmerken helpen hierbij, echter is het niet altijd nodig om aan alle kenmerken te voldoen.

Er zal als volgt gescoord worden:

Eigenschap	Waarde
Integratie	6
Dashboard	6
Leesbaarheid	3
Reacties/bot	0.5
Suggesties	0.5

Tabel 8.6.1 *User interface eigenschappen scoring*

Tabel 8.6.1 wordt afgelezen bij een test case, aan de hand hiervan is de scoring te bepalen, wanneer een tool bijvoorbeeld aan alle eisen voldoet heeft deze een score van 10 voor User interface (maximale score van 10).

8.6.2 F-Score (Accuracy)

De false positive rate en false negative rate is erg van belang voor DevSecOps (zie Paragraaf 5.10), het aantonen van deze parameters zou gedaan moeten worden aan de hand van een Benchmarking methode. Er is gekozen om te werken met *Performance benchmarks*, dit type benchmarks heeft de volgende kenmerken:

- Werkt op basis van kwantitatieve data
- Data wordt voornamelijk uitgedrukt in getallen, grafieken, diagrammen en tabellen
- Resulteert in een onafhankelijke en accurate beoordeling

Deze kenmerken zijn erg passend bij dit onderzoek, er zijn een aantal elementen nodig om deze benchmark uit te kunnen voeren:

- Standaard opzet om benchmark op uit te voeren
- Manier om de data te extraheren, te verzamelen en te analyseren

Benchmark opzet

Als opzet zal er gebruik gemaakt worden van het PoC dat is geïmplementeerd in fase 3. Dit is de repository *PyGoat* van *OWASP* die gebaseerd is op de *OWASP top 10 2017*:

• *A1:2017-Injection*
• *A2:2017-Broken Authentication*
• *A3:2017-Sensitive Data Exposure*
• *A4:2017-XML External Entities (XXE)*
• *A5:2017-Broken Access Control*
• *A6:2017-Security Misconfiguration*
• *A7:2017-Cross-Site Scripting (XSS)*
• *A8:2017-Insecure Deserialization*
• *A9:2017-Using Components with Known Vulnerabilities*
• *A10:2017-Insufficient Logging & Monitoring*

Het is van belang dat alle vulnerabilities die aanwezig zijn in de repository van tevoren bekend zijn om zo de resultaten te kunnen berekenen.

Categorie DevSecOps tool

Het verschilt per categorie tooling waarop beoordeeld wordt, er zal dan ook voor sommige tooling een alternatieve repository gebruikt moeten worden.

Toelichting Linter
Er wordt een melding gemaakt wanneer een stuk code niet volgens de ingesteld parameters is geschreven. Er zal worden gekeken naar de meldingen of de tool correct aangeeft of de ingestelde code standard gebroken wordt of niet.

Toelichting SAST
SAST tooling zal de repository scannen op kwetsbare stukken code, het doel zal zijn om zoveel mogelijk kwetsbaarheden te vinden in de code uit de *OWASP top 10* lijst die genoemd is.

Toelichting SCA
SCA tooling zal de repository scannen op kwetsbare dependencies, in het geval van de repository *PyGoat* zal de file *requirements.txt* hierop worden gescand.

Toelichting Secret Detection
Secret Detection tooling scant specifiek op secrets, er zal gebruik gemaakt worden van de *OWASP* repository *Wrong Secrets* die zich specialiseert op het verkeerd gebruik van Secrets.

Toelichting Container Scanning
Container scanning tooling scant containers, in het geval van *PyGoat* zal de *Dockerfile* gescand worden op kwetsbaarheden. In dit geval worden de kwetsbare componenten van de containers bedoeld, bijvoorbeeld een verouderde versie van een gebruikte Python library.

Verzamelen en van verwerken data

Het verzamelen en verwerken van data zal worden gedaan aan de hand van tabellen en formules.
SAST Aanwezige kwetsbaarheden (kwetsbaarheden in PyGoat)

OWASP	*Locatie*	*Gevonden*
A1:2017-Injection	*views.py : 70, 334*	
A2:2017-Broken Authentication	*views.py : 378*	
A3:2017-Sensitive Data Exposure	*views.py : 310*	
A4:2017-XML External Entities (XXE)	*views.py : 155*	
A5:2017-Broken Access Control	*views.py : 260*	
A6:2017-Security Misconfiguration	*views.py : 428*	
A7:2017-Cross-Site Scripting (XSS)	*views.py : 49*	
A8:2017-Insecure Deserialization	*views.py : 123*	
A9:2017-Using Components with Known Vulnerabilities	*views.py : 450*	
A10:2017-Insufficient Logging & Monitoring	*views.py : 482*	
A11 (*Next Steps)*	*views.py : 511*	
Server Side Request Forgery (SSRF)	*views.py : 566*	

Tabel 8.6.2 *Aanwezige kwetsbaarheden PyGoat*

SCA Aanwezige kwetsbaarheden

In de *requirements.txt* van *PyGoat* zijn een aantal libraries toegevoegd die vulnerabilities bevatten, er is specifiek gezocht naar de meest voorkomende Python libraries. Alle aanwezige kwetsbare libraries zijn als volgt:

Library	*Versie*	*CVE*	*Gevonden*
pyyaml	5.1	CVE-2020-14343 CVE-2020-1747	
Pillow	8.4.0	CVE-2022-22815 CVE-2022-22817	
urllib3	1.25.6	CVE-2020-7212 CVE-2021-28363	
Nltk	3.4.4	CVE-2019-14751 CVE-2021-43854	
Requests	2.19.1	CVE-2018-18074	
Jinja2	2.10	CVE-2019-8341	

Tabel 8.6.3 *Aanwezige kwetsbaarheden libraries PyGoat*

Secret Detection Aanwezige kwetsbaarheden (OWASP Wrong Secrets)

Voor het benchmarken van Secret Detection tooling is er een samenwerking aangegaan met een Project Leader van het OWASP team. Er was al een issue op de Github pagina van Wrong Secrets met betrekking tot benchmarking, hier is via Slack verder over nagedacht en gediscussieerd. (Link naar Github issue)

Hierbij zijn de volgende stappen genomen:

1. Opsommen van secrets die zich al in het project bevinden
2. Opsommen welk type secrets er nog ontbreken in het project
3. Toevoegen van ontbrekende type secrets

De student heeft het volgende bijgedragen:

1. Scannen van de repository met Secret Detection tooling
2. Meldingen van de tooling bevestigen aan de hand van de repository
3. Doorgeven van bevindingen aan de Project Leader

Secrets in de repository	*Gevonden*
5 Random human rememberable passwords in Git & Docker container	
1 file containing a secret base64 encoded in Docker	
1 random password in Java code with higher entropy	
3 AWS keypairs in git history	
3 secrets in TF state (requires cloud installation)	
1 human readable secret in k8s/secret, 1 in k8s/configmap (requires k8s/cloud installation)	

1 root token for vault after deployment of vault(requires vault&k8s/cloud installation)	
1 random value generated after startup	
1 secret in github action	
1 AES key	
multiple ciphertexts (6)	
1 human readable secret in pw manager file(keepass)	
5 canarytoken-urls in container & code	
multiple secrets in java testing code (of which some used in the actual app)	

Tabel 8.6.4 *Aanwezige Secrets WrongSecrets*

Container scanning aanwezige kwetsbaarheden

Voor het benchmarken van de container scanning tools wordt de Dockerfile in de repository gescand op kwetsbaarheden, deze Dockerfile maakt gebruik van een Python base image. Vervolgens wordt deze gescand op kwetsbare componenten, een aantal kwetsbare componenten hierin zijn als volgt:

Component	*Versie*	*CVE*	*Gevonden*
libaom0	< 2021-03-24	CVE-2021-30474 CVE-2021-30473 CVE-2021-30475	
libde265	1.0.8-1	CVE-2021-35452 CVE-2022-1253	
mariadb	10.5	CVE-2022-27387 CVE-2022-27444 CVE-2022-27445	
unzip	6.0-26	CVE-2022-0530 CVE-2022-0529	
openexr	libopenexr25	CVE-2021-3941 CVE-2021-3933 CVE-2021-3605	
PyYaml	5.1	cve-2019-20477 CVE-2020-14343 cve-2020-1747	
libc-dev	5.10.106-1	CVE-2019-1010022 CVE-2019-1010023	
gzip	1.10-4	cve-2022-1271	
libcurl4	7.74	CVE-2021-22945	
libfreetype6	2.10	CVE-2022-27404	
libwmf	0.2-7	CVE-2015-4695 CVE-2015-0848	
Python	3.9-minimal	CVE-2015-20107	

Tabel 8.6.5 *Aanwezige kwetsbare componenten Container scanning*

8.6.3 Tijd/inspanning

De hoeveelheid tijd en inspanning die het kost om de tooling op te zetten telt ook mee voor de beoordeling van de tool. Er wordt hierbij gelet op de complexiteit van het opstellen van de tool, denk hierbij aan de benodigde handelingen die genomen moeten worden. Waaronder:

- Koppelen Github of GitLab account aan de website van de tooling
- Instellen van parameters van de tooling
- Opspinnen van applicatie voor testing in het geval van DAST tooling

Er zullen 3 niveaus worden genoemd:

Complexiteit	*Betekenis*
Laag	*'Plug & Play', geen verdere handelingen noodzakelijk voor goede functionaliteit*
Gemiddeld	*Enkele aanpassingen aan instellingen (parameters) noodzakelijk voor goede functionaliteit*
Hoog	*Gecompliceerde handelingen nodig voor functionaliteit van de tool of er is tegen problemen aangelopen m.b.t. de werking van de tool*

Tabel 8.6.6 *Complexiteit niveaus*

Laag: *'Plug & Play'* = simpele handeling, alleen het koppelen van de tool aan de repository is voldoende, de rest wordt automatisch afgehandeld.

Gemiddeld: Er zijn integraties aanwezig voor de gebruikte versie-beheer software, bijvoorbeeld kant en klare Github Actions (7.2), workflow-files of andere gebruikshandleidingen van het bedrijf om installatie van de tool makkelijk te maken. Er zijn enkele aanpassingen nodig voor goede functionaliteit.

Hoog: Een hoge complexiteit kan 2 dingen betekenen. *Scenario 1:* er is geen officiële support voor de gebruikte versie-beheer software. Er moet zelf een workflow-file opgesteld worden waarin de stappen worden genomen om de applicatie te scannen met desbetreffende tool. *Scenario 2*: er is officiële support voor integratie met de gebruikte versie-beheer software maar er is of tegen problemen gelopen (tool werkt niet naar behoren na het volgen van de gebruikershandleiding voor de opzet) of de opzet van de tool is gecompliceerd. Dit is bijvoorbeeld het geval bij de meeste DAST tooling waarbij de applicatie moet draaien voor het starten van de test.

Verder zal er worden gekeken naar de tijd die nodig is voor het opstellen van de tool, vervolgens zal er aan de hand van een combinatie van de 3 complexiteit nivo's en de tijd die nodig is voor het opstellen van de tool gescoord worden van 0 tot 10.

8.7 Weight toekennen

De 100% zal verdeeld worden onder de volgende parameters:

- User interface (aanwezigheid hiervan, gebruiksvriendelijkheid)
- F-Score (Accuracy)
- Tijd/inspanning opzetten tool

Weight	*Parameter*
20%	User interface
65%	F-Score (Accuracy)
15%	Tijd/inspanning

Tabel 8.7.1 *Weight toekenning*

Deze verdeling is gedaan aan de hand van de informatie die is opgedaan in het onderzoek in Subparagraaf 5.9.3, het zal per gebruiker verschillen welke parameter meer waarde (weight) zal hebben. Er zal een mogelijk zijn om in de resultaten deze weighting aan te kunnen passen.

8.8 Opzetten weighted scoring grafiek

De informatie die in Paragraaf 8.4-8.7 is verzameld en de tabellen die zijn opgezet zullen moeten worden verwerkt in sheets en tabellen, om deze vervolgens te kunnen gebruiken als functioneel scoringsformulier (test case). De manier waarop de weighted scoring tabellen zijn opgezet zal in deze paragraaf worden toegelicht, ook zal er een voorbeeld worden laten zien van een ingevuld scoringsformulier. De uitgebreide toelichting (met screenshots) is te vinden in het testrapport.

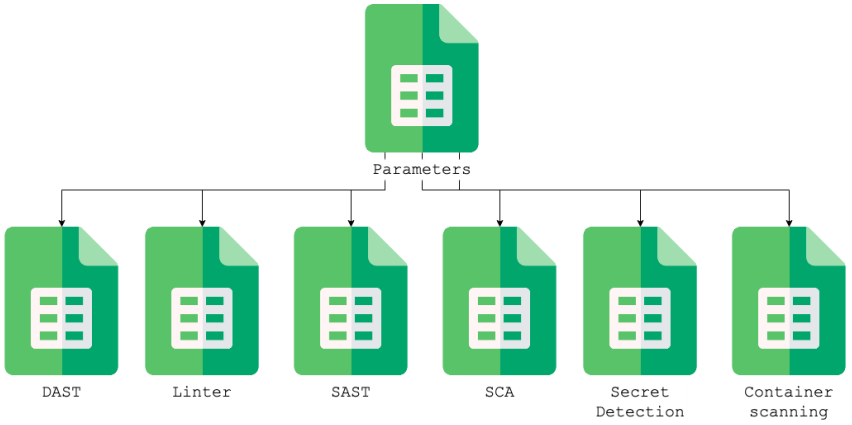
8.8.1 Google Sheets

Er is gekozen om te werken met Google Sheets, deze keuze is gemaakt aan de hand van de volgende punten:

- Volledig vanuit de cloud (Back-ups, makkelijk delen van bestanden)
- Functionaliteit wat betreft formules en genereren van grafieken

De tabellen en sheets die worden behandeld in deze paragraaf zijn opgezet aan de hand van Google Sheets.

8.8.2 Opzet Sheets



Figuur 8.8.1 *Opzet Google Sheets*

Parameters

In de parameters sheet worden de weights van scoringstabellen (test cases) bepaald, deze waardes worden vervolgens geïmporteerd in de overige sheets (bijvoorbeeld DAST).
Op deze manier kunnen de weights later aangepast worden en worden deze changes dynamisch doorgevoerd naar de overige sheets. Dit voorkomt dubbel werk.

	User interface	*F-Score (Accuracy)*	*Tijd/inspanning*
Linter	20.00%	65.00%	15.00%
SAST	20.00%	65.00%	15.00%
SCA	20.00%	65.00%	15.00%
Secret Detection	20.00%	65.00%	15.00%
Container scanning	20.00%	65.00%	15.00%
DAST	20.00%	65.00%	15.00%

Tabel 8.8.2 *Weighting eigenschappen*

Weighted scoring (SAST voorbeeld)
Er is per categorie tooling een template opgezet voor de weighted scoring, in deze templates zijn de tabellen opgenomen die eerder behandeld zijn in Paragraaf 8.6. Het grootste belang is dat zoveel mogelijk automatisch wordt ingevuld/berekend, dit is met Google Sheets zeker haalbaar. Er zijn maar enkele velden die handmatig moeten worden ingevuld.

	Hoeveelheid
True Positives	7
False Positives	8
False Negatives	5
Totaal aantal meldingen	15

Precision	0.4667
Recall (Sensitivity)	0.5833
F-Score (Accuracy)	0.5185

Weighted scoring

	User interface	*F-Score (Accuracy)*	*Tijd/inspanning*	*Totale score*
Weight	20.00%	65.00%	15.00%	100%
Scoring tool	9	5.185185185	9	7.728395062
Weighted scoring	1.8	3.37037	1.35	6.52037037

Codacy
SAST

User interface	9/10
F-Score (Accuracy)	5.2/10
Tijd/inspanning	9/10
Programmeertaal	Talen
Middelen	Commercieel
Ondersteund	✓Github ✓GitLab

6.5
OUT OF 10

Tabel 8.8.3 *Weighted scoring SAST voorbeeld*
De waardes die handmatig ingevuld moeten worden zijn als volgt:

- Tooling naam, versie en datum
- Aanvinken van selectievakjes
- Totaal aantal meldingen
- Ondersteund programmeertalen en middelen

De rest van de waardes worden vervolgens automatisch berekend aan de hand van de ingevulde waardes. Hieruit volgt een ‘weighted’ scoring als resultaat.

8.9 Testen

In deze paragraaf wordt de aanpak van het testen benoemd en de vereisten die hiervoor nodig zijn.

8.9.1 Aanpak testing

De volgende stappen zullen worden genomen tijdens het testen:

1. Tool uit de lijst van opties (Paragraaf 8.5) selecteren, om deze vervolgens te testen
2. Opzetten en opspinnen tooling aan de hand van het opgestelde PoC (Hoofdstuk 7)
3. Tool de repository laten scannen
4. Triage uitvoeren op de resultaten aan de hand van opgezette test cases (8.8)
5. Eventuele toelichtingen of opmerkzaamheden beschrijven

Met triage (stap 3) wordt het controleren en verifiëren van de meldingen uit de testresultaten bedoeld. Er wordt gecontroleerd of de gemelde vulnerabilities daadwerkelijk zorgen voor kwetsbaarheden (True Positive of False Positive). Er zal in stap 4 per tool een nieuwe tab worden aangemaakt in de desbetreffende sheet, wanneer er bijvoorbeeld een SAST tool getest wordt wordt er in de SAST sheet hier een nieuw tabblad voor aangemaakt (op basis van de template).

8.9.2 Vereisten

Tijdens stap 2 *'Opzetten en opspinnen tooling aan de hand van het opgestelde PoC'* zullen de volgende handeling mogelijk nodig zijn voor het kunnen uitvoeren van de tests:

Aanvragen tijdelijk licentie DevSecOps tooling
Voor de commerciële tooling op de lijst aan opties (Paragraaf 8.5) zal er een proeflicentie moeten worden aangevraagd om deze te kunnen testen.

Aanvragen tijdelijk licentie GitLab
Om gebruik te kunnen maken van GitLab tooling zal er een proeflicentie moeten worden aangevraagd.

8.9.3 Ruwe resultaten

De resulterende reports en resultaten die gegenereerd zijn door de CI/CD security tooling (resulterend van de tests die zijn uitgevoerd) zijn bijgesloten in het bijlagenboek.

8.10 Conclusie

Er is een manier opgesteld om de verschillende tooling te 'benchmarken' (scoren) aan de hand van de opgestelde test cases. Deze test cases zijn zo opgesteld dat de meeste waardes automatisch worden berekend. In het volgende hoofdstuk zullen de resultaten van de uitgevoerde tests worden behandeld.

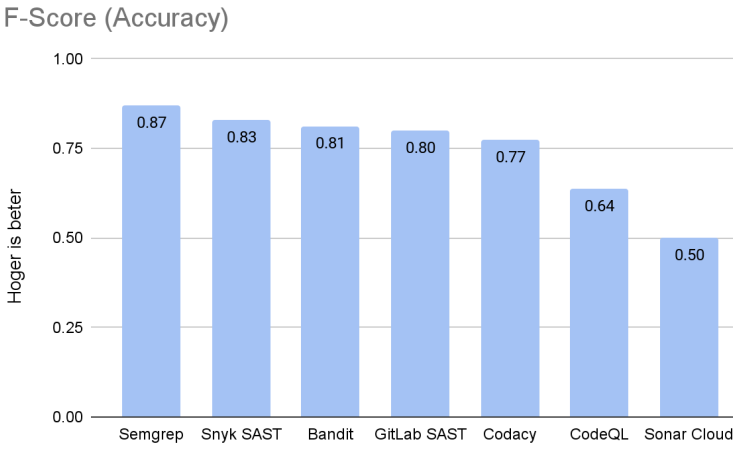
9. Resultaten

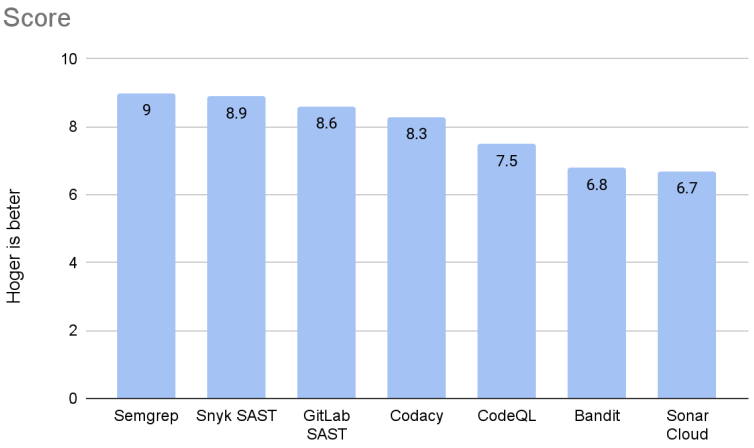
In dit hoofdstuk worden de resultaten van de uitgevoerde test verwerkt, dit zal worden gedaan aan de hand van de testresultaten die zijn weergegeven in het bijlagenboek. De resultaten zijn afgeleid van de opgestelde test cases uit Hoofdstuk 8, in Hoofdstuk 8 is ook een uitgebreide toelichting te vinden over de genoemde waardes in de grafieken. Verder is er een paragraaf discussie waarin opvallende resultaten of toelichtingen wordt gegeven naar aanleiding van de tests.

9.1 Verwerken resultaten

Aan de hand van de uitgevoerde tests zijn per categorie 2 grafieken opgesteld, in de linker grafiek is de F-Score (Accuracy) behandeld en in de rechter grafiek de Score (resultaat van weighted scoring, behandeld in Hoofdstuk 8).

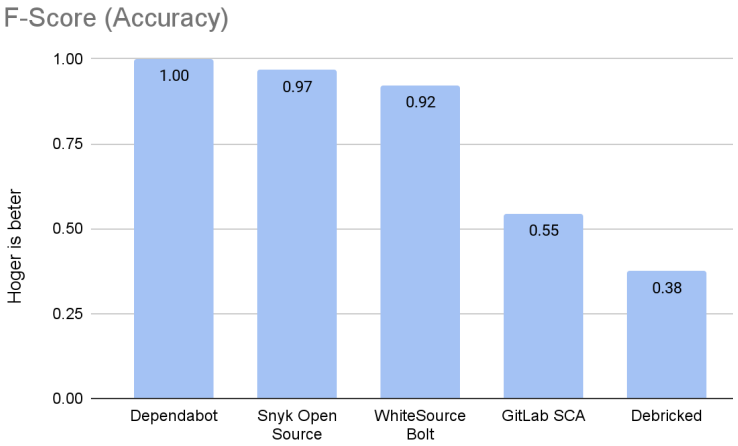
9.1.1 SAST

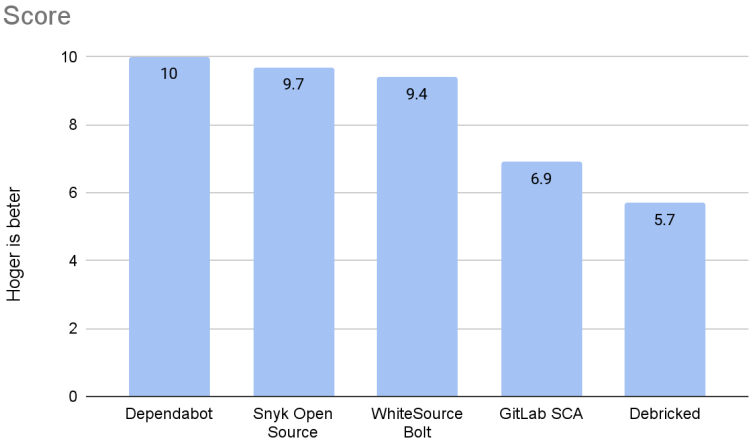




Grafiek 9.1.1 *SAST resultaten*

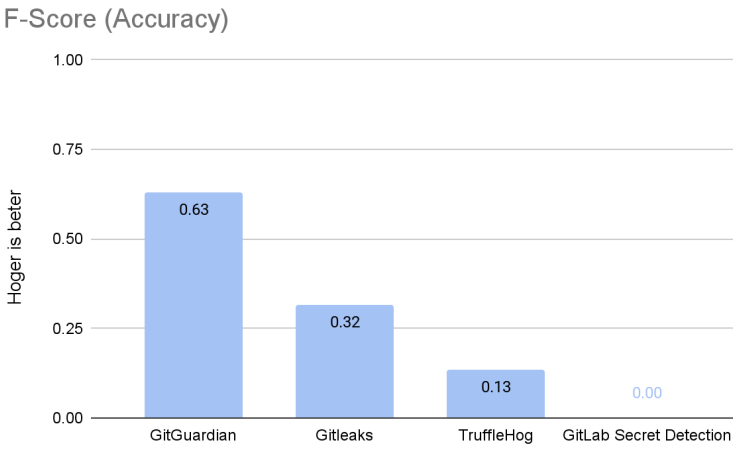
9.1.2 SCA

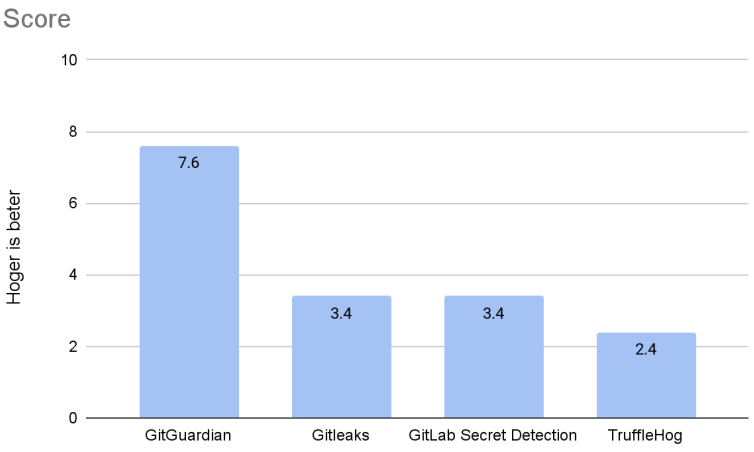




Grafiek 9.1.2 *SCA resultaten*

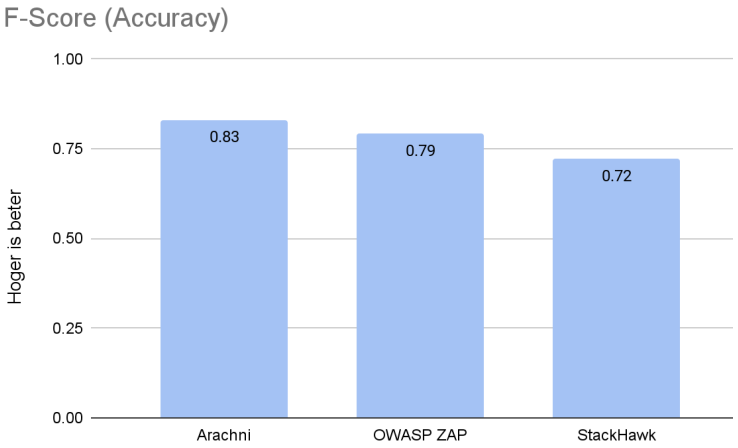
9.1.3 Secret Detection

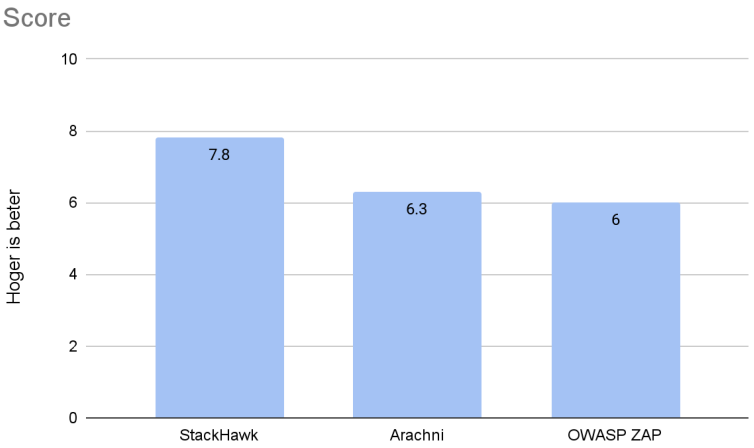




Grafiek 9.1.3 *Secret Detection resultaten*

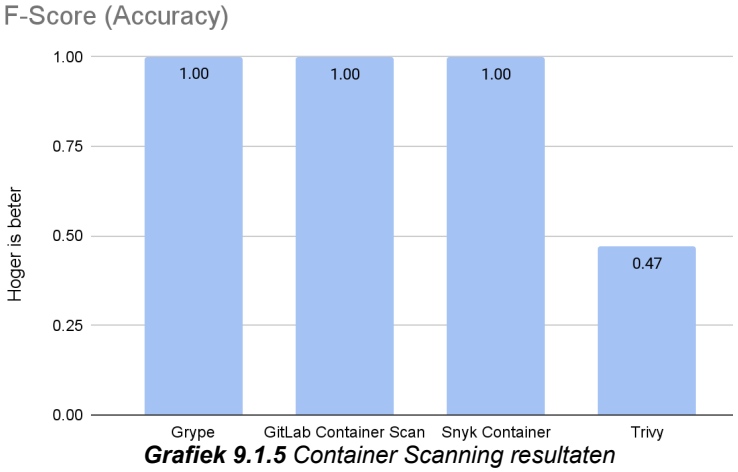
9.1.4 DAST

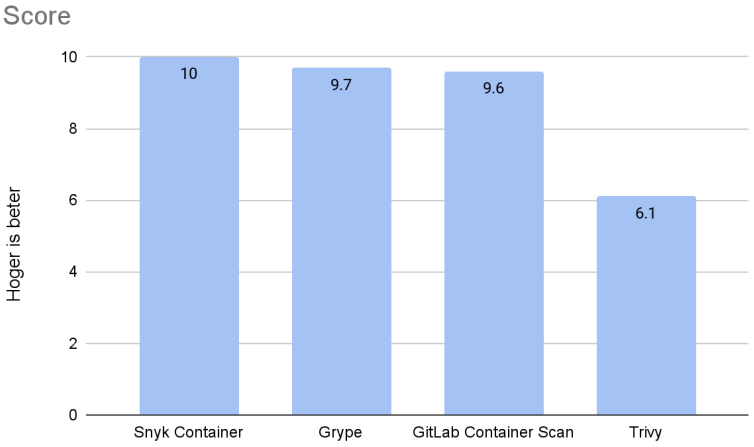




Grafiek 9.1.4 *DAST resultaten*

9.1.5 Container Scanning





Grafiek 9.1.5 *Container Scanning resultaten*

9.1.6 Linter (s)

De geteste linters, Pylint en Flake8, hadden beide een F-Score van 1 en een score van 7.9.

9.2 Discussie

SAST
Tijdens het testen van SAST tools is het gebleken dat er een redelijk groot verschil zit in kwetsbaarheden die wel of niet afgevangen worden door de geteste SAST tool. Echter worden sommige categorieën aan kwetsbaarheden (OWASP top 10) (bijna) altijd vermeld. Een voorbeeld hiervan is de categorie A1:2017-Injection, dit type kwetsbaarheid blijkt 'makkelijk' gedetecteerd door de tools die getest zijn. Tooling die uitblinkt in deze categorie is Semgrep, Snyk SAST en GitLab SAST.

SCA
De geteste SCA tools hebben allemaal bijna alle kwetsbaarheden kunnen vinden, verder waren de tools makkelijk op te zetten (plug & play). De enige tool die er tussenuit sprong was Debricked, deze had 60 false positives, wat vervolgens leidt tot de lage F-Score.

Secret detection
Aan de resultaten van de tests van Secret Detection tools is te zien dat een aantal tools erg veel false negatives hebben. De tools Gitleaks en TruffleHog hebben maar 1 van de 14 kwetsbare secrets kunnen vinden in de repository, dit is erg opmerkelijk aangezien deze tools correct zijn opgezet aan de hand van de beschikbare gebruikershandleiding. Daarentegen heeft de tool GitGuardian een stuk meer kwetsbaarheden gevonden.

DAST
Tijdens het werken met de verschillende DAST tooling is gebleken dat de geteste tools niet allemaal beschikken over Github/GitLab integraties of over een dashboard. De tool die in dit opzicht uitblinkt is StackHawk, deze tool beschikt over een uitgebreid dashboard en steekt dus ver boven de andere tools uit in de behaalde score.

Container scanning
Naar aanleiding van de tests die zijn uitgevoerd met container scanning tools blijkt dat de base image die gebruikt wordt in de Dockerfile (van PyGoat) veel kwetsbare componenten bevat. Deze base image is *python:3.8*, deze zou bijvoorbeeld geupgrade kunnen worden naar -> *python:3.11.0a7-slim* (suggestie van Snyk Container).

Linter(s)
Zoals genoemd is in Sectie 6.2.1 en het onderzoeksrapport is een linter lastig te beoordelen op de F-Score (Accuracy). Er bleek tijdens de tests dat de linter tooling naar behoren werkte en de code correct wordt beoordeeld aan de hand van de styling die standaard staat ingesteld. Dit is ook de reden dat de F-Score van desbetreffende linters dan ook 1 is.

Tooling die niet getest is
Een groot gedeelte tools uit de lijst met opties zijn commerciële tools. Na contact met meerdere bedrijven die dit type tooling aanbiedt bleek dat er geen trial accounts worden aangeboden voor research gerelateerde doeleinden. De trial licenties zijn alleen bedoeld voor bedrijven die potentieel een klant zouden zijn. Naast een aantal commerciële tools die niet getest konden worden waren er een aantal tools die bijvoorbeeld verouderd waren of om een andere reden niet zijn meegenomen in de tests.

10. Conclusie

10.1 Advies

Per categorie tooling is onderzocht welke CI/CD security tools het meest efficiënt/effectief zijn, de resultaten van dit onderzoek zullen worden gepresenteerd als advies aan Computest. Dit advies is gebaseerd op de resultaten uit Hoofdstuk 9. Enkele scorekaarten zullen worden weergegeven ter beeldvorming, alle scorekaarten zijn te vinden in de bijlagen.

10.1.1 SAST

Voor SAST tooling wordt aangeraden om gebruik te maken van Semgrep of Snyk SAST, deze tools hebben hoog gescoord aan de hand van de test cases. Beide tools beschikken over een dashboard met uitgebreide functionaliteiten waaronder user management en integraties met onder andere Github en GitLab.

Snyk Code Analysis SAST		
User interface	10/10	**8.9** OUT OF 10
F-Score (Accuracy)	8.3/10	
Tijd/inspanning	10/10	
Programmeertaal	link	
Middelen	Commercieel	
Ondersteund	✓Github ✓GitLab	

Tabel 10.1.1 *Snyk SAST scorekaart*

Semgrep SAST		
User interface	9/10	**9.0** OUT OF 10
F-Score (Accuracy)	8.7/10	
Tijd/inspanning	10/10	
Programmeertaal	link	
Middelen	Commercieel	
Ondersteund	✓Github ✓GitLab	

Tabel 10.1.2 *Semgrep scorekaart*

Verder kost het opzetten van beide tools weinig tijd.

10.1.2 SCA

SCA tools die goed gescoord hebben zijn Dependabot, Snyk Open Source en WhiteSource Bolt. Wanneer er gebruik gemaakt wordt van Github wordt er aangeraden om Dependabot te gebruiken, in het geval van GitLab worden Snyk Open Source en WhiteSource Bolt aangeraden.

Snyk Open Source SCA		
User interface	9.5/10	**9.7** OUT OF 10
F-Score (Accuracy)	9.7/10	
Tijd/inspanning	10/10	
Programmeertaal	link	
Middelen	Commercieel	
Ondersteund	✓Github ✓GitLab	

Tabel 10.1.3 *Snyk SCA scorekaart*

WhiteSource Bolt SCA		
User interface	9.5/10	**9.4** OUT OF 10
F-Score (Accuracy)	9.2/10	
Tijd/inspanning	10/10	
Programmeertaal	link	
Middelen	Commercieel	
Ondersteund	✓Github ✓GitLab*	

*(self managed only)

Tabel 10.1.4 *WhiteSource Bolt scorekaart*

10.1.3 Secret Detection

Voor Secret Detection wordt de tool GitGuardian aangeraden, deze tool beschikt over een uitgebreid dashboard. Verder is het opzetten van de tool erg gemakkelijk en zijn er integratie mogelijkheden aanwezig voor (bijna) alle gebruikte CI/CD pipeline tools (waaronder Github en GitLab).

GitGuardian **Secret Detection**		
User interface	10/10	**7.6** **OUT OF 10**
F-Score (Accuracy)	6.3/10	
Tijd/inspanning	10/10	
Programmeertaal	*Language-agnostic*	
Middelen	Commercieel	
Ondersteund	✓Github ✓GitLab	

Tabel 10.1.5 *GitGuardian scorekaart*

10.1.4 DAST

De DAST tool StackHawk wordt aangeraden, StackHawk maakt onder water gebruik van de DAST tool OWASP ZAP en beschikt over een uitgebreid dashboard en integratie met zowel Github als GitLab. Door deze integratie kost het opzetten van de tooling relatief weinig tijd (ten opzichte van de overige geteste tooling).

StackHawk **DAST**		
User interface	9.5/10	**7.8** **OUT OF 10**
F-Score (Accuracy)	7.2/10	
Tijd/inspanning	8/10	
Programmeertaal	*Language agnostic*	
Middelen	Commercieel	
Ondersteund	✓Github ✓GitLab	

Tabel 10.1.6 *StackHawk scorekaart*

10.1.5 Container Scanning

De Container Scanning tools die worden aangeraden zijn Snyk Container, Grype en GitLab Container Scan. De Container Scanning tool van GitLab is alleen beschikbaar voor GitLab.

Snyk Container **Container scanning**		
User interface	10/10	**10.0** **OUT OF 10**
F-Score (Accuracy)	10/10	
Tijd/inspanning	10/10	
Programmeertaal	link	
Middelen	Commercieel	
Ondersteund	✓Github ✓GitLab	

Tabel 10.1.7 *Snyk Container scorekaart*

Grype **Container scanning**		
User interface	9/10	**9.7** **OUT OF 10**
F-Score (Accuracy)	10/10	
Tijd/inspanning	9/10	
Programmeertaal	link	
Middelen	Open source	
Ondersteund	✓Github ✓GitLab	

Tabel 10.1.8 *Grype scorekaart*

10.1.6 Linter(s)

In het geval van linter tooling wordt bij het gebruik van Python aangeraden om de tooling Pylint of Flake8 te gebruiken. Uit de resultaten blijkt dat deze tooling goed functioneert. In het geval van andere programmeertalen wordt aangeraden om een bijhorende linter te gebruiken.

10.1.7 Complete oplossing: Snyk

Uit de resultaten blijkt dat de tooling die aangeboden wordt door Snyk in de categorieën SAST, SCA en Container Scanning erg hoog scoort. Wanneer er wordt overwogen om een

combinatie van SAST, SCA en container scanning tools in gebruik te nemen wordt aangeraden om een pakket van Snyk in gebruik te nemen. Hierdoor zijn de meldingen en overige handelingen te beheren vanuit één centraal dashboard, die van Snyk. Eén centraal punt voor de meldingen van deze tools kan de efficiëntie bevorderen, aangezien er op deze manier een duidelijk overzicht is van alle gevonden problemen.

10.2 Samenvattende conclusie

De hoofdvraag van het project is beantwoord, aan de hand van de fases die zijn doorgelopen (Paragraaf 4.1) is er tot een antwoordt gekomen op de vraag:

> *''Welke CI/CD security tools binnen open-source-gemeenschappen bewijzen het meest efficiënt/effectief te zijn in CI/CD pipelines?"*

Het antwoord op deze vraag is in dit hoofdstuk behandeld, er is een advies gegeven met behulp van onderzoek, tests en de hieruit volgende resultaten.

In open-source projecten worden sommige categorieën DevSecOps tools veel gebruikt (zoals Linter(s) en SAST tooling), terwijl andere categorieën tools minder veel voorkwamen (zoals Secret Detection, Container Scanning, SCA en DAST) (Paragraaf 5.6). Verder is het duidelijk geworden welke specifieke DevSecOps tools er gebruikt worden in open-source projecten, voorbeelden hiervan zijn te vinden in Paragraaf 5.5.

Het is van belang hoe bevindingen van DevSecOps tools worden afgehandeld (Paragraaf 5.7). De prioriteit van het afhandelen van de bevinding zou afhankelijk moeten zijn van de impact van de gevonden kwetsbaarheid. De manier van afhandeling varieert per open-source project, bij sommige projecten worden meldingen van tools direct verwerkt, terwijl hier bij sommige projecten niet op gereageerd wordt. De ingebruikname van DevSecOps tooling bij open-source projecten kan vlekkeloos gaan, soms komt hier extra werk bij kijken zoals het moeten aanpassen van source code om de meldingen op te lossen (Paragraaf 5.8).

Naar aanleiding van een literatuurstudie is gebleken dat er meerdere elementen zijn van DevSecOps tools die van belang zijn voor de effectiviteit/efficiëntie. Hieronder valt de user interface, precision, tijd/inspanning en de nodige middelen (Paragraaf 5.9 tot en met 5.11).

10.3 Vervolgonderzoek

Er zijn een aantal punten die bij verder onderzoek van het onderwerp meegenomen zouden kunnen worden. Dit zijn de volgende punten:

- Verder onderzoek naar een omvangrijker PoC met meer diepgaande test cases (verder toegelicht in Paragraaf 11.2)
- Contact met commerciële bedrijven voor eventuele toegang tot hun tooling (9.2)
- Onderzoek naar Infrastructure as Code (IaC) (bijvoorbeeld van Snyk), voor het oplossen van eventuele misconfiguraties in cloud-omgevingen. Dit zou een interessante zijtak van DevSecOps zijn om verder te onderzoeken, zeker met de ontwikkelingen op het gebied van onder andere Microsoft Azure.

11. Project evaluatie

In dit hoofdstuk wordt geëvalueerd over zowel het proces als het product, verder zal er worden ingegaan op de competenties die zijn opgesteld.

11.1 Proces evaluatie

Over het algemeen ben ik tevreden over hoe het traject procesmatig is gegaan, de methodiek die is gevolgd bleek functioneel. Door gebruik te maken van de Sashimi watervalmethode was het mogelijk aan het begin van de ontwerpfase nog terug te vallen op een stuk analyse.

Mocht ik het stagetraject opnieuw kunnen volgen met de kennis die ik heb opgedaan zou ik nogmaals gebruik maken van dezelfde methodiek.

De communicatie tussen zowel de bedrijfsbegeleider van Computest, de stagebegeleider en expert examinator van de studie is ook goed verlopen. Hier zijn verder geen opmerkzaamheden over te benoemen.

11.2 Product evaluatie

Voor de tijd die beschikbaar was voor het traject ben ik tevreden met de geleverde producten en het eindresultaat. Er is succesvol een analyse uitgevoerd en een functioneel PoC ontworpen en opgezet die vervolgens is gebruikt om aan de hand van opgestelde test cases verschillende tools te beoordelen.

Als er meer tijd ter beschikking was geweest zijn er een aantal verbeterpunten die tot een beter product zouden kunnen leiden. De testopstelling van het PoC zou verder uitgewerkt kunnen worden om zo preciezere testresultaten te kunnen krijgen.

De huidige opstelling voor het PoC maakt gebruik van de repository PyGoat. Er zou dieper ingedoken kunnen worden op de OWASP top 10 om zelf een stuk code te schrijven die gebruikt zou kunnen worden voor de test cases van SAST tools. Of er zou een grotere repository gebruikt kunnen worden (bijvoorbeeld de OWASP-Benchmark repository), het nadeel hiervan is dat de analyse van de resultaten (triage) erg veel tijd in beslag zou nemen. In combinatie met de grote hoeveelheid tools die getest zou moeten worden zou dit veel tijd kosten. Zo zou de testopstelling per categorie (SAST/SCA etc.) verder uitgebreid kunnen worden, in het geval van SCA zouden er bijvoorbeeld meer kwetsbare libraries/componenten getest kunnen worden. Mocht er meer tijd beschikbaar zijn geweest zou ik deze dus besteden aan het verder uitwerken van het PoC.

11.3 Competenties

A1 Analyseren probleemdomein & opstellen probleemstelling
Het is van belang dat het probleemdomein in kaart wordt gebracht, de uitgangssituatie duidelijk wordt weergeven en de risico's worden geëvalueerd.

Deze competentie is aangetoond aan de hand van het opgestelde plan van aanpak, ook is een analyse van het probleemdomein en de opstelling van de probleemstelling te vinden in Hoofdstuk 3.

B1 Gemotiveerd selecteren van ICT-gerelateerde oplossingen
Het selecteren van een ICT-gerelateerde oplossing stond in dit project centraal, dit is ook terug te vinden in de hoofdvraag. Er is te werk gegaan om aan de hand van de opgestelde methodiek een gemotiveerde keuze te kunnen maken voor security tools. De selectie die gemaakt is in Hoofdstuk 10 is dan ook gedaan aan de hand van analyse, het ontworpen en geïmplementeerde PoC en de opgestelde test cases. In Hoofdstuk 9 zijn de resultaten te zien die uiteindelijk hebben geleid tot de gemotiveerde keuze.

D2 Testen & Evalueren
Testen en evalueren waren een ander belangrijk punt in het project, er zijn gedetailleerde test cases opgesteld (te zien in Hoofdstuk 8) die vervolgens zijn gebruikt om een aantal tools te kunnen evalueren. Er zijn per categorie aan security tools voor CI/CD pipelines specifieke test cases opgesteld, in het geval van SAST tooling is er bijvoorbeeld gebruik gemaakt van de OWASP top 10 (voor meer toelichting kan er in Hoofdstuk 8 worden gekeken).

D4 Configureren
Voor het opstellen van het PoC waren er uitgebreide configuraties van pas gekomen, naast configuraties voor de geteste security tools was er ook configuratie nodig voor de platformen zoals Github Actions en GitLab CI/CD. Deze configuraties zijn terug te vinden in het bijlagenboek. Een voorbeeld van een desbetreffende configuratie is te zien in Hoofdstuk 7 (de complete config staat in het bijlagenboek).

Ga Effectief (internationaal) communiceren
Communicatie is een essentieel onderdeel van elk project, ook bij dit project was dit een belangrijk onderdeel. De communicatie is goed verlopen, aan de hand van wekelijkse meetings met de bedrijfsbegeleider van Computest was het aan beide kanten duidelijk wat er van elkaar verwacht werd. Verder liep de communicatie met de stagebegeleider en expert examinator van de studie ook soepel, ik heb initiatief genomen in het afspreken en opzetten van meetings (waaronder het bedrijfsbezoek, de conceptbespreking en het TTA).

Gc Kritisch, onderzoekend en methodisch werken
Methodisch werken was in het project erg van belang, dit aangezien de opdracht waaraan gewerkt werd erg grootschalig bleek te zijn. Door methodisch te werken wordt voorkomen dat er teveel tijd wordt besteed aan één onderdeel en er in tijdsnood wordt gekomen later in het project. Methodes die gebruikt zijn in het project zijn te vinden in Hoofdstuk 4.

Gf Leren leren: voorbereiden op volgende studiefase en beroep
Een belangrijk doel van de afstudeerstage is het voorbereiden op de fase na de stage, in mijn geval zou dit de beroepsfase zijn. Door fysiek aanwezig te zijn op kantoor en actief mee te doen met het security team waar ik contact mee heb is er goed voorbereid op deze fase. Dit is gedaan door onder andere mee te doen met de dagelijkse stand ups, deel te nemen aan kennis-deelsessies en overige activiteiten binnen Computest.

12. Bibliografie

Voor het noteren van de bronvermelding is APA 7de editie gebruikt

Atlassian. (z.d.-a). *DevSecOps* [Illustratie]. https://www.atlassian.com/devops/devops-tools/devsecops-tools

Atlassian. (z.d.-b). *DevSecOps Tools*. Geraadpleegd op 10 februari 2022, van https://www.atlassian.com/devops/devops-tools/devsecops-tools

Blender. (z.d.-a). *Buildbot Blender* [Screenshot]. Buildbot Blender. https://builder.blender.org/admin/#/

Blender. (z.d.-b). *Tools - Blender Developer Wiki*. Blender Wiki. Geraadpleegd op 16 februari 2022, van https://wiki.blender.org/wiki/Tools

Codacy. (z.d.). *Codacy*. Geraadpleegd op 15 februari 2022, van https://www.codacy.com/about

Codacy. (2020, 7 augustus). *Engines - Codacy docs*. Codacy Docs. Geraadpleegd op 26 april 2022, van https://docs.codacy.com/v2.0/related-tools/engines/

Computest. (z.d.). *Computest*. Geraadpleegd op 14 februari 2022, van https://www.computest.nl/nl/

Computest. (2022, 11 februari). Functional Test Automation [Presentatie]. Functional Test Automation, Zoetermeer, Nederland.

Github. (z.d.-a). *Overview Github Actions* [Illustratie]. Github Docs. https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions

Github. (z.d.-b). *Saving repositories with stars.* Github Docs. Geraadpleegd op 22 februari 2022, van https://docs.github.com/en/get-started/exploring-projects-on-github/saving-repositories-with-stars

Github. (z.d.-c). *Understanding GitHub Actions*. Github Docs. Geraadpleegd op 10 maart 2022, van https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions

GitLab. (z.d.-a). *Auto DevOps*. GitLab Docs. Geraadpleegd op 10 maart 2022, van https://docs.gitlab.com/ee/topics/autodevops/

GitLab. (z.d.-b). *Pricing*. GitLab About. Geraadpleegd op 10 maart 2022, van https://about.gitlab.com/pricing/

GitLab. (z.d.-c). *Secure your application | GitLab*. Geraadpleegd op 12 februari 2022, van https://docs.gitlab.com/ee/user/application_security/

Google. (2020, 10 februari). *Classification: True vs. False and Positive vs. Negative*. Google Developers. Geraadpleegd op 24 februari 2022, van https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative

Klogix. (2020, 25 juni). *Shift Left* [Illustratie]. klogixsecurity. https://www.klogixsecurity.com/blog/shift-left-the-rise-of-devsecops

Morpus, N. (2021a, januari 16). *A Step-by-Step Guide for Using a Weighted Scoring Model*. The Blueprint. Geraadpleegd op 17 maart 2022, van https://www.fool.com/the-blueprint/weighted-scoring-model/

Morpus, N. (2021b, januari 16). *Weighted scoring model* [Illustratie]. The Blueprint. https://www.fool.com/the-blueprint/weighted-scoring-model/

Morpus, N. (2021c, januari 16). *Weighted scoring model* [Tabel]. The Blueprint. https://www.fool.com/the-blueprint/weighted-scoring-model/

Mozilla. (z.d.-a). *Code quality — Firefox Source Docs documentation*. Firefox Source Docs. Geraadpleegd op 16 februari 2022, van https://firefox-source-docs.mozilla.org/code-quality/index.html

Mozilla. (z.d.-b). *GitHub - mozilla/code-review*. Mozilla Code-Review. Geraadpleegd op 16 februari 2022, van https://github.com/mozilla/code-review

Mozilla. (z.d.-c). *Mozilla reviewbot* [Screenshot]. Phabricator mozilla reviewbot. https://phabricator.services.mozilla.com/p/reviewbot/

Mozilla. (2022, 28 januari). *Differential D137274*. Phabricator services mozilla. Geraadpleegd op 23 februari 2022, van https://phabricator.services.mozilla.com/D137274

NIST. (2022, 11 januari). *About*. Geraadpleegd op 6 april 2022, van https://www.nist.gov/about-nist

NSA. (2011). CAS Static Analysis Tool Study - Methodology. *CAS Static Analysis Tool Study - Methodology*, 3–4. https://samate.nist.gov/docs/CAS_2011_SA_Tool_Method.pdf

OWASP. (z.d.). *OWASP Zed Attack Proxy (ZAP)*. Zaproxy. Geraadpleegd op 15 februari 2022, van https://www.zaproxy.org/

OWASP Foundation. (z.d.-a). *GitHub - adeyosemanputra/pygoat: intentionally vuln web Application Security in django*. GitHub. Geraadpleegd op 2 maart 2022, van https://github.com/adeyosemanputra/pygoat

OWASP Foundation. (z.d.-b). *GitHub - commjoen/wrongsecrets: Examples with how to not use secrets*. GitHub. Geraadpleegd op 2 maart 2022, van https://github.com/commjoen/wrongsecrets

OWASP Foundation. (z.d.-c). *OWASP Foundation | Open Source Foundation for Application Security*. OWASP. Geraadpleegd op 24 februari 2022, van https://owasp.org/

OWASP Foundation. (z.d.-d). *OWASP Vulnerable Web Applications Directory*. OWASP. Geraadpleegd op 2 maart 2022, van https://owasp.org/www-project-vulnerable-web-applications-directory/

OWASP Foundation. (z.d.-e). *XML External Entity (XXE) Processing*. OWASP. Geraadpleegd op 22 februari 2022, van https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing

Pennington, J. P. (2019, 18 juli). *Shifting Left: DevSecOps as an Approach to Building Secure Applications* [Illustratie]. Medium. https://medium.com/taptuit/shifting-left-devsecops-as-an-approach-to-building-secure-products-3a418fbbafbe

Python Software Foundation. (z.d.). *XML Processing Modules — Python 3.10.4 documentation*. Python documentation. Geraadpleegd op 22 februari 2022, van https://docs.python.org/3/library/xml.html#xml-vulnerabilities

Red Hat. (2019, 8 januari). *What is a CI/CD pipeline?* Geraadpleegd op 9 februari 2022, van https://www.redhat.com/en/topics/devops/what-cicd-pipeline

Snyk. (z.d.). *Snyk | Developer security | Develop fast. Stay secure.* Geraadpleegd op 15 februari 2022, van https://snyk.io/

Snyk. (2021, 3 augustus). *3 parameters to measure SAST testing*. Geraadpleegd op 6 april 2022, van https://snyk.io/blog/parameters-to-measure-sast-testing/

Synopsys. (2016, 7 maart). SAST vs. DAST: What's the best method for application security testing? Geraadpleegd op 12 februari 2022, van https://www.synopsys.com/blogs/software-security/sast-vs-dast-difference/

Van Dale. (2022a). effectief. In *Van Dale*. https://www.vandale.nl/

Van Dale. (2022b). efficient. In *Van Dale*. https://www.vandale.nl/

WebFinance Inc. (2016). Proof of Concept. Proof of Concept. http://www.investorwords.com/3899/proof_of_concept.html

WhiteSource Software. (z.d.). *WhiteSource Bolt: Find & Fix Open Source vulnerabilities*. WhiteSource. Geraadpleegd op 15 februari 2022, van https://www.whitesourcesoftware.com/free-developer-tools/bolt/

Xopero Software S.A. (2021, 8 december). Top Git hosting services for 2022. GitProtect Blog. Geraadpleegd op 16 februari 2022, van https://gitprotect.io/blog/top-git-hosting-services-for-2022/

13. Woordenlijst

Begrip	Definitie
DevOps	Reeks practises die software-ontwikkeling (Dev) en IT (Ops) combineert.
DevOps-pipeline	Reeks aan geautomatiseerde processen en tools die ontwikkelaars toelaat samen te werken aan het opbouwen en uitbrengen van code naar een productie omgeving
CI/CD-pipeline	Serie aan stappen die uitgevoerd moeten worden voordat een nieuwe versie van een software uitgebracht kan worden
DevSecOps	Integreren van security in DevOps
Open-source	Software waarvan de originele broncode vrij beschikbaar is gesteld en kan worden gedistribueerd en gewijzigd
False-positive	Een testresultaat dat ten onrechte aangeeft dat een bepaalde eigenschap aanwezig is
False-negative	Een testresultaat dat ten onrechte aangeeft dat een bepaalde eigenschap ontbreekt
Penetration test	Gesimuleerde cyber-aanval op een computer systeem, uitgevoerd om de security van het systeem op proef te stellen
Github	Version-control platform wat gebruik maakt van Git
GitLab	Version-control platform wat gebruik maakt van Git
Commit	Vermelden aanpassingen aan een repository
Pull request	Overzicht van wijzigingen aan de repository die in behandeling zijn
Push	Lokale repository-inhoud naar externe repository uploaden
Repository	Virtuele opslag van een project, gebruikt om versies van code op te slaan
Bot	Autonoom programma wat kan samenwerken met andere systemen of programma's
Dependency	Afhankelijkheid van een stuk software aan een ander stuk software, software A is afhankelijk van software B

14. Afkortingenlijst

Afkorting	Definitie
DevSecOps	Development, security and operations
CI/CD	Continuous integration, continuous delivery
PoC	Proof of Concept
Pentest	Penetration test
FPR	False Positive Rate
FNR	False Negative Rate
SCA	Software Composition Analysis
DAST	Dynamic Application Security Testing
API	Application Programming Interface
SAST	Static Application Security Testing
OWASP	The Open Web Application Security Project